A2Q: Accumulator-Aware Quantization with Guaranteed Overflow Avoidance Supplementary Material

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

A. Motivating Example Details

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 store the result of the dot product, but also all intermediate partial sums. 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 [17]. To demonstrate the impact of overflow, we consider a 1-layer linear quantized neural network (QNN) trained to classify binary MNIST [5] images using 8-bit weights. We flatten each gray-scale 28x28 image so that the inputs to the model are 784-dimensional vectors of 1-bit unsigned integers. Keeping with our notation used throughout our paper, this translates to N=1, M=8, and K=784. We train this 8-bit linear classifier using the baseline quantization-aware training (QAT) algorithm and observe a 91.5% top-1 test accuracy when using a 32-bit accumulator.

Using our accumulator bit width data type bound, we calculate the lower bound on P to be 19 bits. Figure 2 shows the rate of overflows per dot product grows exponentially as we reduce the accumulator bit width below this bound. Our evaluation on the impact of overflow consists of two measures: (1) the mean absolute error on logits as measured between P-bit and 32-bit accumulator results; and (2) the top-1 classification accuracy of the P-bit accumulator result. We observe that the increased overflow rate introduces numerical errors that proportionally increase the mean absolute error on the logits, decreasing classification accuracy.

We first evaluate the default wraparound two's complement arithmetic (**black stars**), which is known to introduce errors that degrade model accuracy [4, 15, 17, 23]. Next, we evaluate naïvely saturating values as they are accumulated (**blue triangles**), which is the industry standard for avoiding overflow. While previous work has shown that overflow ultimately degrades model accuracy due to wraparound two's complement arithmetic [4, 17], they do not benchmark against this standard clipping solution, likely because it is

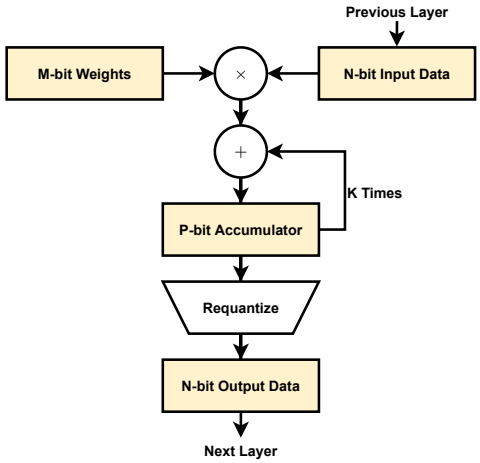


Figure 1: A simplified illustration of fixed-point arithmetic in neural network inference. The accumulator bit width (P) needs to be wide enough to fit the dot product between the M-bit weight vector and the N-bit input vector, which are assumed to both be K-dimensional.

expensive to model with off-the-shelf deep learning frameworks. We show that such clipping can alleviate the accuracy degradation caused by wraparound arithmetic, but still introduce harmful errors. Finally, we benchmark our accumulator-aware quantization (A2Q) method (**green circles**) and re-train the 8-bit linear classifier from scratch using the target accumulator bit width P and the same random seed. We show that using A2Q to avoid overflow significantly improves model accuracy over both wraparound arithmetic and clipping when using extremely low-precision accumulators. Furthermore, we find that, in our experiments, the overflow rate very quickly grows past 10%, which is the point at which Wrapnet reports training instability [17]. Even in those settings, A2Q maintains accuracy with respect to the floating-point counterparts, which further motivates the need to completely avoid overflow rather than simply reduce its impact on model accuracy.

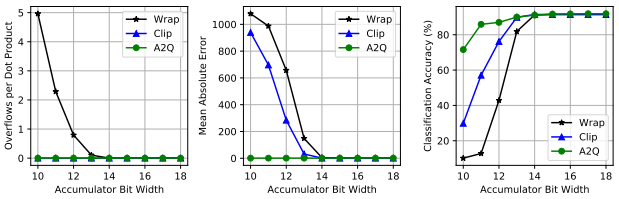


Figure 2: We evaluate the impact of overflow as we reduce the accumulator bit width using a 1-layer QNN trained to classify binary MNIST [5] images using 8-bit weights. We show that using A2Q (**green dots**) to avoid overflow significantly improves model accuracy over both wraparound arithmetic (**black stars**) and clipping (**blue triangles**) when using extremely low-precision accumulators.

A.1. Impact of Breaking Associativity

In applying clipping, the final result of the dot product is made dependent on the order of additions, thus breaking associativity. This can introduce non-deterministic errors when modern processors use optimizations that improve hardware utilization by re-ordering operations (*e.g.*, out-of-order execution [12] or cache swizzling [6]). In Fig. 3, we show how randomly re-ordering the additions in the dot product affects the mean absolute error on the logits (left) and classification accuracy (right). We compare modeling overflow at the outer-most loop using only the dot product result (**red dashed line**) against modeling overflow at the inner-most loop, which accounts for the intermediate partial sums (**blue histogram**). We also provide the baseline classification accuracy as reference (**black dashed line**).

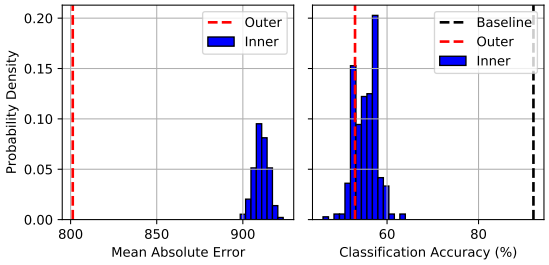


Figure 3: We visualize the impact of re-ordering additions when using saturation logic on the accumulator.

B. Experiment Details and Hyperparameters

Below, we separately detail our image classification and single-image super resolution benchmarks. For all models, we fix the input and output layers to 8-bit weights and activations for all configurations, as is common practice [7, 11, 24]. We also weight our regularization penalty by a constant scalar λ=1e-3 where $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda\mathcal{L}_{\text{reg}}$.

B.1. Image Classification Benchmarks

We train MobileNetV1 [10] and ResNet18 [8] to classify images using the CIFAR10 dataset [14]. We closely follow the network architectures originally proposed by the respective authors, but introduce minor variations that yield more amenable intermediate representations given the reduced image size of CIFAR10 images [14]. We initialize all models from floating-point counterparts pre-trained to convergence on CIFAR10 and evaluate task performance using the observed top-1 test accuracy.

MobileNetV1. We use a stride of 2 for both the first convolution layer and the final average pooling layer. This reduces the degree of downscaling to be more amenable to training over smaller images. All other layer configurations remain the same as proposed in [10]. We use the stochastic gradient descent (SGD) optimizer to fine-tune all models for 100 epochs in batches of 64 images using a weight decay of 1e-5. We use an initial learning rate of 1e-3 that is reduced by a factor of 0.9 every epoch.

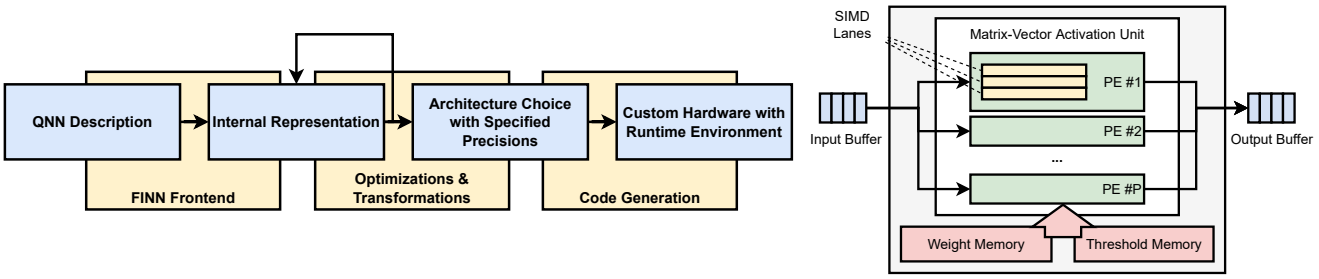
ResNet18. We alter the first convolution layer to use a stride and padding of 1 with a kernel size of 3. We remove the preceding max pool layer to reduce the amount of downscaling throughout the network. We also use a convolution shortcut [9] rather than the standard identity as it empirically proved to yield superior results in our experiments. All other layer configurations remain the same as proposed in [8]. We use the SGD optimizer to fine-tune all models for 100 epochs in batches of 256 using a weight decay of 1e-5. We use an initial learning rate of 1e-3 that is reduced by a factor of 0.1 every 30 epochs.

B.2. Single-Image Super Resolution Benchmarks

We train ESPCN [20] and UNet [19] to upscale single images by a factor of 3x using the BSD300 dataset [16]. Again, we closely follow the network architectures originally proposed by the respective authors, but introduce minor variations that yield more hardware-friendly network architectures. We randomly initialize all models and train them from scratch. We empirically evaluate task performance using the peak signal-to-noise ratio (PSNR) observed over the test dataset.

ESPCN. We replace the sub-pixel convolution with a nearest neighbor resize convolution (NNRC), which has been shown to reduce checkerboard artifacts during training [18] and can be efficiently executed during inference [3]. All other layer configurations remain the same as proposed in [20]. We use the Adam optimizer [13] to fine-tune all models for 100 epochs in batches of 16 images using a weight decay of 1e-4. We use an initial learning rate of 1e-4 that is reduced by a factor of 0.98 every epoch.

UNet. We use only 3 encoders and decoders to create a smaller architecture than originally proposed by [19]. We replace transposed convolutions with NNRCs, which are



(a) FINN Framework Overview

(b) MVAU

Figure 4: We adapt images from [2, 21] to provide: (a) an overview of the FINN framework; and (b) an abstraction of the matrix-vector-activation unit (MVAU), which is one of the primary building blocks used by the FINN compiler.

known to be functionally equivalent during inference [3], but have more favorable behavior during training [18]. We replace all concatenations with additions and reduce the input channels accordingly. We use the Adam optimizer to fine-tune all models for 200 epochs in batches of 16 using a weight decay of 1e-4. We use an initial learning rate of 1e-3 that is reduced by a factor of 0.3 every 50 epochs.

C. Generating Accelerators with FINN

FINN [2, 21] is an open-source framework designed to generate custom QNN inference accelerators for AMD-Xilinx FPGAs. For a given QNN, the FINN framework, depicted in Fig. 4a, generates a specialized accelerator using spatial streaming dataflow architectures that are individually customized for the network topology and the data types used. At the core of FINN is its compiler, which empowers flexible hardware-software (HW-SW) co-design by allowing a user to have per-layer control over the generated accelerator. Weight and activation precisions can be individually specified for each layer in a QNN, and each layer is instantiated as its own dedicated compute unit (CU).

As an example of a layer instantiated as its own CU, we provide a simplified abstraction of the matrix-vector-activation unit (MVAU) in Fig. 4b. The MVAU is one of the primary building blocks used for linear and convolutional layers [2]. Each CU consists of processing elements (PEs), which parallelize work along the output dimension, and single-instruction multiple-data (SIMD) lanes, which parallelize work along the input dimension. All quantized monotonic activation functions in the network are implemented as threshold comparisons that map high-precision accumulated results from the preceding layer into low-precision output values. During compilation, batch normalization, biases and even scaling factors are absorbed into this threshold logic via mathematical manipulation [22]. The input and output data for the generated accelerators are streamed into and out of the chip using AXI-Stream protocols while on-chip data streams are used to interconnect these CUs to propagate intermediate activations through the layers of the network. During inference, all network parameters are stored on-chip to avoid external memory bottlenecks. We refer the reader to [1, 2, 21] for more information.

References

[1] AMD-Xilinx. FINN: Dataflow compiler for QNN inference on FPGAs. https://github.com/Xilinx/finn, 2023. Accessed: 2023-01-19. 3

[2] 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. 3

[3] 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. 2, 3

[4] 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] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 1, 2

[6] 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] 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. 2

[8] 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. 2

[9] 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. 2

[10] 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. 2

[11] 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. 2

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

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[15] 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

[16] 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. 2

[17] 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

[18] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016. 2, 3

[19] 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. 2

[20] 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. 2

[21] 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. 3

[22] Yaman Umuroglu and Magnus Jahre. Streamlined deployment for quantized neural networks. *arXiv preprint arXiv:1709.04060*, 2017. 3

[23] 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

[24] 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. 2