



 Latest updates: <https://dl.acm.org/doi/10.1145/3748173.3779200>

RESEARCH-ARTICLE

## HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs

CHANG SUN, California Institute of Technology, Pasadena, CA, United States

ZHIQIANG QUE, Imperial College London, London, U.K.

THEA AARRESTAD, Swiss Federal Institute of Technology, Zurich, Zurich, ZH, Switzerland

VLADIMIR LONCAR, European Organization for Nuclear Research, Geneva, GE, Switzerland

JENNIFER NGADIUBA, Fermi National Accelerator Laboratory, Batavia, IL, United States

WAYNE W C LUK, Imperial College London, London, U.K.

[View all](#)

Open Access Support provided by:

[Fermi National Accelerator Laboratory](#)

[California Institute of Technology](#)

[Imperial College London](#)

[European Organization for Nuclear Research](#)

[Swiss Federal Institute of Technology, Zurich](#)



PDF Download  
3748173.3779200.pdf  
23 February 2026  
Total Citations: 1  
Total Downloads: 0



Published: 22 February 2026

[Citation in BibTeX format](#)

FPGA '26: The 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays  
February 22 - 24, 2026  
CA, Seaside, USA

Conference Sponsors:  
SIGDA



# HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs

Chang Sun\*  
California Institute of Technology  
Pasadena, California, USA  
chsun@cern.ch

Zhiqiang Que  
Imperial College London  
London, UK  
z.que@imperial.ac.uk

Thea Årrestad  
ETH Zurich  
Zurich, Zurich, Switzerland  
thea.aarrestad@cern.ch

Vladimir Loncar†  
Institute of Physics Belgrade  
Belgrade, Serbia  
vloncar@ipb.ac.rs

Jennifer Ngadiuba  
Fermilab  
Batavia, Illinois, USA  
jennifer.ngadiuba@cern.ch

Wayne Luk  
Imperial College London  
London, UK  
w.luk@imperial.ac.uk

Maria Spiropulu  
California Institute of Technology  
Pasadena, California, USA  
smaria@caltech.edu

## Abstract

Neural networks with sub-microsecond inference latency are required by many critical applications. Targeting such applications deployed on FPGAs, we present High Granularity Quantization (HGQ), a quantization-aware training framework that optimizes parameter bit-widths through gradient descent. Unlike conventional methods, HGQ determines the optimal bit-width for each parameter independently, making it suitable for hardware platforms supporting heterogeneous arbitrary precision arithmetic. In our experiments, HGQ shows superior performance compared to existing network compression methods, achieving orders of magnitude reduction in resource consumption and latency while maintaining the accuracy on several benchmark tasks. These improvements enable the deployment of complex models previously infeasible due to resource or latency constraints. HGQ is open-source<sup>1</sup> and is used for developing next-generation trigger systems at the CERN ATLAS and CMS experiments for particle physics, enabling the use of advanced machine learning models for real-time data selection with sub-microsecond latency.

## CCS Concepts

• **Hardware** → **Hardware-software codesign**; • **Computing methodologies** → **Neural networks**; • **Applied computing** → **Physics**.

\*Corresponding author; Partial work done while at ETH Zurich.

†Also with CERN.

<sup>1</sup><https://github.com/caladoi/hgq2>

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only. Request permissions from owner/author(s).

FPGA '26, Seaside, CA, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2079-6/2026/02

<https://doi.org/10.1145/3748173.3779200>

## Keywords

Quantization-aware training, FPGA, Real-time inference, Neural networks, Hardware-software codesign, Low-latency, Quantization

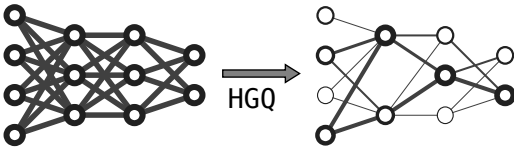
## ACM Reference Format:

Chang Sun, Zhiqiang Que, Thea Årrestad, Vladimir Loncar, Jennifer Ngadiuba, Wayne Luk, and Maria Spiropulu. 2026. HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '26)*, February 22–24, 2026, Seaside, CA, USA. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3748173.3779200>

## 1 Introduction

Edge computing has significantly increased the importance of real-time Deep Neural Network (DNN) inference on specialized hardware [76]. While the typical latency threshold for real-time inference applications is  $O(1)$  ms [39, 60, 102], some critical applications need few-microsecond to sub-microsecond inference latency with high throughput requirements. The most prominent example of such applications is the level-1 *trigger*, a real-time data filtering system at the CERN Large Hadron Collider (LHC) [87] experiment, which requires end-to-end decision latencies not exceeding a few microseconds [83, 86]. Similar latency and resource constraints also apply to other accelerator experiments, such as Belle II [84, 85], where a proposed neural network-based trigger of  $\sim 300$ ns latency is required [46, 59].

Similar requirements are also found in other scientific domains, such as neutrino and dark matter searches [27], quantum circuit control [30], gravitational wave detection [71] and magnetic confinement fusion control [26]. Outside the scientific computing domain, real-time inference with (sub-)microsecond latency is needed in many applications, such as event cameras [21, 37, 43, 51, 74], radar signal processing [100], augmented/virtual reality applications [52], and high-frequency trading [41, 56]. However, it is extremely challenging to achieve (sub-)microsecond end-to-end latency for DNN inference on traditional hardware. Accuracy typically benefits from larger, deeper models, whereas ultra-low latency requires highly



**Figure 1: An illustration of the HGQ quantization scheme on weights and activations. In this example, each weight and activation has its own learnable bit-width. Parameters with zero bit-width are effectively pruned.**

parallel, even fully unrolled, implementation on specialized hardware, like FPGAs. While using FPGAs mitigates such latency requirements, it can lead to high resource consumption, creating a tension between model size and hardware cost. This tension is often acute in scientific and control applications, e.g., for L1 triggers at colliders and fast control loops, where both latency and throughput constraints must be met simultaneously under hard resource caps.

To address such challenges, recent work explores several directions. First, fully unrolled pipelines can deliver large latency reductions by spatially mapping every operation, but they scale poorly in resources as models grow [33, 88]. In the same spirit of pushing logic-level parallelism, LUT-based DNNs map neurons or subfunctions to learned logical look-up operations which are further mapped to physical LUT primitives, yielding DSP-free, deeply pipelined sub- $\mu$ s designs on compact models. However, existing efforts mostly target Multilayer Perceptron (MLP) networks at small scales, typically at most a few thousand learned LUTs, confine computation to LUTs which underuses other on-chip resources such as DSPs and increases routing pressure as models scale, and rely on largely AMD/Xilinx-centric toolflows [6, 8, 16, 42, 94, 98], leaving scalability, architectural breadth, and cross-vendor portability as open challenges.

Quantization-aware training (QAT) is often used to improve hardware efficiency for neural networks, and frameworks such as QKeras [22] are widely used for training FPGA-targeted DNNs with fixed-point quantization. However, the compression ratio is limited with QAT when the model performance degradation is required to be small, and the choice of bit-widths itself becomes an important hyperparameter that needs to be optimized. Differentiable quantization, such as DiffQ [34] has been proposed to optimize the bit-widths by parametrized noise injection, and LSQ [35] uses a straight-through estimator (STE) to optimize the quantization scale. However, these methods still operate at coarse granularity, i.e., per-layer or per-channel, and do not incorporate FPGA-targeted resource models. Moreover, these methods also come with their own challenges, such as extreme instability under low bit-widths [34], or hardware-unfriendly quantization schemes [35]. Consequently, there remains a gap for methods that (a) learn bit-widths at high granularity and (b) explicitly trade off task accuracy against on-chip cost for FPGA-targeted DNNs.

To address this gap, we propose High Granularity Quantization (HGQ), a QAT method tailored for FPGA-targeted neural networks. HGQ has two key components: (i) differentiable fixed-point quantization with learnable bit-widths optimized at an arbitrarily fine granularity (i.e., up to per-parameter, but can also be per channel/kernel/tile). An intuitive illustration of the per-weight and per-activation quantization is shown in Fig. 1. During training, we relax

inherently discrete bit-widths to continuous surrogates, attach gradients to these surrogates, and round to integers in the forward pass before quantization so that standard gradient descent can allocate higher precision only where it most benefits. (ii) A differentiable on-chip resource usage estimator that acts as a regularizer during training. This joint treatment encourages accuracy where it matters while penalizing resource-heavy configurations, producing FPGA-efficient quantizers that help meet (sub-)microsecond latency under tight hardware budgets without sacrificing competitive accuracy.

Unlike LUT-based logic mappings, HGQ preserves arithmetic structure, which enables the use of established neural network architectures and training methods. The synthesis backend of HGQ can flexibly mix LUTs and DSP primitives for the implementation and is designed to scale to larger models than current LUT-based methods, making it a more versatile solution for microsecond DNN inference on FPGAs. For very small networks that HGQ can produce a solution with a few thousand LUTs or less (such as some in Tab. 2), existing LUT-based approaches may achieve better absolute resource efficiency and latency in comparison to HGQ, while both approaches can meet realistic latency constraints in this regime. However, for larger models with microsecond-level latency constraints that the smallest designs from HGQ require tens of thousands of LUTs or more (as shown in Tab. 3 and Tab. 4), the current LUT-based methods are not directly applicable due to scalability challenges, and HGQ provides the best resource-accuracy trade-offs among existing methods for such models. Nevertheless, we note that while HGQ is intended for models of larger scales than the LUT-based ones, the target models are still compact compared to typical machine learning models, as the microsecond-order latency requirement still forces the model to fit reasonably on a single FPGA when fully-or-partially unrolled.

We developed a reusable HGQ framework and released it as a free and open-source software under the LGPL-3 license. The Vivado/Vitis<sup>®</sup> FPGA back-ends are supported through integration with da4m1 [80] and h1s4m1 [75], free and open-source libraries that transform machine learning algorithms into hardware designs. In particular, da4m1 targets designs with initiation interval (II) of 1, optimizes the design with distributed arithmetic (DA), and can emit Verilog and VHDL code for various back-ends, while h1s4m1 is more general and emits HLS code for multiple back-ends, including Vivado/Vitis<sup>®</sup> HLS and Intel<sup>®</sup> HLS. The end-to-end workflow is illustrated in Fig. 2.

To the best of our knowledge, this paper is the first to propose differentiable, quantization-aware codesign that learns per-parameter bit-widths and jointly optimizes accuracy and FPGA resources for sub-microsecond DNNs. The main contributions of this work are:

- HGQ, a QAT approach that achieves differentiable fixed-point quantization with learnable bit-widths optimized at arbitrary granularity for hardware-efficient DNNs on FPGAs. It automatically includes pruning by assigning zero bit-width to pruned parameters.
- Releasing HGQ as a free and open-source library, with an end-to-end software-hardware codesign workflow closely integrated with the public-domain da4m1 and h1s4m1 tools.
- A comprehensive evaluation of the proposed framework. HGQ provides significant resource and latency improvement

while maintaining model accuracy compared to other quantization methods when deploying on FPGAs.

## 2 Background and Related Work

Efforts in recent years have focused on algorithmic efficiency, using compact architecture, pruning and especially quantization [49, 55]. Quantization is widely adopted for compressing DNNs for specialized hardware such as FPGAs or ASICs with low precision quantization, such as binary or ternary, enhancing throughput and latency. Key examples include DoReFa Net [105], ABC-net [50], Binaryconnect [25], XNOR-net [73], TWN [48], TTQ [106], and [88]. With the same principle, several studies [18, 38, 65, 101, 104] utilize multi-bit network designs that represent weights through binary bases and floating-point values.

Heterogeneous quantization assigns layer/channel-specific precision to reduce accuracy loss. In particular, [95] and [54] use reinforcement learning to find optimal bit-widths. [19, 31, 32, 36, 103] focus on optimizing bit-widths with second-order approximations of the loss function. DNAS [99] and AutoQKeras [22] optimize bit-widths and network architecture simultaneously with stochastic sampling, or gradient-free search. Similarly, MetaML [66, 68] applies iterative optimizations to various hyperparameters. FILM-QNN [81] targets hardware-friendly bit-widths for CNNs. Heterogeneous quantization at sub-layer/channel granularity has also been studied. RVQuant [62], BitsandBytes [28], SpQR [29], and SqueezeLLM [44] keep a small set of outlier weights at higher precision. These approaches primarily target weight size reduction of larger models, rather than efficient inference on specialized hardware.

On the subfield of real-time inference of neural networks on FPGAs, the state-of-the-art models in the LHC community are produced with the QKeras-hls4ml workflow [22, 61, 75, 97]. Although many optimizations, such as neural architecture search, pruning strategies or hybrid training schedules are proposed, the core QAT still adopts a conventional fixed-precision, uniform quantization scheme. This is primarily due to constraints from the FPGA platforms together with latency requirements, which effectively forbids the use of non-uniform quantization and grouped scaling factors.

Closely related to this work, QKeras [22], built on Keras with hls4ml [75] for deployment, trains networks with hardware-friendly fixed-point weights and activations. Its AutoQKeras feature tunes layer-wise quantization via gradient-free search. QKeras is the *de facto* framework for training neural networks for sub-microsecond DNN inference on FPGAs in the LHC community, and it is widely used in the CERN LHC experiments [1, 2, 22, 61, 75, 78, 97]. Brevitas [5], the PyTorch [63] counterpart, is commonly used with AMD’s FINN or FINN-R [14, 92] flow for AMD® FPGAs.

LUT-based DNN inference [6–8, 16, 94, 98] instead maps computations directly to FPGA logic look-up tables.

Some approaches begin from standard NNs (e.g., NeuralUT [6]), while more advanced ones (e.g., DWN [10], NeuralUT-Assemble [7]) train or finetune directly in the LUT domain. These methods deliver extremely low latency and high throughput on compact models, at the cost of moderate accuracy degradation versus floating point. Moreover, to the best of our knowledge, these methods have only been demonstrated on small models of a few thousand LUTs, and it

is unclear if they can be applied to larger models of order of  $\sim 100k$  LUTs, which are shown to be necessary for many real-world applications [61, 67, 97]. When scaling to larger models, the sparse gradient in the look-up table space may lead to further challenges in training stability and convergence. Our method is orthogonal to these LUT-based approaches, offering an alternative path to sub-microsecond FPGA inference. Since several LUT-based methods also rely on quantization-aware training of the logic lookup tables, combining them with HGQ to further improve resource efficiency is a promising direction for future work.

On the FPGA deployment side, several open-source libraries exist for deploying machine learning models on FPGAs. Among them, hls4ml is the most widely used open-source library for deploying machine learning models on FPGAs with sub-microsecond latency [75]. It supports various neural network architectures, including fully-connected networks, convolutional networks, and some graph-based networks. The framework has been adopted in production for the L1 trigger systems at the CMS experiment [1, 2]. da4ml [80] is another open-source library for mapping a quantized neural network to FPGA designs, focusing on optimizing the constant-matrix-vector multiplications (CMVMs) in the network with DA [57]. Instead of leaving the multiplication operations to be inferred by the HLS/synthesis tool, it explicitly transforms the CMVMs into optimized adder graphs with common subexpression elimination. The framework can emit standard Verilog or VHDL code, or may be used as a plugin to hls4ml for CMVM operations in various layers. It is also used in production at the CMS L1 trigger system [2] in conjunction with hls4ml. QONNX [91] is a quantization extension to the ONNX [11] format, which provides a unified format for representing quantized neural networks, and acts as the common interface for FINN [14, 92] and hls4ml. HGQ currently has partial support to export its optimized models to QONNX. HGQ integrates seamlessly with both hls4ml and da4ml, acting as a hardware-friendly frontend that, given an accuracy target and resource budget, automatically explores mixed-precision configurations.

## 3 High Granularity Quantization (HGQ)

This section introduces the HGQ algorithm, which consists of a differentiable, fixed-point quantization scheme promoting loss-awareness for lower bit-widths, and a differentiable on-chip resource estimator promoting lower resource consumption.

### 3.1 Differentiable quantization

**3.1.1 Fixed-point quantization scheme.** We adopt hardware-efficient fixed point representations. In particular, we use a scheme compatible with the `ac/ap_fixed` number schemes in `hlslib/vitis`. We define the map  $f: \mathbb{R} \rightarrow \mathbb{Q} \subset \mathbb{R}$  as the quantizer, which takes five parameters –  $s$ : (boolean) signed,  $i$ : (integer) integer bits, excluding sign,  $f$ : (integer) fractional bits,  $r\_mode$ : (choice) rounding mode, and  $o\_mode$ : (choice) overflow mode. The representable values of a quantized number  $x^q$  are  $\mathbb{Q} = [-s \cdot 2^i, 2^i - 2^{-f}]$  with a step size of  $2^{-f}$ . The total width of the number is  $w = s + i + f$ . The quantization operation is given by eq. (1), where  $[*]$  is the rounding operation depending on  $r\_mode$ , while the overflow behavior depends on

*o\_mode*. Depending on *r\_mode* and *o\_mode*, the quantization operation may incur different hardware overheads when performed on FPGAs. Since the weights are known after training, the overhead is only a concern for the activations.

$$q(x) = \begin{cases} \lfloor x \cdot 2^f \rfloor \cdot 2^{-f}, & \text{if } x \in [-s \cdot 2^i, 2^i - 2^{-f}] \\ \text{overflow} & \text{otherwise} \end{cases} \quad (1)$$

Throughout the network, we primarily use *r\_mode* =RND, which recovers the round-to-nearest, ties up scheme ( $\text{round}(x) = \text{floor}(x+0.5)$ ) due to high stability and low overhead. While TRN has the absolute minimal overhead which truncates beyond the LSBs, it introduces an undesirable systematic bias into the quantization error. While the additional addition required by RND is an overhead, it can frequently be fused into the bias addition operation in neural networks, thus the overhead introduced is negligible.

Both *o\_mode* =SAT (clipping) and *o\_mode* =WRAP (modulo) are supported in our framework. The WRAP mode has minimal hardware overhead by truncating beyond the MSB, but this effectively performs a modulo operation, which is chaotic during training due to the discontinuity and the periodicity. To mitigate this issue, we track only  $f$  during training for values with *o\_mode* =WRAP, and determine  $i$  before deployment. For weights, this is trivial, and for activations, this is done by profiling the activation ranges on a representative dataset and statistically avoiding the overflows. In practice, we find using the training dataset for this purpose would usually suffice. In contrast, SAT mode is more stable during training and produces lower bit-widths in general for activations; however, it is also associated with a high overhead, as it requires additional comparators and multiplexers to perform the clipping. Including the overflow behavior, during training, the quantization operation is given by Eq. (2).

$$q_{\text{train}}(x) = \begin{cases} \lfloor x \cdot 2^f \rfloor \cdot 2^{-f} & \text{if WRAP} \\ \text{clip}(\lfloor x \cdot 2^f \rfloor \cdot 2^{-f}, -s \cdot 2^i, 2^i - 2^{-f}) & \text{if SAT} \end{cases} \quad (2)$$

The rounding operation  $\lfloor * \rfloor$  is non-differentiable. During training, we use the Straight Through Estimator (STE) [13] to approximate the gradients of the quantization operation to tune the parameters of the network. The STE is a technique commonly used in training quantized neural networks [22, 23], as it provides a simple and effective way to estimate gradients. By applying STE to the rounding operation, we approximate its gradient as identity, i.e.,  $\partial q(x)/\partial x = 1$ . This allows gradients to flow through the quantization operation during backpropagation, allowing optimization of the network parameters.

**3.1.2 Connection to Pruning.** Eq. (2) shows that the quantized value is zero if  $|x| < 2^{-f-1}$ . As  $f \in \mathbb{Z}$ , a sufficiently small  $f$  will cause the corresponding parameters in the network to vanish, effectively pruning these parameters. Thus, HGQ automates network pruning during training by assigning a distinct bit-width to each parameter group in the network, taking into account both model performance and resource consumption. When the granularity for quantization is on a per-parameter basis, a fully unstructured pruning is automatically performed.

## 3.2 Surrogate gradients

To optimize bit-widths with per-parameter granularity, we need to make the bit-widths differentiable with gradients attached to them. While bit-widths are inherently discrete, we relax them to be continuous values and round them to integers before the quantization operations.

The gradients from the lower and upper bounds for the SAT overflow mode are straightforward, while the gradient from the  $2^{-f} \cdot \lfloor 2^f \cdot x \rfloor$  term in both SAT and WRAP modes requires careful treatment.

To obtain the surrogate gradient from the  $2^{-f} \cdot \lfloor 2^f \cdot x \rfloor$  term, we first consider a parameter  $x$  (e.g., weight or activation) in the network and its corresponding fractional bit-width  $f$ . The associated quantization error  $\delta_f$  related to this term can be expressed as  $\delta_f \equiv x - f^q(x) = x - \lfloor x \cdot 2^f \rfloor \cdot 2^{-f}$ . During training, let  $x$  be a random variable following a smooth distribution  $\mathbb{D}_x$ , such that the variance of  $\mathbb{D}_x$  is significantly larger than the quantization error  $\delta_f$  so that the quantization error follows a uniform distribution:

$$\delta_f \sim \text{Uniform}(-2^{-f-1}, 2^{-f-1}). \quad (3)$$

Let the loss of the network be  $\mathcal{L}$ . The gradient of  $f$  with respect to  $\mathcal{L}$  is given by

$$\frac{\partial \mathcal{L}}{\partial f} = \frac{\partial \mathcal{L}}{\partial \delta_f} \cdot \frac{\partial \delta_f}{\partial f}. \quad (4)$$

In this expression, the first term  $\frac{\partial \mathcal{L}}{\partial \delta_f}$  can be obtained from back-propagation. The second term  $\frac{\partial \delta_f}{\partial f}$  is not well-defined, since  $f$  can only take integer values for a properly defined quantizer and thus has no gradient. To address this issue, we propose a surrogate gradient method that assigns a gradient to  $f$  only on integer values. We can now express the loss as a function of the weights  $\theta$  and all the quantization errors  $\delta$ ,  $\mathcal{L}(\theta, \delta)$ . We assume that the loss function is sensitive to the magnitude of the quantization errors, but not their signs, i.e.  $\mathcal{L}(\theta, |\delta|)$  with  $|\delta|$  being the element-wise absolute value of  $\delta$ .

For a parameter  $x \sim \mathcal{D}_x$  to be quantized with  $f \in \mathbb{Z}$  fractional bits, the corresponding absolute quantization error is  $|\delta_f| \equiv |x - f^q(x)| \sim \text{Uniform}(0, 2^{-f-1})$ . By increasing  $f$  by one, we obtain the absolute quantization error  $|\delta_{f+1}|$  as a function of  $f$  and  $|\delta_f|$ :

$$|\delta_{f+1}| = \begin{cases} |\delta_f| & |\delta_f| \leq 2^{-f-2} \\ 2^{-f-1} - |\delta_f| & |\delta_f| > 2^{-f-2} \end{cases}. \quad (5)$$

A straightforward way to obtain the gradient of  $|\delta_f|$  with respect to  $f$  is to use the finite difference approximation

$$\frac{\partial |\delta_f|}{\partial f} \leftarrow |\delta_{f+1}| - |\delta_f|. \quad (6)$$

However, as the absolute quantization error is bounded by a geometric sequence of  $2^{-f-1}$ , using a linear difference for approximation may be suboptimal. Instead, we use the following heuristic expression to approximate the gradient, which recovers Eq. (6) in the limit of  $|\delta_{f+1}| - |\delta_f| \rightarrow 0$ :

$$\frac{\partial |\delta_f|}{\partial f} \leftarrow \log \frac{|\delta_{f+1}|}{|\delta_f|} \cdot |\delta_f|. \quad (7)$$

Expressing the ratio of  $|\delta_{f+1}|$  and  $|\delta_f|$  as a function of  $|\delta_f|$ , we have

$$\frac{|\delta_{f+1}|}{|\delta_f|} = \begin{cases} 1 & |\delta_f| \leq 2^{-f-2} \\ \frac{2^{-f-1}}{|\delta_f|} - 1 & |\delta_f| > 2^{-f-2} \end{cases} \quad (8)$$

While one may get a surrogate gradient by combining Eq. (7) and Eq. (8), using the local relations as expressed in Eq. (8) between  $|\delta_{f+1}|$  and  $|\delta_f|$  would lead to a loss (gradient) landscape for  $f$  with extensive high-frequency components; such landscape is difficult to optimize. To mitigate this issue, we smooth out the loss (gradient) landscape by taking the expectation of the first term of Eq. (7) over  $|\delta_f| \sim \text{Uniform}(0, 2^{-f-1})$ :

$$\mathbb{E}_{|\delta_f|} \left[ \log \frac{|\delta_{f+1}|}{|\delta_f|} \right] = -\log 2. \quad (9)$$

By substituting Eq. (9) into Eq. (7), and adding a  $\text{sign}(\delta_f)$  term on both sides, we have

$$\frac{\partial \delta_f}{\partial f} \leftarrow -\log 2 \cdot \delta_f. \quad (10)$$

The resulting gradient is smooth and easy to optimize. Intuitively, it is the increment of the loss due to quantization with  $f$  fractional bits, scaled by a factor of  $\log 2$ .

### 3.3 FPGA resource consumption estimation

On-chip resource consumption of neural networks is often dominated by matrix-vector multiplication operations. A common metric for estimating resource usage of such operations on FPGAs is Bit Operations (BOPs) [12]. However, we find that BOPs often significantly overestimates the resource consumption for unrolled logic. We propose a new metric, Effective Bit Operations (EBOPs), to accurately estimate the resource consumption of CMVM operations in models on FPGAs. The total EBOPs of a model is given by

$$\text{EBOPs} = \sum_{i,j \in \mathcal{M}} b_i \cdot b_j + \sum_{k,l \in \mathcal{A}} \max(b_k, b_l), \quad (11)$$

where  $\mathcal{M} = \{\{i, j\}_n\}$  is the set of all multiplication operations between operands with bit-widths  $b_i$  and  $b_j$  in that model, and  $\mathcal{A} = \{\{k, l\}_m\}$  is the set of all explicit addition/subtraction/muxing operations (e.g., bias, average or maximum pooling) between operands with bit-widths  $b_k$  and  $b_l$ . In practice, EBOPs is typically dominated by the first term, which is constructed based on a simplified model of the resource consumption of a constant multiplier on FPGAs: We assume that the multiplication of a constant (e.g., a weight) and a variable (e.g., an activation) is implemented as a series of shift-and-add operations. Let the bit-width of the variable and the constant be  $b_v$  and  $b_c$  (excluding sign), respectively. The multiplication can be implemented with  $b_c - 1$  shift-and-add operations in the worst case, such that resource consumption is linear in  $b_v \cdot b_c$  when the result is further accumulated with other multiplications in a CMVM operation. This relation can still be used if canonical signed digits [9] representation is adopted for the constants when using the da4m1 backend, where EBOPs is proportional to the number of signed digits weighted by the bit-width of the variable operand. In this case, EBOPs can be regarded as a surrogate for the initial complexity of the CMVM operations before common subexpression elimination is applied, and we find that it still maintains a strong correlation with resource consumption after place-and-route.

Experimental findings confirm that EBOPs is a reliable estimator for on-chip resource consumption when using naive unrolled implementations in h1s4m1, which closely mirrors a linear combination of usage of look-up tables (LUTs) and usage of digital signal processors (DSPs). In da4m1, the direct linear relation breaks down due to further optimizations performed, but EBOPs maintains a strong positive correlation, so it is still a useful differentiable surrogate.

EBOPs is incorporated in the loss function as a regularization term with a coefficient  $\beta \in \mathbb{R}_+$  to obtain a trade-off between performance and on-chip resource usage. Since some network parameters are not involved in multiplication operations, e.g., the last-layer's outputs or inputs, we add another  $L_1$  regularization with a coefficient  $\gamma \in \mathbb{R}_+$  to the bit-widths to keep them from growing indefinitely. Hence, the final loss function is given by

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \beta \cdot \text{EBOPs} + \gamma \cdot \sum \text{bit-widths}, \quad (12)$$

As all additional gradients introduced in this section only apply to bit-widths, the loss landscape of the network weights remains unperturbed compared to that of networks with static quantization parameters.

## 4 Implementation

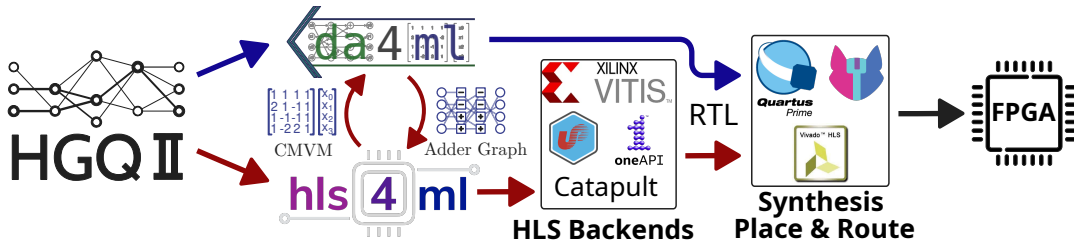
The HGQ method is implemented as an easy-to-use, standalone Python package, HGQ, on top of Keras v3 [20]. The package is open-source under the LGPL-3.0 license.

While the PyTorch style, dynamic graph-based deep learning frameworks are popular in the research community, we choose Keras as the base framework since dynamic graph features cannot be easily supported when exporting the models to RTL/HLS for FPGA deployment. In contrast, Keras uses a static graph-based framework<sup>2</sup>, which provides a straightforward path for training and exporting the models to synthesizable RTL/HLS designs. Furthermore, since Keras v3 supports multiple backends, HGQ can be used with all Keras-supported backends, including JAX [15], TensorFlow [4], and PyTorch [63]. When using the PyTorch backend, one could still directly use Keras layers as torch modules, so Keras is a suitable choice for implementing HGQ. Depending on the specific training condition, HGQ's training speed is typically 50% to 90% of that of the corresponding native Keras models when using the JAX or TensorFlow backends, and 30% to 80% when using the PyTorch backend. In practice, this means that any model that can be trained with the underlying framework can also be trained with HGQ at comparable cost, so there is no intrinsic limitation on model size from the training side.

Specifically, XLA compilation provided by JAX and TensorFlow is supported and can provide significant speedup during training, especially for small models aiming for FPGA deployment. In addition to standard GPU training, HGQ also supports training on Google TPU [40] when using the JAX or TensorFlow backend.

Although our evaluation focuses on dense, convolutional, and graph neural networks, HGQ supports more complex operations, including multihead attention layer [93] and Linformer attention layer [96], which can be used to construct and train Transformer or Linformer models with heterogeneous quantization.

<sup>2</sup>The underlying graph execution engines support dynamic graphs, while the Keras front-end itself makes a static graph when using `keras.Model` by default.



**Figure 2: Overall workflow of the HGQ framework. The dark blue flow uses the da4ml backend, and the brown flow uses the hls4ml backend with optional DA optimizations.**

We add support for both da4ml and hls4ml backends to import models trained with HGQ, and convert them into synthesizable RTL/HLS bit-exact designs. Partial support is also added to the QONNX project [91] to import HGQ models.

The overall workflow of using HGQ is shown in Fig. 2. A user first constructs a model with HGQ-provided quantized layers in the same way as the original Keras model, and then trains the model like any other Keras models in their preferred manner. After training, the user may convert the model into a synthesizable RTL/HLS design with either the da4ml or hls4ml backend, and follow the respective workflow to obtain an RTL design to integrate into their FPGA implementation. While heterogeneous activation quantization is not supported, users may choose to export the models to QONNX format for other backends, such as FINN.

## 5 Evaluation and Analysis

### 5.1 Experimental Setup

**5.1.1 Task description and model architectures.** To evaluate the performance of HGQ, we train and evaluate models on a variety of tasks, including the jet substructure classification (JSC) task at the LHC [64], the SVHN image classification task [58], and the muon tracking task [78]. For the JSC task, two variants of inputs for the same task are available: one with 16 high-level features (HLF), and the other with up to 150 particle-level features (PLF). For the HLF inputs, two versions of the dataset are available: one hosted on OpenML [24], and the other hosted on CERNBox [17].

All JSC tasks are classification tasks with 5 classes, and the performance is evaluated using the test accuracy. For both HLF JSC tasks, we use a fully-connected network with 3 hidden layers taken from [22], and the size for the datalanes is 16–64–32–32–5. For the PLF JSC task, the input is an  $N \times n$  array, where  $N$  is the maximum number of particles, and  $n$  is the number of features per particle. The array is padded with zeros when the number of particles is less than  $N$ . When  $n = 3$ , we apply an additional selection of  $p_T > 2^3$  GeV in align with [61]. We use a linearized graph neural network, JEDI-linear [70], for all PLF JSC tasks. The SVHN task is a 10-class classification task with an input image of size  $32 \times 32 \times 3$ . We use a LeNet-like [47], small CNN with 3 convolutional layers and 2 fully-connected layers taken from [3]. The muon tracking task is a regression task to predict the angle of incidence of a muon passing through a simplified model of the ATLAS Thin Gap Chambers (TGC) detector [78] [83]. The input is a set of three 2-d arrays of size  $50 \times 3$ ,  $50 \times 2$ , and  $50 \times 2$ , representing the three stations of the TGC

<sup>3</sup>The  $p_T$  is the transverse momentum of the particle with respect to the beam axis.

Task	Batch size	HGQ	Keras	NLA
HLF JSC	16600	0.645	0.414	164.
JSC PLF (32 particles, 16 features)	2790	1.85	1.25	-
JSC PLF (64 particles, 16 features)	2790	4.10	2.76	-
TGC Muon Tracking	51200	2.30	1.68	-
SVHN Classifier	2048	5.03	3.49	-

**Table 1: Training time of HGQ, native Keras, and NeuralUT-Assemble (NLA) on various tasks. Results are obtained on a single NVIDIA RTX 4090 GPU with Core i7 13700 CPU. The unit is in milliseconds (ms) per step at specified batch size.**

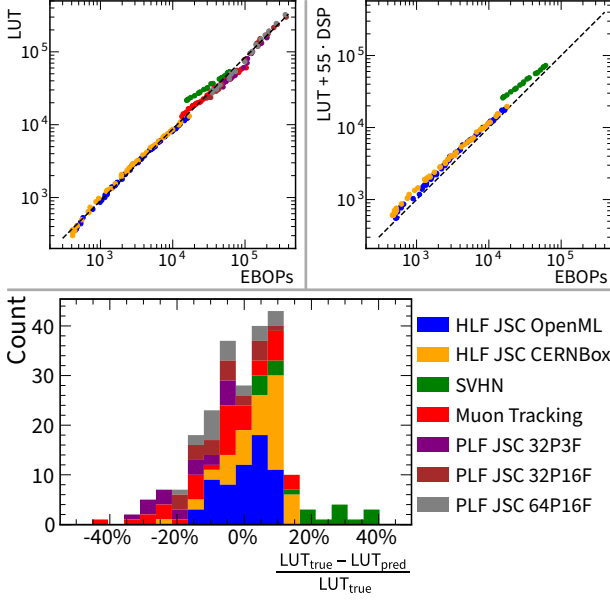
detector, respectively. All arrays are binary-valued, with 1 representing a hit in that pixel. The model used is taken from [78], which is a multi-stage model designed to align with the dynamics of the detector. The exact architectures for the models used for these tasks can be found in their respective references. The performance is evaluated with the test resolution, defined as the root-mean-square of the residual between the predicted and true angle of incidence, excluding up to 0.1% of outliers greater than 30 mrad away from the true value.

All resource/latency values in this section from the HGQ-trained models are obtained after place-and-route out-of-context with Vivado 2025.1 targeting Xilinx Virtex UltraScale+ VU13P FPGA<sup>4</sup> (part: xcvu13p-f1ga2577-2-e) with global retiming enabled. The model accuracy is measured on the generated HLS or RTL models over test datasets. For all RTL models generated by da4ml, we verify the functional correctness with Verilator [77] against the Keras models. For all models obtained, we did not notice any bit-mismatches between the Verilated models and the Keras models that cannot be explained by floating-point rounding errors, and there is no noticeable<sup>5</sup> difference in the accuracy/resolution between the Keras models and the Verilated models.

**5.1.2 Training configuration.** To explore the trade-off between accuracy (or resolution) and resource usage with different quantization levels, we adjust the value of the  $\beta$  factor for each task during one training session to map out the Pareto fronts. For each training session, we initialize the model with a small  $\beta$  value and then gradually increase it throughout the training with an exponential schedule. The initial and final  $\beta$  values vary depending on the task: 5e-7 to 1e-3 for HLF JSC, 2e-8 to 3e-6 for both PLF JSC and TGC

<sup>4</sup>One of the two (the other one is VU9P) proposed FPGA models for the majority of both ATLAS and CMS Experiments' future trigger system, which is also a common benchmark target used in prior works.

<sup>5</sup>> 99.5% of the models we tested had 0 bit mismatch on their corresponding test datasets. For those models with at least one mismatch, no larger than  $10^{-3}\%$  accuracy degradation is observed. Only models requiring at least 23 bits during accumulation may have mismatches when using the JAX backend running on CPU.



**Figure 3: The relationship between EBOPs and the post place-and-route resource consumption with (top left) and without (top right) DA optimization, and the distribution of error between the estimated and actual LUT consumption after place-and-route with DA (lower). For the error distribution,  $LUT_{pred}$  is given by  $\exp(0.985 \cdot \log(EBOPs))$ .**

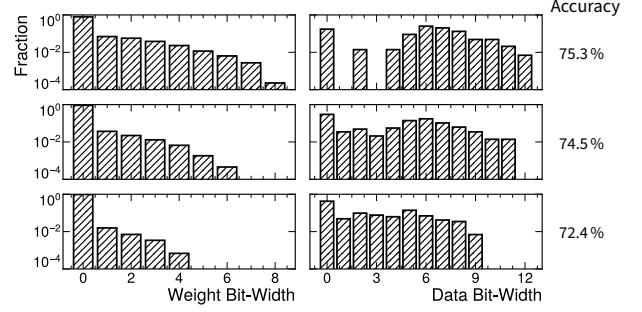
Muon Tracking, and  $1e-7$  to  $1e-5$  for SVHN Classification. Meanwhile, we maintain the  $\gamma$  value fixed at an arbitrarily small value of  $2e-8$  for all experiments to avert the risk of diverging bit-widths for some parameters not involved in multiplication operations.

We use the Adam optimizer [45] with a cosine annealing with restart learning rate schedule for all experiments. After each epoch, we record the validation accuracy and EBOPs, and save the checkpoints for models that are on the Pareto front defined by these two metrics for further evaluation. Post-training, we use the entire training and validation datasets for calibration to determine the required bit-widths and evaluate the exact EBOPs for all the checkpointed models.

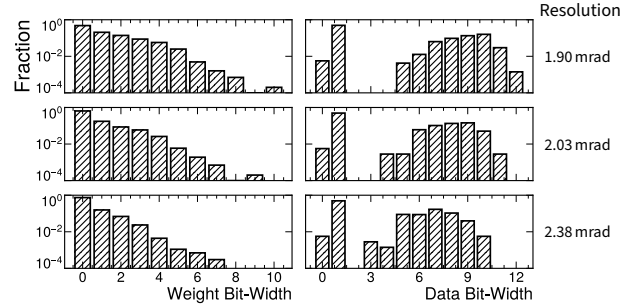
The training time per step for HGQ, native Keras, and NeuralUT-Assemble [7] are compared in Tab. 1. For the models benchmarked, HGQ achieves a training speed that is  $\sim 70\%$  of the native Keras models, whereas NeuralUT-Assemble is significantly slower ( $\sim 254$  times slower for the HLF JSC task).

### 5.2 Resource Estimation via EBOPs

We first demonstrate that EBOPs is a reasonable differentiable surrogate for on-chip resource consumption with both da4m1 and h1s4m1. Empirically, we find that EBOPs maintains a strong correlation with the actual on-chip resource consumption after place-and-route when using da4m1 as the backend for each individual architecture, as shown on the top left of Fig. 3. Within each architecture, as EBOPs increases, the LUT consumption also increases monotonically with a diminishing rate. This diminishing rate is expected as common subexpression elimination used in da4m1 is effective when the CMVM problems are more complex and with



**Figure 4: Distributions of the weights and data activations bit-widths of the HGQ models for HLF JSC task (CERNBox).**



**Figure 5: Distributions of the weights and data activations bit-widths of the HGQ models for the muon tracking task.**

higher bit-widths. While the consumption for each individual architecture is not linear to EBOPs, the overall trend combining different architectures is approximately linear in logarithmic space with a very small intercept, and an approximate surrogate relation of  $LUT \approx \exp(0.985 \cdot \log(EBOPs))$  can be used to estimate LUT consumption from EBOPs. The error distribution between estimated and actual LUT consumption is shown on the lower plot of Fig. 3, where the relative error is within 20% for the majority of the models, and the actual LUT consumption never exceeds more than 20% of the estimated value among all models we tested, excluding the SVHN Classifier model which requires significant additional switching and buffering logic not modeled by EBOPs. Hence, EBOPs could be a useful rough estimator for model selection, for instance, during model architecture optimizations. While EBOPs does not provide the most accurate resource estimation by itself, as shown on the top left of Fig. 3, the relative ordering of resource consumption is mostly preserved – for each individual architecture, a model with higher EBOPs always consumes more LUTs than a model with lower EBOPs when implemented in the same way. This suggests that EBOPs is a reasonable differentiable surrogate for LUT consumption when using da4m1 as the backend.

When using h1s4m1 as the backend, we find EBOPs to be approximately a linear combination of LUT and DSP consumption:  $EBOPs \approx LUT + 55 \cdot DSP$ , as shown on the top right of Fig. 3. This relation is also empirically observed for the Xilinx UltraScale+ chips, and it may be different for other series of FPGAs or other vendors.

However, it is important to note that EBOPs only estimates resource consumption from the CMVM operations and other arithmetic operations in the network. Other non-arithmetic operations that are not directly associated with the network architecture, such

HLF JSC (OpenML)							
Implementation	Accuracy $\uparrow$	Latency [cycles]	LUT	DSP	FF	$F_{\max}$ [MHz]	II [cycles]
<b>HGQ</b>	76.9%	20 (36.3 ns)	10,182	0	10,480	551.6	1
<b>HGQ</b>	76.6%	16 (26.2 ns)	5,991	0	5,890	611.2	1
<b>HGQ</b>	75.6%	12 (18.6 ns)	2,298	0	2,217	645.2	1
<b>HGQ</b>	74.9%	10 (14.5 ns)	1,390	0	1,316	691.6	1
<b>HGQ</b>	73.2%	6 (9.1 ns)	366	0	363	662.7	1
QKeras [ICFPT'23] [72]**	76.3%	15 (105.0 ns)	5,504	175	3,036	> 142.9	2
DWN [ICLR'24] [10]***	76.3%	10 (14.4 ns)	6,302	0	4,128	695.	1
QKeras [CoRR'21] [75]*	76.2%	9 (45)	63,251	38,	4,394	$\sim$ 200	1
MetaML-Pro [TRET'S'26] [66]*	76.1%	10 (50 ns)	13,042	70	N/A	$\sim$ 200	1
NeuraLUT-Assemble [FCCM'25] [7]	76.0%	2 (2.1 ns)	1,780	0	540	940.	1
TreeLUT [FPGA'25] [42]	75.6%	2 (2.7 ns)	2,234	0	347	735.	1
SymbolNet [MLST'25] [89]	71.1%	2 (10 ns)	177	3	109	$\sim$ 200	1
HLF JSC (CERNBox)							
Implementation	Accuracy $\uparrow$	Latency [cycles]	LUT	DSP	FF	$F_{\max}$ [MHz]	II [cycles]
<b>HGQ</b>	75.3%	18 (31.1 ns)	10,921	0	11,183	578.4	1
<b>HGQ</b>	75.1%	15 (24.6 ns)	5,974	0	5,775	609.8	1
<b>HGQ</b>	74.5%	13 (20.4 ns)	3,152	0	2,941	636.9	1
<b>HGQ</b>	73.2%	11 (14.6 ns)	976	0	904	754.1	1
<b>HGQ</b>	72.4%	9 (9.9 ns)	623	0	642	905.8	1
<b>HGQ<math>^\dagger</math></b>	72.4%	2 (5.4 ns)	641	0	227	367.4	1
PolyLUT [TC'25] [8]	75.1%	5 (24.6 ns)	246,071	0	12,384	203.	1
NeuraLUT-Assemble [FCCM'25] [7]	75.0%	2 (5.7 ns)	8,539	0	1,332	352.	1
NeuraLUT-Assemble [FCCM'25] [7]	75.0%	7 (7.0 ns)	8,535	0	2,717	994.	1
PolyLUT-Add [FPL'24] [53]	75.1%	5 (15.9 ns)	36,484	0	1,209	315.	1
NeuraLUT [FPL'24] [6]	75.1%	5 (13.6 ns)	92,357	0	4,885	368.	1
ReducedLUT [FPGA'25] [16]	74.9%	N/A	58,409	0	N/A	302.8	N/A
ReducedLUT [FPGA'25] [16]	72.5%	N/A	2,786	0	N/A	408.5	N/A
QKeras [NMI'21] [22]*	74.8%	11 (55 ns)	39,782	124	8,128	$\sim$ 200	1
QKeras [NMI'21] [22]*	72.3%	11 (55 ns)	9,149	66	1,781	$\sim$ 200	1
AmigoLUT-NeuraLUT [FPGA'25] [98]	74.4%	5 (9.6 ns)	42,742	0	4,717	520.	1
AmigoLUT-NeuraLUT [FPGA'25] [98]	72.9%	5 (5.0 ns)	1,243	0	1,240	1,008.	1
LogicNets [FPL'20] [90]	71.8%	5 (11.7 ns)	37,931	0	810	427.	1

**Table 2: Performance and resource consumption of the HLF JSC models on the Xilinx UltraScale+ FPGAs with speedgrade -2. The models marked with \* only synthesized to netlist but did not perform place and route, and the  $F_{\max}$  and latency shown are based on the HLS target clock period. The models marked with \*\* did not report  $F_{\max}$ , but reported no timing violations at the specified target clock period. The results of DWN\*\*\* are cited from [7] instead of the original paper, as the original work omits preprocessing resource consumption. The design marked with  $\dagger$  is obtained with the same model as the 9-cycle one in the row above it, but uses a less aggressive pipelining strategy to achieve lower latency.**

as those arising from the control logic for processing element scheduling, or FIFOs for buffering added at implementation time when needed, are not currently accounted for in EBOPs. In these cases, the actual resource consumption will be higher than what EBOPs predicts. For example the SVHN classifier, which requires additional FIFO buffers and intensive muxing logic that are not modeled in EBOPs, is shown on the lower plot of Fig. 3. We validate that the resulting models have heterogeneous bit-widths as expected, and we show respectively the distribution of the bit-widths for the weights and activations of a few representative models in Fig. 4 and Fig. 5 for the HLF JSC CERNBox and the muon tracking tasks. The bit-widths are shown to distribute across a wide range, with the majority of the weights being quantized to zero bits (pruned), and the bit-widths becoming smaller in general as the model performance is traded off for lower resource consumption.

### 5.3 Comparison with Prior Work

**5.3.1 HLF JSC tasks.** We compare the performance of the models trained with HGQ against various prior efforts, including quantized neural networks, LUT-based models, decision forests, and symbolic models on the HLF JSC task. The accuracy, latency,  $F_{\max}$ , and on-chip resource utilization of various models are summarized in Tab. 2. To visualize the trade-off between accuracy and resource consumption, we also study accuracy versus LUT consumption in Fig. 6 for the two HLF JSC tasks. In both datasets, the HGQ-trained models outperform all previous methods based on quantized neural networks by a large margin, both in terms of accuracy and resource usage, and achieve similar accuracy-resource trade-offs as the state-of-the-art decision forest and symbolic models.

PLF JSC (3 features)								
Implementation	Particles	Accuracy $\uparrow$	Latency [cycles]	LUT	DSP	FF	F <sub>max</sub> [MHz]	II [cycles]
HGQ (GNN)	32	79.2%	24 (108.8 ns)	159,238	0	91,027	220.5	1
HGQ (GNN)	32	78.5%	20 (72.3 ns)	80,618	0	42,101	276.5	1
QKeras, DS [MLST'24] [61]*	32	$\leq 75.9\%$	26 (130 ns)	903,284	434	358,754	$\sim 200$	2
QKeras, GNN [MLST'24] [61]*	32	$\leq 75.8\%$	32 (160 ns)	1,162,104	2,120	761,061	$\sim 200$	3
PLF JSC (16 features)								
Implementation	Particles	Accuracy $\uparrow$	Latency [cycles]	(A)LUT	DSP	FF	F <sub>max</sub> [MHz]	II [cycles]
HGQ (GNN)	64	82.4%	26 (122.5 ns)	244,515	0	112,993	212.2	1
HGQ (GNN)	64	80.7%	9 (45.0 ns)	53,546	0	13,629	199.9	1
HGQ (GNN)	32	81.5%	26 (119.8 ns)	238,255	0	116,039	217.1	1
HGQ (GNN)	32	80.2%	9 (45.5 ns)	48,343	0	10,012	197.7	1
HGQ (MLPM, Altera Agilex 7)	64	81.1%	10 (51.6 ns)	95,650	0	20,112	193.7	1
GNN U4 (TECS'24) [67]*	50	80.9%	130 (650 ns)	855k	8,945	201k	$\sim 200$	100
GNN U5 (TECS'24) [67]*	50	81.2%	181 (905 ns)	815k	8,986	189k	$\sim 200$	150
GNN J4 (TECS'24) [67]*	30	78.4%	58 (290 ns)	865k	8,776	138k	$\sim 200$	30
GNN J5 (TECS'24) [67]*	30	79.9%	181 (905 ns)	911k	9,833	158k	$\sim 200$	150
GNN (FPL'22) [69]*	50	80.4%	2132 (10660 ns)	1515k	12,284	533k	$\sim 200$	650
GNN (FPL'22) [69]*	30	78.7%	382 (1910 ns)	1158k	11,504	246k	$\sim 200$	400

**Table 3: The performance and resource consumption of the PLF JSC models on the Xilinx UltraScale+ FPGAs with speedgrade -2 and Agilex 7 (part number AGFB014R24A2E2V) for the Quartus synthesized one. The models marked with \* only synthesized to netlist but did not perform place and route, and the F<sub>max</sub> and latency shown are based on the HLS target clock period.**

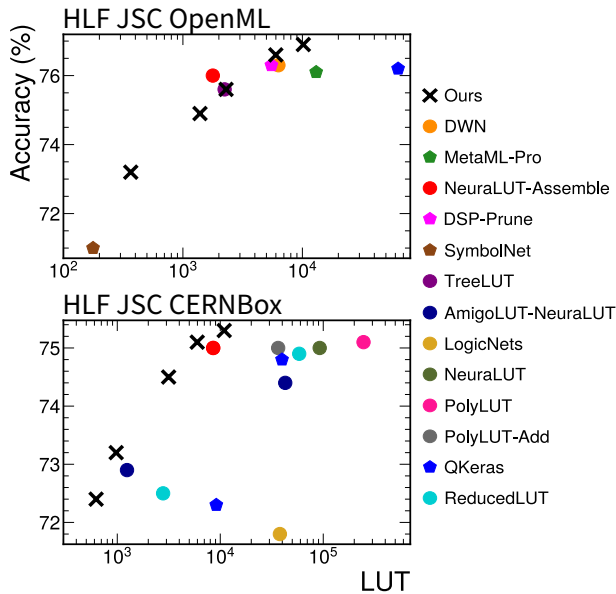
Muon tracking							
Implementation	Resolution $\downarrow$	Latency [cycles]	LUT	DSP	FF	F <sub>max</sub> [MHz]	II [cycles]
HGQ	1.90 mrad	8 (47.4 ns)	41,830	0	10061	168.9	1
HGQ	2.03 mrad	6 (35.2 ns)	25,716	0	3455	170.3	1
HGQ	2.38 mrad	5 (28.7 ns)	14,789	0	3091	174.1	1
QKeras [NIMA'23] [78]**	1.95 mrad	17 (106.3 ns)	37,867	1,762	8,443	$> 160$	1
QKeras [NIMA'23] [78]**	2.04 mrad	13 (81.3 ns)	54,638	324	6,525	$> 160$	1
QKeras [NIMA'23] [78]**	2.45 mrad	10 (62.5 ns)	28,526	24	2,954	$> 160$	1
SVHN classification							
Implementation	Accuracy $\uparrow$	Latency [cycles]	LUT	DSP	FF	F <sub>max</sub> [MHz]	II [cycles]
HGQ (hls4ml)	93.8%	1,048 (5319.6 ns)	66,056	52	24,207	197.0	1,029
HGQ (hls4ml)	92.0%	1,060 (5272.4 ns)	41,733	20	18,774	201.0	1,030
QKeras [MLST'21] [3]*	94.4%	1,035 (5,175 ns)	111,152	174	32,554	$\sim 200$	1,030
QKeras [MLST'21] [3]*	88.8%	1,059 (5,295 ns)	38,795	70	14,802	$\sim 200$	1,029
DSP-Prune [ICFPT'23] [72]**	92.4%	5,447 (43,576 ns)	59,279	1,215	46,584	$> 125$	N/A

**Table 4: The performance and resource consumption of the Muon tracking and SVHN classification models on the Xilinx UltraScale+ FPGAs with speedgrade -2. The models marked with \* only synthesized to netlist but did not perform place and route, and the F<sub>max</sub> and latency shown are based on the HLS target clock period. The models marked with \*\* did not report F<sub>max</sub>, but reported no timing violations at the specified target clock period.**

Compared to NeuralUT-Assemble [7] on the OpenML dataset, HGQ can achieve higher accuracy, though achieving the same accuracy typically consumes more resources. On CERNBox, HGQ outperforms NeuralUT-Assemble in both accuracy and resource use. Latency-wise, HGQ is substantially faster than prior quantized neural networks but remains slower than NeuralUT-Assemble. For simple classification tasks that fit within a few thousand LUTs, both approaches are competitive. But, LUT-based methods have not yet been shown to scale to models with tens to hundreds of thousands of LUTs, which are often required for more complex problems. HGQ preserves standard neural-network semantics and scales to deeper/wider models and to a broader set of architectures

(beyond MLPs), as shown in Tab. 3 and Tab. 4. HGQ can also utilize both LUT and DSP resources (Tab. 4), and it has a significantly shorter training time than competing approaches (Tab. 1).

HGQ has been smoothly integrated into da4ml and hls4ml which makes it vendor-portable, whereas current LUT-based approaches are largely tied to AMD devices. Although LUT-based pipelines can deliver the very lowest latencies at high accuracy, level-1 trigger applications do not necessarily require such extreme margins since the LHC collision occurs every 25 ns, so HGQ can produce competitive end-to-end designs. These properties give HGQ better scalability, architectural generality, heterogeneous-resource efficiency, and cross-vendor portability, making it a more versatile



**Figure 6: Accuracy versus LUT consumption of the HLF JSC models on the OpenML and CERNBox datasets. Designs with circle markers use only LUTs for logical operations, while designs with pentagon markers use both LUTs and DSPs. Our designs are marked as "X", and uses only LUTs.**

choice for larger networks and broader deployments. As noted in related work, HGQ can also be combined with NeuraLUT-Assemble to further improve hardware efficiency; we leave this integration to future work.

**5.3.2 PLF JSC tasks.** We show the performance of HGQ-trained models for the PLF JSC tasks in Tab. 3, where the input contains up to 32 or 64 particles with 3 or 16 features per particle, respectively. On this task, HGQ outperforms the prior quantized neural network based models on both accuracy and resource consumption by a substantial margin. We also show an MLP-Mixer (MLPM) based model [79] design running on an Altera FPGA, showing that HGQ can be applied to FPGAs from other vendors other than AMD.

**5.3.3 Other tasks.** As shown in Tab. 4, the HGQ-trained models outperform the prior art on the muon tracking task with significant resource reduction and speedup while achieving higher accuracy. On the SVHN classification task, since intense resource reuse is required to fit the model into the FPGA, the latency is significantly higher than other models so we use the hls4ml backend for this task. The resulting models achieve similar accuracy while using less resource, compared to prior quantized neural network based models on this task. It also shows that our approach can use both LUT and DSPs efficiently.

## 5.4 Ablation

To isolate the contribution of HGQ from backend optimizations in da4ml (e.g., common subexpression elimination), we conduct an ablation study to show the effectiveness of HGQ when using hls4ml as the backend without DA adoption. We show the same models for the HLF JSC tasks converted by hls4ml and synthesized with Vivado 2025.1 in Tab. 5 with prior works using hls4ml

HLF JSC OpenML						
Implementation	Accuracy	Latency	LUT	DSP	FF	F <sub>max</sub> [MHz]
HGQ (hls4ml)	76.9%	53.0 ns	11,444	55	16,299	774.0
HGQ (hls4ml)	76.6%	42.1 ns	7,101	27	9,858	831.3
HGQ (hls4ml)	75.6%	30.5 ns	2,990	4	4,385	917.4
HGQ (hls4ml)	74.9%	25.5 ns	1,933	0	2,859	939.8
HGQ (hls4ml)	73.2%	19.2 ns	556	0	746	939.8
DSP-Prune [ICFPT'23] [72]**	76.3%	105.0 ns	5,504	175	3,036	> 142.9
QKeras [CoRR'21] [75]*	76.2%	45 ns	63,251	38	4,394	~ 200
MetaML-Pro [TRET'S'26] [66]*	76.1%	50 ns	13,042	70	N/A	~ 200
HLF JSC CERNBox						
Implementation	Accuracy	Latency	LUT	DSP	FF	F <sub>max</sub> [MHz]
HGQ (hls4ml)	75.3%	55.1 ns	12,377	56	17,654	743.5
HGQ (hls4ml)	75.1%	50.0 ns	7,119	28	10,053	759.3
HGQ (hls4ml)	74.5%	34.7 ns	4,141	3	5,716	864.3
HGQ (hls4ml)	73.2%	23.4 ns	1,398	1	1,965	939.8
HGQ (hls4ml)	72.4%	20.2 ns	1,045	0	1,445	939.8
QKeras [NMI'21] [22]*	74.8%	55 ns	39,782	124	8,128	~ 200
QKeras [NMI'21] [22]*	72.3%	55 ns	9,149	66	1,781	~ 200

**Table 5: Performance and resource consumption of the HLF JSC models synthesized with hls4ml compared to prior works using the same deployment backend. The models marked with \* only synthesized to netlist but did not perform place and route, and the F<sub>max</sub> and latency shown are based on the HLS target clock period. The models marked with \*\* did not report F<sub>max</sub>, but reported no timing violations at the specified target clock period.**

as the backend for comparison. Compared with the prior quantized neural network based models, HGQ consistently delivers superior accuracy-efficiency trade-offs also on the hls4ml backend, achieving higher accuracy with lower DSP, LUT consumption and achieves lower latency at a much higher F<sub>max</sub>.

## 6 Conclusion and Outlook

This paper presents HGQ, a novel method for training neural networks with heterogeneous quantization for efficient FPGA deployment. The method is validated on a variety of tasks and shows significant improvement over prior efforts on quantized neural networks, achieving competitive performance with state-of-the-art alternative methods on classification tasks. The method is implemented in HGQ, an easy-to-use, open-source Python package which is compatible with multiple backends and supports exporting to synthesizable bit-exact RTL/HLS designs. Future work includes extending the optimization methodology used in HGQ to LUT-based methods, such as NeuraLUT or NeuraLUT-Assemble. Since these methods require quantization on the sub-network level, HGQ could be used potentially to further enhance the hardware efficiency. In addition, HGQ could also be used to produce different neural architectures for them. Moreover, while more complex models such as transformers are supported in HGQ, more work is required for efficient FPGA deployment of these models, and we are working with the hls4ml team on optimizing them for FPGA implementations.

## Acknowledgement

Partial support from the United States DoE (grant numbers DE-SC0011925, DE-FOA-0002705), NSF (grant numbers PHY240298, PHY2117997), United Kingdom EPSRC (grant numbers UKRI256, EP/V028251/1, EP/N031768/1, EP/S030069/1, and EP/X036006/1), Swiss NSF Grant No. PZ00P2\_201594, Schmidt Futures (Grant G-23-64934), KIAT, Intel, and AMD is acknowledged. We acknowledge the Caltech Danny Koh graduate student scholarship and the ETH/Guenther Dissertori partial support for this project.

## References

- [1] 2023. Level-1 Trigger Calorimeter Image Convolutional Anomaly Detection Algorithm. (2023). <https://cds.cern.ch/record/2879816>
- [2] 2024. 2024 Data Collected with AXOL1TL Anomaly Detection at the CMS Level-1 Trigger. (2024). <https://cds.cern.ch/record/2904695>
- [3] Thea Aarrestad, Vladimir Loncar, Nicolò Ghielmetti, Maurizio Pierini, Sioni Summers, Jennifer Ngadiuba, Christoffer Petersson, Hampus Linander, Yutaro Iiyama, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergio Jindariani, Kevin Pedro, Nhan Tran, Mia Liu, Edward Kreinar, Zhenbin Wu, and Duc Hoang. 2021. Fast convolutional neural networks on FPGAs with hls4ml. *Machine Learning: Science and Technology* 2, 4 (jul 2021), 045015. doi:10.1088/2632-2153/ac0ea1
- [4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. <https://www.tensorflow.org/> Software available from tensorflow.org.
- [5] Alessandro, Giuseppe Franco, Nick Fraser, and Yaman Umuroglu. 2021. Xilinx/brevitas: Release version 0.2.1. doi:10.5281/zenodo.4507794
- [6] Marta Andronic and George A. Constantinides. 2024. NeuralUT: Hiding Neural Network Density in Boolean Synthesizable Functions. In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 140–148. doi: 10.1109/FPL64840.2024.00028.
- [7] Marta Andronic and George A. Constantinides. 2025. NeuralUT-Assemble: Hardware-Aware Assembling of Sub-Neural Networks for Efficient LUT Inference. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 208–216. doi:10.1109/FCCM62733.2025.00077
- [8] Marta Andronic and George A. Constantinides. 2025. PolyLUT: Ultra-Low Latency Polynomial Inference With Hardware-Aware Structured Pruning. In *IEEE Transactions on Computers*. IEEE, 3181–3194. doi: 10.1109/TC.2025.3586311.
- [9] Algirdas Avizienis. 1961. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers* EC-10, 3 (1961), 389–400. doi:10.1109/TEC.1961.5219227
- [10] Alan Tandler Leibel Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy Kurian John, Priscila Machado Vieira Lima, and Felipe M.G. França. 2024. Differentiable Weightless Neural Networks. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 2277–2295. <https://proceedings.mlr.press/v235/bacellar24a.html>
- [11] Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. ONNX: Open Neural Network Exchange. <https://github.com/onnx/onnx>.
- [12] Chaim Baskin, Natan Liss, Eli Schwartz, Evgenii Zheltonozhskii, Raja Giryes, Alex M. Bronstein, and Avi Mendelson. 2019. UNIQ. *ACM Transactions on Computer Systems* 37, 1–4 (nov 2019), 1–15. doi:10.1145/3444943
- [13] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR* abs/1308.3432 (2013). arXiv:1308.3432 <http://arxiv.org/abs/1308.3432>
- [14] Michaela Blott, Thomas Preusser, Nicholas Fraser, Giulio Gambardella, Kenneth O'Brien, and Yaman Umuroglu. 2018. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.* 11, 3 (12 2018). arXiv:1809.04570 doi:10.1145/3242897
- [15] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs. <http://github.com/google/jax>
- [16] Oliver Cassidy, Marta Andronic, Samuel Coward, and George A. Constantinides. 2025. ReducedLUT: Table Decomposition with “Don’t Care” Conditions. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25)*. ACM, 36–42. doi:10.1145/3706628.3708823
- [17] CERN Collaboration. 2025. CERNBox LHC Jets Dataset. <https://cernbox.cern.ch/index.php/s/jvFd5MoWhGs1l5v/download> Accessed: 2025-09-15.
- [18] Sung-En Chang, Yanyu Li, Mengshu Sun, Runbin Shi, Hayden K.-H. So, Xuehai Qian, Yanzi Wang, and Xue Lin. 2021. Mix and Match: A Novel FPGA-Centric Deep Neural Network Quantization Framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 208–220. doi:10.1109/HPCA51647.2021.00027
- [19] Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Bridging the Accuracy Gap for 2-bit Quantized Neural Networks (QNN). *CoRR* abs/1807.06964 (2018). arXiv:1807.06964 <http://arxiv.org/abs/1807.06964>
- [20] François Chollet et al. 2015. Keras. <https://keras.io>.
- [21] Claudio Cimarelli, Jose Andres Millan-Romera, Holger Voos, and Jose Luis Sanchez-Lopez. 2025. Hardware, Algorithms, and Applications of the Neuro-morphic Vision Sensor: a Review. arXiv:2504.08588 [cs.CV] <https://arxiv.org/abs/2504.08588>
- [22] Claudionor N. Coelho, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence* 3, 8 (jun 2021), 675–686. doi:10.1038/s42256-021-00356-5
- [23] Ian Colbert, Alessandro Pappalardo, Jakoba Petri-Koenig, and Yaman Umuroglu. 2024. A2Q+: Improving Accumulator-Aware Weight Quantization. In *Forty-first International Conference on Machine Learning*. <https://openreview.net/forum?id=mbx2pLk5Eq>
- [24] OpenML Contributors and LHC Jets HLF Curators. 2020. hls4ml LHC Jets HLF (OpenML Dataset 42468). <https://www.openml.org/d/42468> Accessed: 2025-09-15.
- [25] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *CoRR* abs/1511.00363 (2015). arXiv:1511.00363 <http://arxiv.org/abs/1511.00363>
- [26] G. De Tommasi, D. Alves, T. Bellizio, R. Felton, A. Neto, F. Sartori, R. Vitelli, L. Zabeo, R. Albanese, G. Ambrosino, and P. Lomas. 2010. Real-time systems in tokamak devices. A case study: The JET tokamak. In *2010 17th IEEE-NPSS Real Time Conference*. 1–7. doi:10.1109/RTC.2010.5750334
- [27] Allison McCarn Deiana, Nhan Tran, Joshua Agar, Michaela Blott, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Scott Hauck, Mia Liu, Mark S. Neubauer, Jennifer Ngadiuba, Seda Ogrenci-Memik, Maurizio Pierini, Thea Aarrestad, Steffen Bähr, Jürgen Becker, Anne-Sophie Berthold, Richard J. Bovenre, Tomás E. Müller Bravo, Markus Diefenthaler, Zhen Dong, Nick Fritzsche, Amir Gholami, Ekaterina Govorkova, Dongning Guo, Kyle J. Hazelwood, Christian Herwig, Babar Khan, Sehoon Kim, Thomas Kljinsma, Yaling Liu, Kin Ho Lo, Tri Nguyen, Gianantonio Pezzullo, Seyedramin Rasoulizhad, Ryan A. Rivera, Kate Scholberg, Justin Selig, Sougata Sen, Dmitri Strukov, William Tang, Savannah Thais, Kai Lukas Unger, Ricardo Vilalta, Belina von Krosigk, Shen Wang, and Thomas K. Warburton. 2022. Applications and Techniques for Fast Machine Learning in Science. *Frontiers in Big Data* 5 (April 2022). doi:10.3389/fdata.2022.787421
- [28] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit Optimizers via Block-wise Quantization. *9th International Conference on Learning Representations, ICLR (2022)*.
- [29] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznetsov, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoeftler, and Dan Alistarh. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. arXiv:2306.03078 [cs.CL]
- [30] Giuseppe Di Guglielmo et al. 2025. End-to-end workflow for machine learning-based qubit readout with QICK and hls4ml. (1 2025). arXiv:2501.14663 [quant-ph]
- [31] Zhen Dong, Zhewei Yao, Yaohui Cai, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks. *CoRR* abs/1911.03852 (2019). arXiv:1911.03852 <http://arxiv.org/abs/1911.03852>
- [32] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWARE Quantization of Neural Networks with Mixed-Precision. *CoRR* abs/1905.03696 (2019). arXiv:1905.03696 <http://arxiv.org/abs/1905.03696>
- [33] Javier Duarte, Song Han, Philip Harris, Sergio Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of instrumentation* 13, 07 (2018), P07027.
- [34] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. 2022. Differentiable Model Compression via Pseudo Quantization Noise. arXiv:2104.09987 [stat.ML] <https://arxiv.org/abs/2104.09987>
- [35] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. Learned Step Size Quantization. arXiv:1902.08153 [cs.LG] <https://arxiv.org/abs/1902.08153>
- [36] Elias Frantar, Sidak Pal Singh, and Dan Alistarh. 2024. Optimal brain compression: a framework for accurate post-training quantization and pruning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA, (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 323, 14 pages.
- [37] Shasha Guo, Ziyang Kang, Lei Wang, Shiming Li, and Weixia Xu. 2020. HashHeat: An O(C) Complexity Hashing-based Filter for Dynamic Vision Sensor. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 452–457. doi:10.1109/ASP-DAC47756.2020.9045268
- [38] Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. 2017. Network Sketching: Exploiting Binary Structure in Deep CNNs. *CoRR* abs/1706.02021 (2017). arXiv:1706.02021 <http://arxiv.org/abs/1706.02021>

- [39] Kai Huang and Wei Gao. 2022. Real-time neural network inference on extremely weak devices: agile offloading with explainable AI. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking* (Sydney, NSW, Australia) (*MobiCom '22*). Association for Computing Machinery, New York, NY, USA, 200–213. doi:10.1145/3495243.3560551
- [40] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) (*ISCA '23*). Association for Computing Machinery, New York, NY, USA, Article 82, 14 pages. doi:10.1145/3579371.3589350
- [41] Yi-Chieh Kao, Hung-An Chen, and Hsi-Pin Ma. 2022. An FPGA-Based High-Frequency Trading System for 10 Gigabit Ethernet with a Latency of 433 ns. In *2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 1–4. doi:10.1109/VLSI-DAT54769.2022.9768065
- [42] Alireza Khataei and Kia Bazargan. 2025. TreeLUT: An Efficient Alternative to Deep Neural Networks for Inference Acceleration Using Gradient Boosted Decision Trees. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25)*. ACM, 14–24. doi:10.1145/3706628.3708877
- [43] Alireza Khodamoradi and Ryan Kastner. 2021.  $O(N)O(N)$ -Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors. *IEEE Transactions on Emerging Topics in Computing* 9, 1 (2021), 15–23. doi:10.1109/TETC.2017.2788865
- [44] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiyu Li, Sheng Shen, Michael Mahoney, and Kurt Keutzer. 2023. SqueezeLLM: Dense-and-Sparse Quantization. *arXiv* (2023).
- [45] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).
- [46] Y. T. Lai, M. Aoyama, H. Bae, S. Bähr, M. C. Chang, H. Hayashii, Y. Iwasaki, J. B. Kim, K. T. Kim, C. Kiesling, T. Koga, P. C. Lu, S. M. Liu, F. Meggendorfer, H. K. Moon, T. J. Moon, H. Nakazawa, P. Rados, A. Rostomyan, J. G. Shiu, S. Skambraks, T. A. Sheng, Y. Sue, K. Unger, C. H. Wang, E. Won, and J. Yin. 2020. Development of the Level-1 track trigger with Central Drift Chamber detector in Belle II experiment and its performance in SuperKEKB 2019 Phase 3 operation. *Journal of Instrumentation* 15, 06 (jun 2020), C06063. doi:10.1088/1748-0221/15/06/C06063
- [47] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1 (1989), 541–551. <https://api.semanticscholar.org/CorpusID:41312633>
- [48] Fengfu Li, Bin Liu, Xiaoxing Wang, Bo Zhang, and Junchi Yan. 2022. Ternary Weight Networks. *arXiv:1605.04711 [cs.CV]*
- [49] Zhuo Li, Hengyi Li, and Lin Meng. 2023. Model Compression for Deep Neural Networks: A Survey. *Computers* 12, 3 (2023). doi:10.3390/computers12030060
- [50] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards Accurate Binary Convolutional Neural Network. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/b1a59b315f9a3002ce38bbe070ec3f5-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/b1a59b315f9a3002ce38bbe070ec3f5-Paper.pdf)
- [51] Alejandro Linares-Barranco, Fernando Perez-Peña, Diederik Paul Moeys, Francisco Gomez-Rodriguez, Gabriel Jimenez-Moreno, Shih-Chii Liu, and Tobi Delbruck. 2019. Low Latency Event-Based Filtering and Feature Extraction for Dynamic Vision Sensors in Real-Time FPGA Applications. *IEEE Access* 7 (2019), 134926–134942. doi:10.1109/ACCESS.2019.2941282
- [52] Peter Lincoln, Alex Blate, Montek Singh, Turner Whitted, Andrei State, Anselmo Lastra, and Henry Fuchs. 2016. From Motion to Photons in 80 Microseconds: Towards Minimal Latency for Virtual and Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* 22, 4 (2016), 1367–1376. doi:10.1109/TVCG.2016.2518038
- [53] Binglei Lou, Richard Rademacher, David Boland, and Philip H.W. Leong. 2024. PolyLUT-Add: FPGA-based LUT Inference with Wide Inputs. In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. 149–155. doi:10.1109/FPL64840.2024.00029
- [54] Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. 2020. AutoQ: Automated Kernel-Wise Neural Network Quantization. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=rygfn4twS>
- [55] Gaurav Menghani. 2021. Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better. *Comput. Surveys* 55 (2021), 1–37. <https://api.semanticscholar.org/CorpusID:235446458>
- [56] Etienne Mercier. 2025. The need for speed. <https://www.globaltrading.net/the-need-for-speed/>
- [57] Shahnam Mirzaei, Anup Hosangadi, and Ryan Kastner. 2006. FPGA Implementation of High Speed FIR Filters Using Add and Shift Method. In *2006 International Conference on Computer Design*. IEEE, 308–313. doi:10.1109/ICCD.2006.4380833
- [58] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2011).
- [59] S Neuhaus, S Skambraks, F Abudinen, Y Chen, M Feindt, R Frühwirth, M Heck, C Kiesling, A Knoll, S Paul, and J Schieck. 2015. A neural network z-vertex trigger for Belle II. *Journal of Physics: Conference Series* 608, 1 (apr 2015), 012052. doi:10.1088/1742-6596/608/1/012052
- [60] Wei Niu, Zhengang Li, Xiaolong Ma, Peiyan Dong, Gang Zhou, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2022. GRIM: A General, Real-Time Deep Learning Inference Framework for Mobile Devices Based on Fine-Grained Structured Weight Sparsity. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 10\_Part\_1 (oct 2022), 6224–6239. doi:10.1109/TPAMI.2021.3089687
- [61] Patrick Odagiu, Zhiqiang Que, Javier Duarte, Johannes Haller, Gregor Kasieczka, Artur Lobanov, Vladimir Loncar, Wayne Luk, Jennifer Ngadiuba, Maurizio Pierini, Philipp Rincke, Arpita Seksaria, Sioni Summers, Andre Sznajder, Alexander Tapper, and Thea K Arrestad. 2024. Ultrafast jet classification at the HL-LHC. *Machine Learning: Science and Technology* 5, 3 (July 2024), 035017. doi:10.1088/2632-2153/ad5f10
- [62] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. 2018. Value-Aware Quantization for Training and Inference of Neural Networks. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 608–624.
- [63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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 *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Hanna M. Wallach, Author PictureHugo Larochelle, Author PictureAlina Beygelzimer, Author PictureFlorence d'Alché Buc, and Author PictureEmily B. Fox (Eds.). Curran Associates Inc., Red Hook, NY, USA, Article 721, 12 pages.
- [64] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytso. 2020. HLS4ML LHC Jet dataset (150 particles). doi:10.5281/zenodo.3602260
- [65] Zhongnan Qu, Zimu Zhou, Yun Cheng, and Lothar Thiele. 2020. Adaptive loss-aware quantization for multi-bit networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7988–7997.
- [66] Zhiqiang Que, Jose G. F. Coutinho, Ce Guo, Hongxiang Fan, and Wayne Luk. 2026. MetaML-Pro: Cross-Stage Design Flow Automation for Efficient Deep Learning Acceleration. *ACM Transactions on Reconfigurable Technology and Systems* (2026).
- [67] Zhiqiang Que, Hongxiang Fan, Marcus Loo, He Li, Michaela Blott, Maurizio Pierini, Alexander Tapper, and Wayne Luk. 2024. LL-GNN: Low Latency Graph Neural Networks on FPGAs for High Energy Physics. *ACM Transactions on Embedded Computing Systems* 23, 2 (March 2024), 1–28. doi:10.1145/3640464
- [68] Zhiqiang Que, Shuo Liu, Markus Rognlien, Ce Guo, Jose G. F. Coutinho, and Wayne Luk. 2023. MetaML: Automating Customizable Cross-Stage Design-Flow for Deep Learning Acceleration. In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. 248–252. doi:10.1109/FPL60245.2023.00042
- [69] Zhiqiang Que, Marcus Loo, Hongxiang Fan, Maurizio Pierini, Alexander Tapper, and Wayne Luk. 2022. Optimizing graph neural networks for jet tagging in particle physics on FPGAs. In *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 327–333.
- [70] Zhiqiang Que, Chang Sun, Sudarshan Paramesvaran, Emyr Clement, Katerina Karakoulaki, Christopher Brown, Lauri Laatu, Arianna Cox, Alexander Tapper, Wayne Luk, and Maria Spiropulu. 2025. JEDI-linear: Fast and Efficient Graph Neural Networks for Jet Tagging on FPGAs. In *2025 International Conference on Field Programmable Technology (FPT)*. IEEE.
- [71] Zhiqiang Que, Erwei Wang, Umar Marikar, Eric Moreno, Jennifer Ngadiuba, Hamza Javed, Bartłomiej Borzyszkowski, Thea Arrestad, Vladimir Loncar, Sioni Summers, Maurizio Pierini, Peter Y Cheung, and Wayne Luk. 2021. Accelerating recurrent neural networks for gravitational wave experiments. In *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 117–124.
- [72] Benjamin Ramhorst, George A. Constantinides, and Vladimir Loncar. 2023. FPGA Resource-aware Structured Pruning for Real-Time Neural Networks. *arXiv:2308.05170v1 [cs.AR]* <https://arxiv.org/abs/2308.05170v1>
- [73] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 525–542.
- [74] A. Rios-Navarro, S. Guo, G. Abbarajithan, K. Vijayakumar, A. Linares-Barranco, T. Aarrestad, R. Kastner, and T. Delbruck. 2023. Within-Camera Multilayer Perceptron DVS Denoising. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE Computer Society, Los Alamitos, CA, USA, 3933–3942. doi:10.1109/CVPRW59228.2023.00409
- [75] Jan-Frederik Schulte, Benjamin Ramhorst, Chang Sun, Jovan Mitrevski, Nicolò Ghielmetti, Enrico Lupi, Dimitrios Danopoulos, Vladimir Loncar, Javier Duarte,

- David Burnette, Lauri Laatu, Stylianos Tzelepis, Konstantinos Axiotis, Quentin Berthet, Haoyan Wang, Paul White, Suleyman Demirsoy, Marco Colombo, Thea Aarrestad, Sioni Summers, Maurizio Pierini, Giuseppe Di Guglielmo, Jennifer Ngadiuba, Javier Campos, Ben Hawks, Abhijith Gandrakota, Farah Fahim, Nhan Tran, George Constantinides, Zhiqiang Que, Wayne Luk, Alexander Tapper, Duc Hoang, Noah Paladino, Philip Harris, Bo-Cheng Lai, Manuel Valentin, Ryan Forelli, Seda Ogrenci, Lino Gerlach, Rian Flynn, Mia Liu, Daniel Diaz, Elham Khoda, Melissa Quinnan, Russell Solares, Santosh Parajuli, Mark Neubauer, Christian Herwig, Ho Fung Tsoi, Dylan Rankin, Shih-Chieh Hsu, and Scott Hauck. 2026. hls4ml: A Flexible, Open-Source Platform for Deep Learning Acceleration on Reconfigurable Hardware. *ACM Transactions on Reconfigurable Technology and Systems* (2026).
- [76] Raghubir Singh and Sukhpal Singh Gill. 2023. Edge AI: A survey. *Internet of Things and Cyber-Physical Systems* 3 (2023), 71–92. doi:10.1016/j.iotcps.2023.02.004
- [77] Wilson Snyder, Paul Wasson, Duane Galbi, and et al. 2025. Verilator. <https://verilator.org> If you use this software, please cite it using the metadata from this file..
- [78] Chang Sun, Takumi Nakajima, Yuki Mitsumori, Yasuyuki Horii, and Makoto Tomoto. 2023. Fast muon tracking with machine learning implemented in FPGA. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1045 (Jan. 2023), 167546. doi:10.1016/j.nima.2022.167546
- [79] Chang Sun, Jennifer Ngadiuba, Maurizio Pierini, and Maria Spiropulu. 2025. Fast Jet Tagging with MLP-Mixers on FPGAs. *Machine Learning: Science and Technology* (2025). <http://iopscience.iop.org/article/10.1088/2632-2153/adf596>
- [80] Chang Sun, Zhiqiang Que, Vladimir Loncar, Wayne Luk, and Maria Spiropulu. 2026. da4ml: Distributed arithmetic for real-time neural networks on fpgas. *ACM Transactions on Reconfigurable Technology and Systems* (2026). doi:10.1145/3777387
- [81] Mengshu Sun, Zhengang Li, Alec Lu, Yanyu Li, Sung-En Chang, Xiaolong Ma, Xue Lin, and Zhenman Fang. 2022. FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization. *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2022). doi:10.1145/3490422.3502364
- [82] Ole Tange. 2023. GNU Parallel 20240122 ('Frederik X'). doi:10.5281/zenodo.10558745 GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them..
- [83] The ATLAS Collaboration. 2017. *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*. Technical Report. CERN, Geneva. doi:10.17181/CERN.2LBB.4IAL
- [84] The Belle II Collaboration. 2010. Belle II Technical Design Report. arXiv:1011.0352 [physics.ins-det] <https://arxiv.org/abs/1011.0352>
- [85] The Belle II Collaboration. 2019. The Belle II Physics Book. *Progress of Theoretical and Experimental Physics* 2019, 12 (Dec. 2019). doi:10.1093/ptep/ptz106
- [86] The CMS Collaboration. 2020. *The Phase-2 Upgrade of the CMS Level-1 Trigger*. Technical Report. CERN, Geneva. <https://cds.cern.ch/record/2714892> Final version.
- [87] The LHC Study Group. 1995. *The Large Hadron Collider, Conceptual Design*. Technical Report. CERN/AC/95-05 (LHC) Geneva.
- [88] Stephen Tridgell, Martin Kumm, Martin Hardieck, David Boland, Duncan Moss, Peter Zipf, and Philip HW Leong. 2019. Unrolling Ternary Neural Networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 12, 4 (2019), 1–23.
- [89] Ho Fung Tsoi, Vladimir Loncar, Sridhara Dasu, and Philip Harris. 2025. SymbolNet: neural symbolic regression with adaptive dynamic pruning for compression. *Machine Learning: Science and Technology* 6, 1 (Jan. 2025), 015021. doi:10.1088/2632-2153/adaad8
- [90] Yaman Umuroglu, Yash Akhauri, Nicholas J. Fraser, and Michaela Blott. 2020. LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications. *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)* (2020), 291–297. doi:10.1109/FPL50879.2020.00055
- [91] Yaman Umuroglu, Hendrik Borrás, Vladimir Loncar, Sioni Summers, and Javier Duarte. 2024. QONNX. doi:10.5281/zenodo.14537023
- [92] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Visser. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM Press. arXiv:1612.07119 doi:10.1145/3020078.3021744
- [93] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 <http://arxiv.org/abs/1706.03762>
- [94] Erwei Wang, James J Davis, Peter YK Cheung, and George A Constantinides. 2020. LUTNet: Learning FPGA configurations for highly efficient neural network inference. *IEEE Trans. Comput.* 69, 12 (2020), 1795–1808.
- [95] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2020. Hardware-Centric AutoML for Mixed-Precision Quantization. *International Journal of Computer Vision* 128, 8 (01 Sep 2020), 2035–2048. doi:10.1007/s11263-020-01339-6
- [96] Sinong Wang, Belinda Z. Li, Madian Khabisa, Han Fang, and Hao Ma. 2020. Linformer: Self-Attention with Linear Complexity. arXiv:2006.04768 [cs.LG] <https://arxiv.org/abs/2006.04768>
- [97] Jason Weitz, Dmitri Demler, Luke McDermott, Nhan Tran, and Javier Duarte. 2025. Neural Architecture Codesign for Fast Physics Applications. arXiv:2501.05515 [cs.LG] <https://arxiv.org/abs/2501.05515>
- [98] Olivia Weng, Marta Andronic, Danial Zuberi, Jiaqing Chen, Caleb Geniesse, George A. Constantinides, Nhan Tran, Nicholas J. Fraser, Javier Mauricio Duarte, and Ryan Kastner. 2025. Greater than the Sum of its LUTs: Scaling Up LUT-based Neural Networks with AmigoLUT. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) (FPGA '25). Association for Computing Machinery, New York, NY, USA, 25–35. doi:10.1145/3706628.3708874
- [99] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search. *CoRR* abs/1812.00090 (2018). arXiv:1812.00090 <http://arxiv.org/abs/1812.00090>
- [100] Bin Wu, Xinyu Wu, Peng Li, Youbing Gao, Jiangbo Si, and Naofal Al-Dhahir. 2024. Efficient FPGA Implementation of Convolutional Neural Networks and Long Short-Term Memory for Radar Emitter Signal Recognition. *Sensors* 24, 3 (2024). doi:10.3390/s24030889
- [101] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. Alternating Multi-bit Quantization for Recurrent Neural Networks. *CoRR* abs/1802.00150 (2018). arXiv:1802.00150 <http://arxiv.org/abs/1802.00150>
- [102] Yang Yang, Yury Kartynnik, Yunpeng Li, Jiuqiang Tang, Xing Li, George Sung, and Matthias Grundmann. 2024. Streamvc: Real-time low-latency voice conversion. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 11016–11020.
- [103] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. 2019. PyHessian: Neural Networks Through the Lens of the Hessian. *2020 IEEE International Conference on Big Data (Big Data)* (2019), 581–590. <https://api.semanticscholar.org/CorpusID:209376531>
- [104] Dongqing Zhang, Jialong Yang, Dongqiang Yi, and Gang Hua. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. *CoRR* abs/1807.10029 (2018). arXiv:1807.10029 <http://arxiv.org/abs/1807.10029>
- [105] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR* abs/1606.06160 (2016). arXiv:1606.06160 <http://arxiv.org/abs/1606.06160>
- [106] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained Ternary Quantization. arXiv:1612.01064 [cs.LG]