A2Q: Accumulator-Aware Quantization with Guaranteed Overflow Avoidance

Ian Colbert, Alessandro Pappalardo, Jakoba Petri-Koenig
Advanced Micro Devices, Inc.
{icolbert, alessand, jakobap}@amd.com

Abstract

We present accumulator-aware quantization (A2Q), a novel weight quantization method designed to train quantized neural networks (QNNs) to avoid overflow when using low-precision accumulators during inference. A2Q introduces a unique formulation inspired by weight normalization that constrains the ℓ_1-norm of model weights according to accumulator bit width bounds that we derive. Thus, in training QNNs for low-precision accumulation, A2Q also inherently promotes unstructured weight sparsity to guarantee overflow avoidance. We apply our method to deep learning-based computer vision tasks to show that A2Q can train QNNs for low-precision accumulators while maintaining model accuracy competitive with a floating-point baseline. In our evaluations, we consider the impact of A2Q on both general-purpose platforms and programmable hardware. However, we primarily target model deployment on FPGAs because they can be programmed to fully exploit custom accumulator bit widths. Our experimentation shows accumulator bit width significantly impacts the resource efficiency of FPGA-based accelerators. On average across our benchmarks, A2Q offers up to a 2.3x reduction in resource utilization over 32-bit accumulator counterparts with 99.2% of the floating-point model accuracy.

1. Introduction

Quantization is the process of reducing the range and precision of the numerical representation of data. When applied to the weights and activations of neural networks, integer quantization reduces compute and memory requirements, usually in exchange for minor reductions in model accuracy [14, 19, 20, 47]. During inference, most of the compute workload is concentrated in operators such as convolutions and matrix multiplications, whose products are typically accumulated into 32-bit registers that we refer to as accumulators. It has been shown that reducing accumulators to 16 bits on CPUs and ASICs can increase inference throughput and bandwidth efficiency by up to 2x [9, 45], and reducing to 8 bits can improve energy efficiency by over 4x [32]. However, exploiting such an optimization is highly non-trivial as doing so incurs a high risk of overflow. Due to wraparound two's complement arithmetic, this can introduce numerical errors that degrade model accuracy [32].

Previous works have sought to either reduce the risk of overflow [25, 37, 45] or mitigate its impact on model accuracy [32]. However, such approaches struggle to maintain accuracy when overflow occurs too frequently [32], and are unable to support applications that require guaranteed arithmetic correctness, such as finite-precision fully homomorphic encryption computations [28, 40]. Thus, we are motivated to avoid overflow altogether. As the first principled approach to guarantee overflow avoidance, we provide theoretical motivation in Section 3, where we derive comprehensive accumulator bit width bounds with finer granularity than existing literature. In Section 4, we present accumulator-aware quantization (A2Q); a novel method designed to train quantized neural networks (QNNs) to use low-precision accumulators during inference without any risk of overflow. In Section 5, we show that our method not only prepares QNNs for low-precision accumulation, but also inherently increases the sparsity of the weights.

While our results have implications for general-purpose platforms such as CPUs and GPUs, we primarily target model deployment on custom FPGA-based inference accelerators. FPGAs allow bit-level control over every part of a low-precision inference accelerator and can therefore take advantage of custom data types to a greater extent than general-purpose platforms, which are often restricted to power-of-2 bit widths. In doing so, we show in Section 5 that reducing the bit width of the accumulator can in turn improve the overall trade-off between resource utilization and model accuracy for custom low-precision accelerators.

To the best of our knowledge, we are the first to explore the use of low-precision accumulators to improve the design efficiency of FPGA-based QNN inference accelerators. As such, we integrate A2Q into the open-source Brevitas quantization library [35] and FINN compiler [1] to demonstrate an end-to-end flow for training QNNs for low-precision accumulation and generating custom streaming architectures targeted for AMD-Xilinx FPGAs.

2. Background

2.1. Quantization-Aware Training (QAT)

The standard operators used to emulate quantization during training rely on uniform affine mappings from a high-precision real number to a low-precision quantized number [20]. The quantizer (Eq. 1) and dequantizer (Eq. 2) are parameterized by zero-point z and scaling factor s. Here, z is an integer value that ensures that zero is exactly represented in the quantized domain, and s is a strictly positive real scalar that corresponds to the resolution of the quantization function. Scaled values are rounded to the nearest integers using half-way rounding, denoted by $\lfloor \cdot \rceil$, and elements that exceed the largest supported values in the quantized domain are clipped such that $\text{clip}(x; n, p) = \min(\max(x; n); p)$, where n and p depend on the data type of x. For signed integers of bit width b, we assume $n = -2^{b-1}$ and $p = 2^{b-1} - 1$. For unsigned integers, we assume $n = 0$ and $p = 2^b - 1$ when unsigned.

$$\text{quantize}(x; s, z) := \text{clip}(\left\lfloor \frac{x}{s} \right\rceil + z; n, p) \quad (1)$$

$$\text{dequantize}(x; s, z) := s \cdot (x - z) \quad (2)$$

It has become common to use unique scaling factors for each of the output channels of the learned weights to adjust for varied dynamic ranges [31]. However, extending this strategy to activations requires either storing partial sums or introducing additional control logic. As such, it is standard practice to use per-tensor scaling factors for activations and per-channel scaling factors on the weights. It is also common to constrain the weight quantization scheme such that $z = 0$ [14]. Eliminating these zero points reduces the computational overhead of cross-terms during integer-only inference [21]. During training, the straight-through estimator (STE) [3] is used to allow local gradients to permeate the rounding function such that $\nabla_x \lfloor x \rceil = 1$ everywhere, where ∇_x denotes the local gradient with respect to x.

2.2. Low-Precision Accumulation

As activations are propagated through the layers of a QNN, the intermediate partial sums resulting from convolutions and matrix multiplications are typically accumulated in a high-precision register before being requantized and passed to the next layer, as depicted in Fig. 1. Reducing the precision of the accumulator incurs a high risk of overflow which, due to wraparound two's complement arithmetic, introduces numerical errors that can degrade model accuracy [32]. As shown in Fig. 2, the rate of overflows per dot product often grows exponentially as the accumulator bit width is reduced (please see Appendix A for details). The increased overflow rate introduces numerical errors that proportionally increase the mean absolute error on the logits, decreasing classification accuracy. The industry standard for avoiding overflow is to either use high-precision accumulators or naïvely saturate values as they are accumulated. However, such clipping can still: (1) introduce numerical errors that cascade when propagated through a QNN; and (2) require additional logic that can break associativity while adding to latency and area requirements [2]. In breaking associativity, the final result of the dot product is made dependent on the order of additions. This can introduce non-deterministic errors when modern processors use optimizations that improve hardware utilization by reordering operations [13, 22] (please see Appendix A.1 for details). This further motivates us to train QNNs to completely avoid overflow rather than simply reduce its impact on model accuracy. This way, A2Q entirely circumvents these numerical errors while delivering improved resource efficiency and increased accuracy.

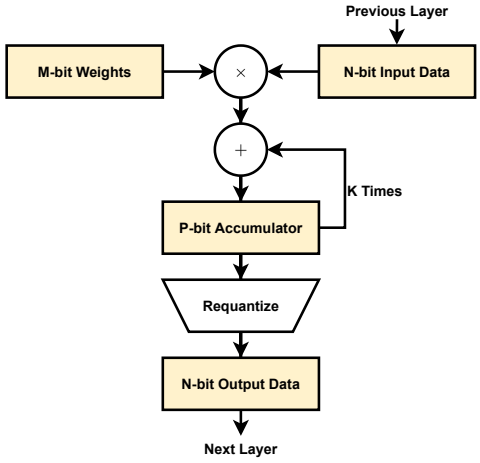


Figure 1: A simplified illustration of fixed-point arithmetic in neural network inference. The accumulator bit width (P) needs to be wide enough to fit the dot product between the M-bit weight vector and the N-bit input vector, which are assumed to both be K-dimensional.

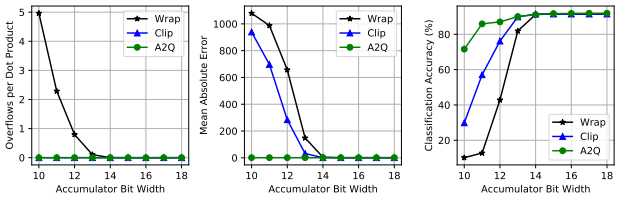


Figure 2: We evaluate the impact of overflow as we reduce the accumulator bit width using a 1-layer QNN trained to classify binary MNIST [10] images using 8-bit weights. We show that using A2Q (**green dots**) to avoid overflow significantly improves model accuracy over both wraparound arithmetic (**black stars**) and clipping (**blue triangles**) when using extremely low-precision accumulators.

3. Accumulator Bit Width Bounds

Figure 1 illustrates a simplified abstraction of accumulation in QNN inference. To avoid overflow, the register storing the accumulated values needs to be wide enough to not only contain the result of the dot product, but also all intermediate partial sums.

Consider the dot product of input data $\boldsymbol{x}$ and learned weights $\boldsymbol{w}$, which are each K-dimensional vectors of integers. Let y be the scalar result of their dot product given by Eq. 3, where x_i and w_i denote element i of vectors $\boldsymbol{x}$ and $\boldsymbol{w}$, respectively. Since the representation range of y is bounded by that of $\boldsymbol{x}$ and $\boldsymbol{w}$, we use their ranges to derive lower bounds on the bit width P of the accumulation register, or accumulator.

$$y = \sum_{i=1}^{K} x_i w_i \tag{3}$$

It is common for input data to be represented with unsigned integers either when following activation functions with non-negative dynamic ranges (*e.g.*, rectified linear units, or ReLUs), or when an appropriate zero point is adopted (*i.e.*, asymmetric quantization). Otherwise, signed integers are used. Since weights are most often represented with signed integers, we assume the accumulator is always signed in our work. Therefore, given that the scalar result of the dot product between $\boldsymbol{x}$ and $\boldsymbol{w}$ is a P-bit integer defined by Eq. 3, it follows that $\sum_{i=1}^{K} x_i w_i$ is bounded such that:

$$-2^{P-1} \leq \sum_{i=1}^{K} x_i w_i \leq 2^{P-1} - 1 \tag{4}$$

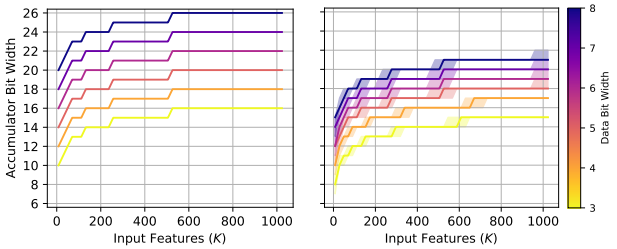
To satisfy both sides of this double inequality, it follows that $|\sum_{i=1}^{K} x_i w_i| \leq 2^{P-1} - 1$. However, the accumulator needs to be wide enough to not only store the final result of the dot product, but also all intermediate partial sums.

Since input data is not known *a priori*, our bounds must consider the worst-case values for every MAC. Thus, because the magnitude of the sum of products is upper-bounded by the sum of the product of magnitudes, it follows that if $\sum_{i=1}^{K} |x_i||w_i| \leq 2^{P-1} - 1$, then the dot product between $\boldsymbol{x}$ and $\boldsymbol{w}$ fits into a P-bit accumulator without numerical overflow, as shown below.

$$|\sum_i x_i w_i| \leq \sum_i |x_i w_i| \leq \sum_i |x_i||w_i| \leq 2^{P-1} - 1 \tag{5}$$

3.1. Deriving Lower Bounds Using Data Types

The worst-case values for each MAC can naïvely be inferred from the representation range of the data types used. When x_i and w_i are signed integers, their magnitudes are bounded such that $|x_i| \leq 2^{N-1}$ and $|w_i| \leq 2^{M-1}$, respectively. In scenarios where x_i is an unsigned integer, the magnitude of each input value is upper-bounded such that $|x_i| \leq 2^N - 1$; however, we consider the case where $|x_i| \leq 2^N$ to simplify our derivation[1]. Combining these upper bounds, it follows that $|x_i| \leq 2^{N - \mathbb{1}_{\text{signed}}(\boldsymbol{x})}$, where $\mathbb{1}_{\text{signed}}(\boldsymbol{x})$ is an indicator function that returns 1 if and only if $\boldsymbol{x}$ is a vector of signed integers.



(a) Using Data Type (b) Using Weights

Figure 3: We visualize the differences between our accumulator bit width bounds as we vary the size of the dot product (K) as well as the bit width of both the weights (M) and inputs (N), which we jointly refer to as "data bit width."

Building from Eq. 5, it follows that the sum of the product of the magnitudes is bounded such that:

$$\sum_{i=1}^{K} |x_i||w_i| \leq K \cdot 2^{N+M-1-\mathbb{1}_{\text{signed}}(\boldsymbol{x})} \leq 2^{P-1} - 1 \tag{6}$$

Taking the log of both sides of Eq. 6, we can derive a lower bound on the accumulator bit width P:

$$\log_2 \left(2^{\log_2(K)+N+M-1-\mathbb{1}_{\text{signed}}(\boldsymbol{x})} + 1\right) + 1 \leq P \tag{7}$$

This simplifies to the following lower bound on P:

$$P \geq \alpha + \phi(\alpha) + 1 \tag{8}$$

$$\alpha = \log_2(K) + N + M - 1 - \mathbb{1}_{\text{signed}}(\boldsymbol{x}) \tag{9}$$

$$\phi(\alpha) = \log_2(1 + 2^{-\alpha}) \tag{10}$$

In Fig. 3a, we visualize this bound assuming that $\boldsymbol{x}$ is a vector of unsigned integers. There, we show how the lower bound on the accumulator bit width increases as we vary the size of the dot product (K) as well as the bit width of both the weights and input activations.

3.2. Deriving Lower Bounds Using Weights

Since learned weights are frozen during inference time, we can use knowledge of their magnitudes to derive a tighter lower bound on the accumulator bit width. Building again from Eq. 5, the sum of the product of magnitudes is bounded by Eq. 11, where $\|\boldsymbol{w}\|_1$ denotes the standard ℓ_1-norm over vector $\boldsymbol{w}$.

$$\sum_{i=1}^{K} |x_i||w_i| \leq 2^{N - \mathbb{1}_{\text{signed}}(\boldsymbol{x})} \cdot \|\boldsymbol{w}\|_1 \leq 2^{P-1} - 1 \tag{11}$$

[1]Note that our simplification of the upper bound for unsigned input data means that the lower bound on the accumulator is not as tight as possible, but it does not compromise overflow avoidance.

Accounting for $\|\boldsymbol{w}\|_1$ in our derivation allows us to tighten the lower bound on P as follows:

$$P \geq \beta + \phi(\beta) + 1 \tag{12}$$

$$\beta = \log_2(\|\boldsymbol{w}\|_1) + N - \mathbb{1}_{\text{signed}}(\boldsymbol{x}) \tag{13}$$

$$\phi(\beta) = \log_2(1 + 2^{-\beta}) \tag{14}$$

In Fig. 3b, we visualize this bound, again assuming that $\boldsymbol{x}$ is a vector of unsigned integers. Because Eq. 13 is dependent on the values of the learned weights, we randomly sample each K-dimensional vector from a discrete Gaussian distribution and show the median accumulator bit width along with the minimum and maximum observed over 1000 random samples. Across K, M, and N, we visualize how using knowledge of the weights provides a tighter lower bound on the accumulator bit width than using data types.

4. A2Q: Accumulator-Aware Quantization

To train QNNs to use low-precision accumulators without overflow, we use weight normalization as a means of constraining learned weights $\boldsymbol{w}$ to satisfy the bound derived in Section 3.2. Building from Eq. 11, we transform our lower bound on accumulator bit width P to be the upper bound on the ℓ_1-norm of $\boldsymbol{w}$ given by Eq. 15. Note that because each output neuron requires its own accumulator, this upper bound needs to be enforced channelwise.

$$\|\boldsymbol{w}\|_1 \leq \left(2^{P-1} - 1\right) \cdot 2^{\mathbb{1}_{\text{signed}}(\boldsymbol{x}) - N} \tag{15}$$

4.1. Constructing Our Quantization Operator

Weight normalization, as originally proposed by Salimans *et al.* [38], reparameterizes each weight vector $\boldsymbol{w}$ in terms of a parameter vector $\boldsymbol{v}$ and a scalar parameter g as given in Eq. 16, where $\|\boldsymbol{v}\|_2$ is the Euclidean norm of the K-dimensional vector $\boldsymbol{v}$ [38]. This simple reparameterization fixes the Euclidean norm of weight vector $\boldsymbol{w}$ such that $\|\boldsymbol{w}\|_2 = g$, which enables the magnitude and direction to be independently learned.

$$\boldsymbol{w} = g \cdot \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2} \tag{16}$$

Inspired by this formulation, we reparameterize our quantizer such that each weight vector $\boldsymbol{w}$ is represented in terms of parameter vectors $\boldsymbol{v}$ and $\boldsymbol{g}$. Similar to the standard weight normalization formulation, this reparameterization decouples the norm from the weight vector; however, unlike the standard formulation, our norm is learned for each output channel rather than per-tensor. To enforce our constraint during QAT, we also replace the per-tensor ℓ_2-norm with a per-channel ℓ_1-norm. This reparameterization, given by Eq. 17, allows for the ℓ_1-norm of weight vector $\boldsymbol{w}$ to be independently learned per-channel such that $g_i = \|\boldsymbol{w}_i\|_1$ for all $i \in \{1, \cdots, C\}$. Here, $\boldsymbol{w}_i$ denotes the weights of channel i and g_i denotes element i in parameter vector $\boldsymbol{g}$ for a given layer with C output channels.

$$\boldsymbol{w}_i = g_i \cdot \frac{\boldsymbol{v_i}}{\|\boldsymbol{v_i}\|_1} \quad \forall\, i \in \{1, \cdots, C\} \tag{17}$$

Similar to the standard weight quantizer, our weight normalization-based quantization operator relies on a uniform affine mapping from the high-precision real domain to the low-precision quantized domain using learned per-channel scaling factors $\boldsymbol{s} = \{s_i\}_{i=1}^{C}$. Thus, by constraining g_i to satisfy Eq. 18, we can learn quantized weights that satisfy our accumulator bit width bound and avoid overflow.

$$g_i \leq s_i \cdot \left(2^{P-1} - 1\right) \cdot 2^{\mathbb{1}_{\text{signed}}(\boldsymbol{x}) - N} \tag{18}$$

Below, we articulate our weight normalization-based quantizer. For clarity and convenience of notation, we consider a layer with one output channel (*i.e.*, $C = 1$) such that parameter vectors $\boldsymbol{g} = \{g_i\}_{i=1}^{C}$ and $\boldsymbol{s} = \{s_i\}_{i=1}^{C}$ can be represented as scalars g and s, respectively.

$$\text{quantize}(\boldsymbol{w}; s, z) := \text{clip}(\left\lfloor \frac{g}{s} \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_1} \right\rfloor + z; n, p) \tag{19}$$

We construct **accumulator-aware quantization (A2Q)** from our weight normalization-based quantizer (Eq. 19) and the standard dequantizer (Eq. 2). During training, A2Q applies the following four elementwise operations in order: scale, round, clip, then dequantize. As is standard practice, we eliminate the zero points in our mapping such that $z = 0$. We use an exponential parameterization of both the scaling factor $s = 2^d$ and the norm parameter $g = 2^t$, where d and t are both log-scale parameters to be learned through stochastic gradient descent. This is similar to the work of [21] with the difference that we consider the more common scenario of floating-point scaling factors. The scaled tensors are then rounded towards zero, which we denote by $\lfloor \cdot \rfloor$, to prevent any upward rounding that may cause the norm to increase past our constraint[2]. Note that this is another difference from the conventional quantization operators, which primarily use half-way rounding [14, 21]. Finally, once scaled and rounded, the elements in the tensor are then clipped and dequantized. To update learnable parameters throughout training, we use STE [3] as is common practice.

$$q(\boldsymbol{w}; s) := \text{clip}\left(\left\lfloor \frac{g}{s} \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_1} \right\rfloor; n, p\right) \cdot s \tag{20}$$

$$\text{where } s = 2^d \tag{21}$$

$$\text{and } g = 2^{\min(T,t)} \tag{22}$$

$$\text{and } T = \mathbb{1}_{\text{signed}}(\boldsymbol{x}) + \log_2(2^{P-1} - 1) + d - N \tag{23}$$

[2]It is important to note that rounding towards zero is functionally different from floor or ceiling rounding [27].

We apply A2Q to only the weights of a QNN. To avoid t getting stuck when $t > T$, we introduce the following regularization penalty for the l-th layer of the network: $R_l = \sum_i \max\{t_i - T_i, 0\}$. This penalty is imposed on every hidden layer and combined into one regularizer: $\mathcal{L}_{\text{reg}} = \sum_l R_l$. When quantizing our activations, we use the standard quantization methods discussed in Section 2.1. All activations that follow non-negative functions (*i.e.*, ReLU) are represented using unsigned integers, otherwise they are signed.

5. Experiments

Without guaranteed overflow avoidance, one cannot reliably design hardware accelerators around low-precision accumulators. Therefore, in our experiments, we do not compare against methods that cannot provide such guarantees. Given that, we consider two scenarios in the following evaluations. In Section 5.2, we optimize QNNs for accumulator-constrained processors, where the goal is to maximize task performance given a user-defined accumulator bit width. Such a scenario is a direct application of our method and has implications for both accelerating inference on general-purpose platforms [9, 32, 45] and reducing the computational overhead of homomorphic encryption arithmetic [28, 40]. In Section 5.3, we optimize QNNs for overall resource utilization within a hardware-software (HW-SW) co-design setting. In this scenario, the goal is to maximize task performance given a user-defined hardware resource budget. Our experiments show that including the accumulator bit width as part of the design space can improve the trade-off between resources and accuracy. We target model deployment on custom FPGA-based accelerators, rather than CPUs or GPUs, as they allow bit-level control over every part of the network and can therefore take advantage of custom data types to a greater extent. To do so, we adopt FINN [4, 41], an open-source framework designed to generate specialized streaming architectures for QNN inference acceleration on AMD-Xilinx FPGAs. We build on top of FINN v0.8.1 [1] and open-source our A2Q implementation[3] as part of Brevitas v0.10 [35].

5.1. Experiment Setup

We apply A2Q to the following two computer vision tasks: (1) image classification on CIFAR10 [24] using MobileNetV1 [18] and ResNet18 [15]; and (2) single-image super resolution on BSD300 [29] using ESPCN [39] and UNet [36]. For each model, we measure task performance over the test dataset, where image classification and single-image super resolution models are evaluated using top-1 classification accuracy and peak signal-to-noise ratio (PSNR), respectively. We include more details on model and training settings in Appendix B.

[3] https://github.com/Xilinx/brevitas/tree/master/src/brevitas_examples

Throughout our experiments, we constrain our quantization design space to uniform-precision models. For every hidden layer in each network, we enforce the same weight, activation, and accumulator bit width, respectively denoted as M, N, and P. We perform a grid search over our quantization design space, focusing our attention on weight and activation bit widths between 5 and 8 bits. Doing so allows even comparisons across bit widths, as reducing the precision below 5 bits often requires unique hyperparameters to maximize performance. For each weight and activation bit width combination, we calculate the largest lower bound on the accumulator bit width for each model. In a model with L layers, this is determined by the data type bound of the layer with the largest dot product size K^*, where $K^* = \arg\max_{K_l} \{K_l\}_{l=0}^{L}$. Using this to guide our grid search over P for each model, we evaluate up to a 10-bit reduction in the accumulator bit width to create a total of 160 configurations per model.

5.2. Optimizing for Accumulator Constraints

To the best of our knowledge, A2Q is the first to allow a user to train a QNN to avoid overflow by only specifying a target accumulator bit width P. As an alternative, a designer can choose to heuristically reduce data bit widths based on our data type bound given by Eq. 8. Such an approach would still guarantee overflow avoidance for a user-defined P, but is a limited and indirect means of enforcing such a constraint. By heuristically manipulating data bit widths, the minimum attainable P is bounded by both the quantization design space and the architecture of the QNN because it is a function of M, N, and the size of the dot product K. Conversely, A2Q exposes P as an independent variable to be directly specified in the design, orthogonal to the rest of the design space. To compare the performance of models trained with A2Q against the baseline heuristic approach, we vary M, N, and P across our benchmark models. We visualize this comparison as a Pareto frontier in Fig. 4 and provide the floating-point task performance as reference. For each model and each algorithm, the Pareto frontier shows the maximum observed task performance for a given target accumulator bit width P.

It is important to note that, while this is not a direct comparison against the algorithm proposed by [9], the experiment is similar in principle. Unlike [9], we use the more advanced quantization techniques detailed in Section 2.1 and constrain our quantization design space to uniform-precision models. Within this design space, we observe that A2Q can push the accumulator bit width lower than what is attainable using current methods while also maintaining task performance. Furthermore, most models show less than a 1% performance drop with a 16-bit accumulator, which is often the target bit width for low-precision accumulation in general-purpose processors [9, 25, 45].

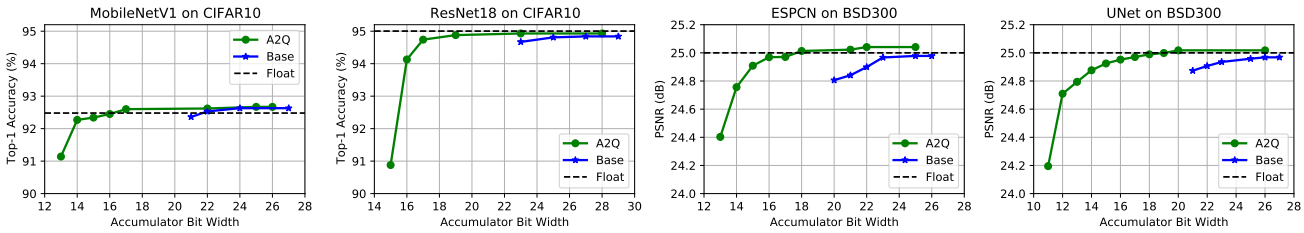


Figure 4: We visualize the trade-off between accumulator bit width and task performance using Pareto frontiers. We observe that A2Q (**green dots**) dominates the baseline QAT (**blue stars**) in all benchmarks, showing that we can reduce the accumulator bit width without sacrificing significant model performance even with respect to a floating-point baseline.

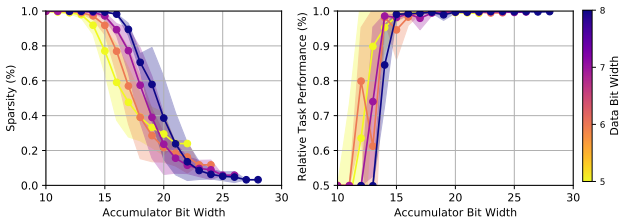


Figure 5: As a result of our ℓ_1-norm constraints, reducing the accumulator bit width exposes opportunities to exploit unstructured sparsity (left) without sacrificing model accuracy relative to the floating-point baseline (right). We observed that these trends were similar for each model; thus, to simplify our analysis, we provide the average and standard deviation calculated over all of our benchmark models.

5.2.1 Accumulator Impact on Model Sparsity

Given a target accumulator bit width P, our quantization method constrains the ℓ_1-norm of model weights according to the upper bound given by Eq. 15. Consequently, these constraints exponentially tighten as P is reduced (see Eqs. 18 and 23). Previous work has studied the use of ℓ_1-norm weight regularization as a means of introducing sparsity [6, 46]. We observe A2Q to inherently have a similar effect when training QNNs for low-precision accumulation.

In Fig. 5, we visualize how the sparsity and relative task performance are affected by reductions to P. We use the models from our grid search described in Section 5.1, but focus on configurations where the weight and input activation bit widths are the same (*i.e.*, $M = N$) to simplify our analysis. For each accumulator bit width, we plot the average sparsity and relative task performance observed across all of our benchmark models and provide the standard deviation. On average, we observe that constraining the hidden layers of our QNNs to use less than 32-bit accumulators yields up to 92% unstructured weight sparsity while maintaining 99.2% of the floating-point model accuracy.

5.3. Optimizing for Resource Utilization

To evaluate the trade-offs between resource utilization and task performance within our quantization design space, we target QNN model deployment on AMD-Xilinx FPGAs. We use the FINN framework to generate specialized hardware architectures that are individually customized for the network topology and data types used. This exposes control over the accumulators used in each layer so that we can take advantage of custom data types in our experiments. At the core of FINN is its compiler, which typically relies on FPGA look-up tables (LUTs) to perform multiply-and-accumulates (MACs) at low precision. In such scenarios, LUTs are often the resource bottleneck for the low-precision inference accelerators it generates. Therefore, we simplify our analysis by configuring the FINN compiler to assume that LUTs are the only type of resources available, and we leverage the LUT utilization estimates for each of QNN trained in our grid search. We include additional details on FINN in Appendix C.

Previous work has shown that reducing the precision of weights and activations provides resource utilization savings in exchange for minor reductions to model accuracy [41]. We observe that adding the accumulator bit width to the design space can improve this trade-off. To demonstrate the impact, we consider four HW-SW co-design settings. First, we consider a fixed accumulator bit width and remove P from the quantization design space discussed in Section 5.1. We use the baseline QAT to train each model and configure the generated accelerator to use a constant 32-bit accumulator for all layers. Second, we again use the baseline QAT, but configure FINN to use the minimum accumulator bit width P according to the data type bound (Eq. 8) per-layer. Third, we again use the baseline QAT, but configure the FINN compiler to further minimize P for each layer according to the ℓ_1-norm of the final weight values post-training (Eq. 13). Finally, we evaluate the end-to-end design flow when using A2Q to train QNNs for a specified weight, activation, and accumulator bit width. For each co-design setting, we visualize the trade-offs between re-

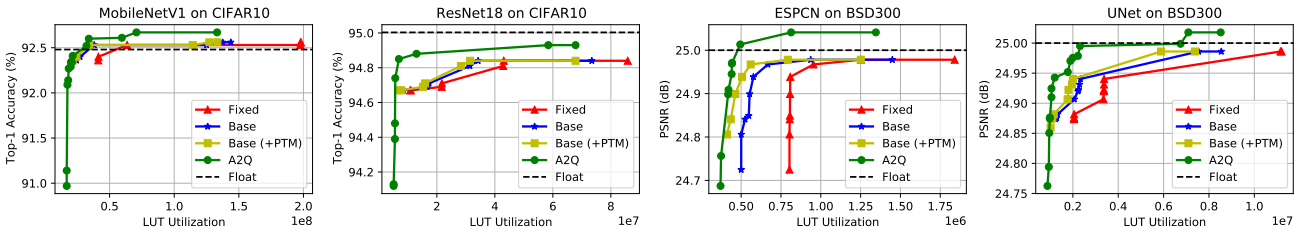


Figure 6: We visualize the trade-off between resource utilization and model accuracy using Pareto frontiers. We compare A2Q (**green dots**) to baseline QAT with: (1) a fixed accumulator bit width P (**red triangles**); (2) layer-wise selection of P using data type bounds (**blue dots**); and (3) post-training minimization (PTM) of P using weight values (**yellow squares**). We observe that A2Q provides a better trade-off between LUT utilization and task performance than existing baselines.

source utilization and task performance as a Pareto frontier in Fig. 6 and provide the floating-point task performance for reference. For each model and each co-design setting, the Pareto frontier shows the maximum observed task performance for a given total LUT utilization.

As expected, we observe that the layer-wise minimization of the accumulator bit width provides a better resource-to-accuracy trade-off than using a fixed 32-bit accumulator. We also observe that post-training minimization of P according to the final weight values provides consistent improvements over the data type bounds. Furthermore, our results show that using A2Q to train models for reduced accumulator bit widths provides a dominant Pareto frontier across all models. For the tasks at hand, A2Q pushes the best attainable task performance above the baselines. Thus, for a given target accuracy or resource budget, A2Q can offer a better trade-off between LUT utilization and task performance, confirming the benefits of including the accumulator bit width in the overall HW-SW co-design space.

5.3.1 Evaluating Resource Savings

Because we configure the FINN compiler to use LUTs wherever possible, we provide a deeper analysis of where the resource savings come from. To do so, we separate LUT utilization into compute and memory resources but ignore control flow overhead, which remains constant for each network because neural architecture is not impacted by the data types used. In Fig. 7, we visualize this break down for each of the Pareto optimal models that correspond to the A2Q Pareto frontier provided in Fig. 6.

We observe that LUT savings along the A2Q Pareto frontier come from reductions to both compute and memory resources. The accumulator bit width affects not only the width of the register storing intermediate partial sums, but also the width of the adder circuit doing the accumulation. Therefore, the reductions in compute resources primarily come from the reduced cost of MACs, which are directly impacted by the precision of the weights, inputs, and accumulators. Furthermore, because FINN implements activation functions as threshold comparisons, their resource utilization exponentially grows with the precision of the accumulator and output activations [4, 42]. Therefore, the reductions in memory resources are largely from the reduced storage costs of thresholds and intermediate activation buffers.

6. Discussion

The primary contribution of our work is A2Q, which trains QNNs to use low-precision accumulators during inference without any risk of overflow. Without guaranteed overflow avoidance, one cannot reliably design hardware accelerators around low-precision accumulation. In such scenarios, empirical estimates of overflow impact rely on *a priori* knowledge of input data, which is impractical to assume in many real-world use cases. Our guarantees allow for models and hardware to be jointly co-designed for low-precision accumulation. A2Q exposes the accumulator bit width P as an independent variable that can be specified by a user. Our experiments show that including the accumulator bit width in the quantization design space can improve the trade-offs between resource utilization and model accuracy. Furthermore, while reducing the size of the accumulator invariably degrades model accuracy, using A2Q yields higher performing models than existing baselines.

It is important to highlight that our results have implications outside of the accelerators generated by FINN. Constraining the accumulator bit width has been shown to increase inference performance on general-purpose platforms [9, 32, 45] and reduce the compute overhead of homomorphic encryption arithmetic [28, 40]. Furthermore, in training QNNs for low-precision accumulation, A2Q also inherently promotes unstructured weight sparsity. This exposes opportunities that can be exploited by both programmable hardware [7, 33] as well as general-purpose processors [12, 13] as the most common use of sparsity in machine learning workloads is to accelerate inference by reducing compute and memory requirements [17].

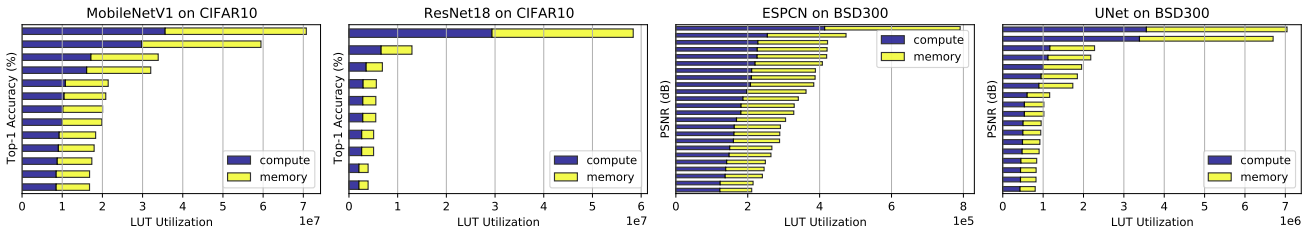


Figure 7: We break down LUT utilization for each of our Pareto optimal models from Fig. 6.

Limitations. The flexibility of FPGAs is a double-edged sword. The bit-level control allows for the precisions of weights, activations, and now accumulators to be individually tuned for each layer in a QNN; however, this vast design space introduces a complex optimization problem. Our study only considers uniform-precision models, but mixed-precision methods have shown promise [11, 43]. State-of-the-art neural architecture search algorithms may be able to navigate this large design space more efficiently.

Finally, we observe that round-to-zero performs poorly in post-training quantization (PTQ) scenarios. Since A2Q relies on round-to-zero to prevent rounding errors from violating our constraints, we observe poor results for A2Q in this scenario. We conjecture that adaptive rounding techniques [30] could alleviate the issue.

7. Related Work

Another approach to training QNNs to use lower precision accumulators is to mitigate the impact of overflow on model accuracy. Xie *et al.* [45] and Li *et al.* [25] sought use adaptive scaling factors to reduce the expected overflow rate as empirically estimated over the training data. Their formulation is unable to guarantee overflow avoidance as it is data dependent. Ni *et al.* [32] proposed training QNNs to be robust to overflow using a cyclic activation function based on modulo arithmetic. They report training instability when the overflow rate is too large, which is common when using extremely low-precision accumulators (please see Appendix A for more details). Furthermore, both approaches model overflow using only the result of the dot product without accounting for intermediate partial sums. Accounting for these partial sums is not easily supported by off-the-shelf deep learning frameworks nor easily generalized across target platforms. In our work, we train QNNs to completely avoid overflow rather than simply reducing its impact on model accuracy.

Most similar to our work is that of [9], which proposed an iterative layer-wise optimization strategy to select mixed-precision bit widths to avoid overflow using computationally expensive heuristics. Their derived bounds on accumulator bit width do not guarantee overflow avoidance for all edge cases and assume only signed bit widths for all data types. Our proposed quantization method adds negligible training overhead and constrains QNNs to guarantee overflow avoidance while accounting for both signed and unsigned input data types.

Prior research has also sought to leverage weight normalization for quantization, but as a means of regularizing long-tail weight distributions during QAT [5, 26]. Cai *et al.* [5] replace the standard ℓ_2-norm with an ℓ_∞-norm and derive a projection operator to map real values into the quantized domain. Li *et al.* [26] normalize the weights to have zero mean and unit variance and observe increased stability. In our work, we replace the ℓ_2-norm with an ℓ_1-norm to use the weight normalization parameterization as a means of constraining learned weights during training to use a user-defined accumulator bit width during inference.

Tangential to our work, [44] and [37] study the impact of reducing the precision of floating-point accumulators for the purpose of accelerating training. Such methods do not directly translate to integer quantization and fixed-point arithmetic, which is the focus of this work.

8. Conclusion

We present accumulator-aware quantization (A2Q), a novel quantization method designed to train QNNs for low-precision accumulation during inference. Unlike previous work, which has sought to either reduce the risk of overflow or mitigate its impact on model accuracy, A2Q guarantees overflow avoidance and exposes the accumulator bit width as an independent variable to be specified. To do so, A2Q constrains the ℓ_1-norm of weights according to accumulator bounds that we derive, inherently promoting unstructured weight sparsity. As the first principled approach to avoiding overflow, we provide theoretical motivation and derive comprehensive bounds on the accumulator bit width with finer granularity than existing literature. We explore the use of low-precision accumulators as a means of improving the design efficiency of FPGA-based QNN inference accelerators. Our experiments show that using our algorithm to train QNNs to use low-precision accumulators improves the trade-offs between resource utilization and model accuracy when compared to existing baselines.

Acknowledgements

We would like to thank Gabor Sines, Michaela Blott, Yaman Umuroglu, Nicholas Fraser, Thomas Preusser, Mehdi Saeedi, Ihab Amer, Alex Cann, Arun coimbatore Ramachandran, Chandra Kumar Ramasamy, Prakash Raghavendra, and the rest of the AMD RTG Software Technology, Edge Inference, and AECG teams for insightful discussions and infrastructure support.

References

[1] AMD-Xilinx. FINN: Dataflow compiler for QNN inference on FPGAs. https://github.com/Xilinx/finn, 2023. Accessed: 2023-01-19. 1, 5, 10

[2] AMD-Xilinx. Vivado design suite user guide: Notes for higher performance FPGA design. https://docs.xilinx.com/r/en-US/ug897-vivado-sysgen-user/Use-Saturation-Arithmetic-and-Rounding-Only-When-Necessary, 2023. Accessed: 2023-01-19. 2

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2, 4

[4] Michaela Blott, Thomas B Preusser, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. FINN-R: an end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–23, 2018. 5, 7, 10, 11

[5] Wen-Pu Cai and Wu-Jun Li. Weight normalization based quantization for deep neural network compression. *arXiv preprint arXiv:1907.00593*, 2019. 8

[6] Shih-Kang Chao, Zhanyu Wang, Yue Xing, and Guang Cheng. Directional pruning of deep neural networks. *Advances in Neural Information Processing Systems*, 33:13986–13998, 2020. 6

[7] Ian Colbert, Jake Daly, Ken Kreutz-Delgado, and Srinjoy Das. A competitive edge: Can FPGAs beat GPUs at DCNN inference acceleration in resource-limited edge computing applications? *arXiv preprint arXiv:2102.00294*, 2021. 7

[8] Ian Colbert, Kenneth Kreutz-Delgado, and Srinjoy Das. An energy-efficient edge computing paradigm for convolution-based image upsampling. *IEEE Access*, 9:147967–147984, 2021. 10

[9] Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocessors and Microsystems*, 72:102872, 2020. 1, 5, 7, 8, 9

[10] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 2, 9

[11] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: Hessian-aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019. 8

[12] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14629–14638, 2020. 7

[13] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse GPU kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020. 2, 7, 9

[14] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021. 1, 2, 4, 10

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5, 10

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 10

[17] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021. 7

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5, 10

[19] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017. 1, 10

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018. 1, 2

[21] Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *Proceedings of Machine Learning and Systems*, 2:112–128, 2020. 2, 4

[22] David R Kaeli, Perhaad Mistry, Dana Schaa, and Dong Ping Zhang. *Heterogeneous computing with OpenCL 2.0*. Morgan Kaufmann, 2015. 2, 9

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 10

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5, 10

[25] Haokun Li, Jing Liu, Liancheng Jia, Yun Liang, Yaowei Wang, and Mingkui Tan. Downscaling and overflow-aware model compression for efficient vision processors. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 145–150. IEEE, 2022. 1, 5, 8, 9

[26] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2020. 8

[27] Dominik Marek Loroch, Franz-Josef Pfreundt, Norbert Wehn, and Janis Keuper. Tensorquant: A simulation toolbox for deep neural network quantization. In *Proceedings of the Machine Learning on HPC Environments*, pages 1–8. 2017. 4

[28] Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 5, 7

[29] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001. 5, 10

[30] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020. 8

[31] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019. 2

[32] Renkun Ni, Hong-min Chu, Oscar Castañeda Fernández, Ping-yeh Chiang, Christoph Studer, and Tom Goldstein. Wrapnet: Neural net inference with ultra-low-precision arithmetic. In *International Conference on Learning Representations ICLR 2021*. OpenReview, 2021. 1, 2, 5, 7, 8, 9

[33] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 5–14, 2017. 7

[34] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016. 10

[35] Alessandro Pappalardo. Xilinx/brevitas, 2021. 1, 5

[36] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 5, 10

[37] Charbel Sakr, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Ankur Agrawal, Naresh Shanbhag, and Kailash Gopalakrishnan. Accumulation bit-width scaling for ultra-low precision training of deep networks. *arXiv preprint arXiv:1901.06588*, 2019. 1, 8

[38] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016. 4

[39] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 5, 10

[40] Andrei Stoian, Jordan Frery, Roman Bredehoft, Luis Montero, Celia Kherfallah, and Benoit Chevallier-Mames. Deep neural networks for encrypted inference with TFHE. *arXiv preprint arXiv:2302.10906*, 2023. 1, 5, 7

[41] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 65–74. ACM, 2017. 5, 6, 10, 11

[42] Yaman Umuroglu and Magnus Jahre. Streamlined deployment for quantized neural networks. *arXiv preprint arXiv:1709.04060*, 2017. 7, 10

[43] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 8

[44] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018. 8

[45] Hongwei Xie, Yafei Song, Ling Cai, and Mingyang Li. Overflow aware quantization: Accelerating neural network inference by low-bit multiply-accumulate operations. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 868–875, 2021. 1, 5, 7, 8, 9

[46] Chen Yang, Zhenghong Yang, Abdul Mateen Khattak, Liu Yang, Wenxin Zhang, Wanlin Gao, and Minjuan Wang. Structured pruning of convolutional neural networks via l1 regularization. *IEEE Access*, 7:106385–106394, 2019. 6

[47] Xinyu Zhang, Ian Colbert, and Srinjoy Das. Learning low-precision structured subnetworks using joint layerwise channel pruning and uniform quantization. *Applied Sciences*, 12(15):7829, 2022. 1, 10