Dynamic Quantization Range Control for Analog-in-Memory Neural Networks Acceleration

NATHAN LAUBEUF[†‡], JONAS DOEVENSPECK[†], IOANNIS A. PAPISTAS[†], MICHELE CASELLI[†], STEFAN COSEMANS[†], PETER VRANCX[†], DEBJYOTI BHATTACHARJEE[†], ARINDAM MALLIK[†], PETER DEBACKER[†], DIEDERIK VERKEST[†], FRANCKY CATTHOOR[†‡] and RUDY LAUWEREINS[†‡],

† Interuniversity Microelectronics Center (IMEC), Belgium and

‡ ESAT Laboratory, Katholieke Universiteit (K.U.) Leuven, Belgium

Analog in Memory Computing (AiMC) based neural network acceleration is a promising solution to increase the energy efficiency of deep neural networks deployment. However, the quantization requirements of these analog systems are not compatible with state of the art neural network quantization techniques. Indeed, while the quantization of the weights and activations is considered by modern deep neural network quantization techniques, AiMC accelerators also impose the quantization of each Matrix Vector Multiplication (MVM) result. In most demonstrated AiMC implementations, the quantization range of MVM results is considered a fixed parameter of the accelerator. This work demonstrates that dynamic control over this quantization range is possible but also desirable for analog neural networks acceleration. An AiMC compatible quantization flow coupled with an hardware aware quantization range driving technique is introduced to fully exploit these dynamic ranges. Using CIFAR-10 and ImageNet as benchmarks, the proposed solution results in networks that are both more accurate and more robust to the inherent vulnerability of analog circuits than fixed quantization range based approaches.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Hardware** → **Analog and mixed-signal circuits**.

Additional Key Words and Phrases: Neural networks, Quantization, In-Memory-Computing

ACM Reference Format:

Nathan Laubeuf[†‡], Jonas Doevenspeck[†], Ioannis A. Papistas[†], Michele Caselli[†], Stefan Cosemans[†], Peter Vrancx[†], Debjyoti Bhattacharjee[†], Arindam Mallik[†], Peter Debacker[†], Diederik Verkest[†], Francky Catthoor[†‡] and Rudy Lauwereins[†‡]. 2021. Dynamic Quantization Range Control for Analog-in-Memory Neural Networks Acceleration. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1 (October 2021), 21 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The acceleration of MVM operations provided by graphical processing units (GPU) has been instrumental to the success of deep neural networks [28, 33]. By providing high parallelism and large memory buffers to store the weights and activations of deep neural networks, GPUs remain to this day the main platform to train and deploy deep neural

This research received funding from the Flemish Government (AI Research Program) and KU Leuven grant C14/18/100.

Authors' address: Nathan Laubeuf[†‡], Jonas Doevenspeck[†], Ioannis A. Papistas[†], Michele Caselli[†], Stefan Cosemans[†], Peter Vrancx[†], Debjyoti Bhattacharjee[†], Arindam Mallik[†], Peter Debacker[†], Diederik Verkest[†], Francky Catthoor[†‡]; Rudy Lauwereins[†‡], nathan.laubeuf@imec.be,

† Interuniversity Microelectronics Center (IMEC), Kapeldreef 75, Leuven, Belgium, 3001 and

‡ ESAT Laboratory, Katholieke Universiteit (K.U.) Leuven, Kasteelpark Arenberg 10, Leuven, Belgium, 3001.

Manuscript submitted to ACM

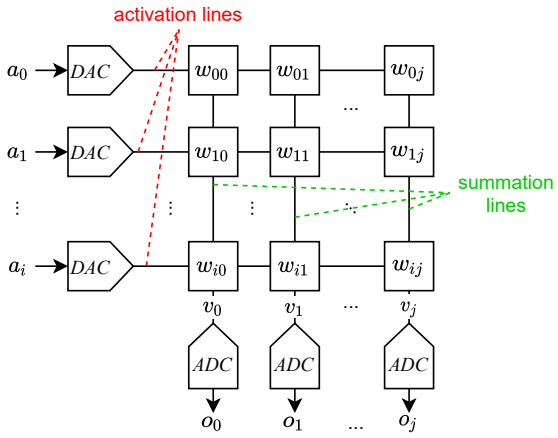

Fig. 1. AiMC based matrix vector multiplier. The input vector a is encoded on activation lines and multiplied by a weights matrix W. The result vector o is read out by analog-to-digital converters (ADCs) on summation lines.

networks. As most GPUs are designed for 32 bit floating point arithmetic, it is common for deep neural networks to be executed with 32 bit weights and activations.

However, recent advancements in quantizing deep neural networks for inference have shown that networks using smaller bit width representation for their weights and activations [19] can achieve accuracies on par with full-precision networks. Exploiting these quantization solutions, digital neural network accelerators, such as Google TPUs [18], Microsoft BrainWave [9] or Nvidia's Volta architecture, are able to achieve significant time and energy gains compared to conventional GPUs. Being tied to the progress of CMOS technology, the gains achievable by these digital accelerators are unable to match the growing complexity of recent deep neural networks [3].

To overcome this limitation, emerging non-Von Neumann architectures for neural networks accelerators are being developed. Among them, mixed-signal systems based on Analog in Memory Computing (AiMC) are a radical example to push DNNs' energy efficiency [31]. The working principle of AiMC based MVM acceleration is illustrated in Fig. 1. In AiMC, an input integer vector a is encoded by digital-to-analog-converters (DACs) on a set of activation lines. These lines are connected to memories containing the weights W of the MVM operation. These weights determine the contribution of each activation to rows of summation lines. The MVM result is digitized on these summation lines using analog-to-digital converters (ADCs). This approach allows a drastic reduction of the data movement and energy consumption compared to more conventional architectures, with further benefits in terms of energy due to the analog processing.

In typical Convolutional Neural Networks (CNNs), convolutional layers are composed of three successively applied operations: *convolution, batch normalization* and *activation*. To quantize these layers, quantization functions are applied to the weights and the input activations of the convolution operation. Using this method, convolution operations are processed using integer arithmetic while floating point precision is maintained for the rest of the operations. Though this quantization flow is valid for digital accelerators, able to mix floating point and mixed precision integer arithmetic, its assumptions are incompatible with AiMC execution.

Due to energy and area constrains, most AiMCs do not perform true integer based MVMs. Given an MVM array containing i rows, using b_a bit DACs, and b_w bit weights, an ADC capable of providing $2^{b_a-1} \times 2^{b_w-1} \times i$ output codes would be required to represent all possible MVM output values. As an example, given an array of ternary weights with

8 bit DACs deployed over 256 rows, 15 bits are required by the output ADC to match all possible values. AiMC solutions using such high resolution ADCs are not as energy efficient as equivalent digital designs [23]. As such, high precision ADCs are not used in AiMC prototypes, implicitly requantizing and clipping all MVM results instead. By leaving convolution outputs unconstrained, this quantization effect is overlooked by the previous state-of-the-art quantization flows, reducing the accuracy achievable by these quantized networks on AiMC.

Carefully determining the quantization ranges of commonly used linear quantization functions, is crucial to maintain the accuracy of quantized neural networks [19]. ADC resolution constrains limit the amount of control given over this quantization range in AiMCs based systems. *Ad hoc* methods to determine ADC quantization ranges have proven efficient on relatively small networks [7, 17, 24, 36]. However, the unique network-wide MVM output quantization range assumed by these methods is incompatible with the tensor wise range determination techniques used by most quantization methods and can result in significant accuracy degradation for deeper networks [36].

To address these discrepancies, this paper introduces a deep neural network quantization flow which is fully compatible with AiMC execution. AiMC control techniques are exposed to achieve flexible MVM output quantization ranges. This flow is coupled with a software solution to model the limits of ADC resolution, adapt the quantized networks to these limits and generate appropriate AiMC driving patterns to control these dynamic quantization ranges. In order, the contributions of this paper are as follows:

- A fully quantized AiMC aware neural network quantization flow is demonstrated.
- AiMC driving techniques, enabling flexible quantization range control in AiMC are introduced.
- On a simulated AiMC, the combination of the described methods is shown to result in quantized versions of Resnet-20 and Resnet-18, that outperform empirical methods in both accuracy and resilience to inherent AiMC variabilities.

The rest of this paper is organized as follows. Section 2 provides information on the different quantization flows in the literature and quantization range determination in AiMC. Section 3 introduces the methods used by this paper for the quantization dataflow, hardware-based quantization range scaling and the associated software model used to train AiMC compatible neural networks. Section 4 experimentally validates these methods on relevant image classification benchmarks. Finally, conclusions are provided in Section 5.

2 BACKGROUND AND RELATED WORK

Though deploying quantized neural networks on AiMC holds the potential for more energy efficient networks, current neural network quantization solutions are not adapted for the design constrains of AiMCs. Additional considerations regarding quantization flow and achievable quantization resolutions need to be taken into account. This section presents background information and related literature regarding, quantization flow considerations and quantization range determination and control.

2.1 Quantization flow

As most digital neural network accelerators support mixed precision data-types, most quantization techniques make active use of both integer and floating point operations. As mentioned in Section 1, the convolution layers of a CNN are composed of three successive operations: *convolution*, *batch normalization* and *activation*. Most quantization methods consider discretizing the weights and input activations of the convolution in order to accelerate the convolution

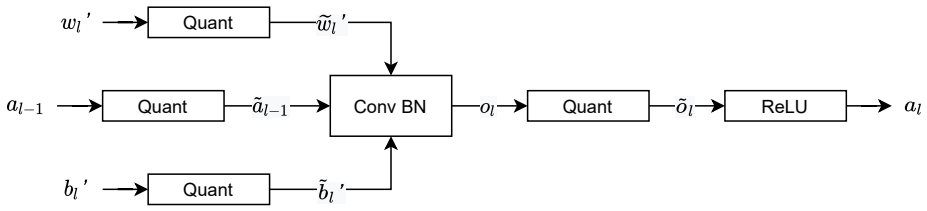

Fig. 2. Batch norm folded convolution quantization. The Batch normalization parameters are used to rescale the weights and shift the biases of the original convolution. The resulting Conv-Batch Norm operation can then be quantized.

operation using integer arithmetic, maintaining floating point precision for the batch normalization and activation function [8, 11, 12].

To limit reliance on floating point logic, alternative quantization flows have considered quantizing the batch normalization operation. In [4] and [22], the multiplicative and additive factors of batch normalization operations are explicitly quantized. In [15], batch normalization multiplications are performed using a bit shift based mechanism relying on powers of 2 approximations. The bit widths of the convolution results in these quantization flows, however, remain unconstrained and possibly operate at precisions higher than those usually supported by AiMC systems.

In [19] and [16], the batch normalization operation is merged with the convolution operation during training by rescaling the weights and shifting the biases of the convolution before their quantization (Fig. 2). This quantization flow, referred to as batch norm folding, results in a merged convolution batch normalization operation that, if the DACs and ADCs of the AiMC share the same precision, can be mapped on AiMC consistently. However, due to the binary nature of the most of the memory elements used for AiMC accelerators, weight precisions higher than ternary are rarely supported (Table 1). To the best of authors knowledge, the batch norm folding in ternary neural networks has yet to be demonstrated.

2.2 Range determination

In most quantization methods, uniform quantization functions are inserted in the network computation flow to quantize the weights and activations. Using a uniform quantization function Q, a floating point tensor $x \in \mathbb{R}^n$ is decomposed as the product of a discrete integer tensor $x_i \in \mathbb{N}^n$ and a floating point step size $x_s \in \mathbb{R}$, such that $Q(x) = x_i \cdot x_s = \tilde{x}$. The quantization step size x_s is expressed as the ratio of the maximum absolute floating point value in the quantization range s_f and the maximum absolute value representable by the chosen integer representation s_i, such that: $x_s = \frac{s_f}{s_i}$. The integer tensor x_i is calculated by dividing x by the scaling factor x_s, clipping this value to the quantization's representable integer range $[l_{min}, l_{max}]$, and rounding the obtained value up, down or to the nearest integer. Rounding to the nearest integer, x_i can be expressed as:

$$x_i = \text{clamp}(\left\lfloor \frac{x}{x_s} \right\rceil, l_{min}, l_{max}) \tag{1}$$

The quantization ranges s_f of the inserted quantization functions need to be properly determined to maintain the accuracy of quantized neural networks. These ranges can be determined based on the statistics of the tensors to quantize. The approach proposed in [19] uses the minimum and maximum values of the weights and the moving average of the minimum and maximum of the activations. [22] uses a multiple of the standard deviation of the weights and the 99.99^{th} or 99.9^{th} percentile of the activations. The quantization ranges can also be learned as parameters of the network. In [8]

Table 1. AiMC macro bit widths, input vector length and resulting quantization ranges.

	Precision			Vector length	Quantization range
	DAC	Memory	ADC		
[36]	ternary	1 bit	11 levels	256	$[-60, 60]$
[17]	ternary	1 bit	11 levels	256	$[-64, 64]$
[24]	7 bit	ternary	6 bit	1024	$[-6400, 6400]$
[7]	7 bit	1 bit	7 bit	64	$[-2048, 2048]$
[32]	2 bit	5 bit	5 bit	64	NA

and [5], they are learned as parameters of quantized activation functions, while in [16] and [11] they are learned for both weights and activations quantization.

As quantization ranges are determined on a tensor [16] or filter basis [19], different ranges are used throughout the network. These range determination methods differ significantly from the methods adopted by most reported AiMC implementations which tend to adopt a single network wide quantization range, determined empirically on a small set of targeted neural networks. In [36], the distributions of network-wide accumulation results are observed to select a quantization range able to encompass most of the MVM results dynamic range. The number of quantization levels dedicated to that range is then determined by selecting the minimal number of levels resulting in no accuracy degradation on the selected tasks. [17] adopts the opposite approach of first selecting the largest quantization step size resulting in no accuracy degradation and discarding the levels outside of the observed dynamic range.

In both cases, the post-hoc methods used to determine the bit-width and step size of the ADC are calibrated on distributions gathered from relatively shallow networks (maximum 9 layers). By determining the accelerator design parameters(e.g. ADC bit-width and step size), from such networks, the resulting accelerators offer no guaranties regarding the accuracy of deeper networks. The type of degradation that such method can cause on deeper networks is illustrated in [36] where an originally more accurate ternary Resnet-14 becomes less accurate by 1.73% on CIFAR-10 than the shallower VGG network used to determine the considered accelerator's ADCs step-size and bit-width. This example illustrates that designing accelerators around specific networks, and thus not enabling control over the quantization range of MVM operations, can cause significant accuracy degradation on the application of interest.

Other notable AiMC designs are not limited by such constrains, and still allow flexibility regarding the quantization range of the ADC. As an example, [34] uses a current-sensing approach based on an RRAM array whose load resistance consists of a programmable RRAM cell used to dynamically rescale the summation line current range to a fixed voltage range. Similarly, [32] is a charge-discharge based SRAM AiMC design that includes a configurable replica SRAM column used to provide a voltage reference to dynamically change the quantization range of the ADC.

Although these methods can enable flexible quantization ranges, specific implementations often parametrize this range as a single value, empirically determined network wide [24]. Table 1 lists examples, taken from the literature, of bit-widths and vector lengths supported by AiMC implementations along with the quantization ranges reported for these implementations. To our knowledge, paired quantization and control patterns to enable algorithm driven, tensor-wise control over these quantization ranges are still missing from the literature. Our experiments show that enabling such control is advantageous for the deployment of realistic networks on large, energy efficient, AiMC arrays.

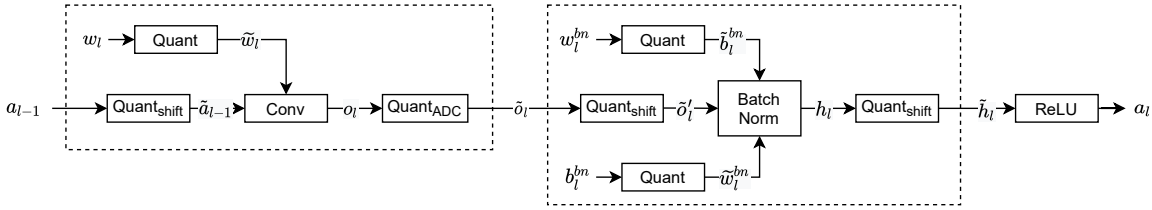

Fig. 3. Mixed signal convolution quantization. The convolution and batch normalization are quantized separately with different precisions.

3 METHODS

To reconcile the assumptions of modern quantization techniques and the constrains of AiMC, novel contributions are introduced by this work. In subsection 3.1, a new CNNs quantization flow, adapted for execution on AiMC accelerators, is introduced. In subsection 3.2, methods to support dynamic output quantization ranges in AiMC accelerators are discussed. Subsection 3.3 proposes a method to appropriately control these quantization ranges, with the quantization step sizes learned using the previous quantization flow.

3.1 AiMC Aware CNN Quantization flow

Batch norm folding is a widely adopted technique, used by default by two of the most popular neural-networks-based machine learning frameworks [1, 26] to enables fully quantized CNN execution. However, this method causes large precision degradation below 4 bit weight precision (as shown in subsection 4.2). This paper proposes an alternative CNN quantization flow (Fig. 3), adapted for AiMC execution. Contrary to batch norm folding, all operations in the introduced flow are quantized individually considering both the operands and results of each operation. These quantization functions ensure that the bit width of the represented data is supported by the considered platforms at each step of the network's execution. As opposed to the operands-only quantization schemes reviewed in subsection 2.1, the different precisions imposed by the DACs, ADCs, and memory elements in AiMC (Table 1) are accounted for in this scheme, constraining analog convolution outputs to precisions supported by the components on the considered platform.

Since each operation can be tied individually to specific execution methods by using this quantization flow, flexible deployment configurations can be considered. By mapping convolution operations to AiMC and batch normalizations to digital support logic, higher precision batch normalization can be applied with this method. However, to support such mixed signal configurations, the requantization of the results to the bit-width of the next operations is mandatory. An approach similar to [16], is adopted to avoid the expensive floating point conversions and the requantization operations. By constraining the quantization step sizes x_s of quantized tensors to powers of 2, the conversion steps between operation can be executed using bit-shift and clipping operations, minimizing their computation overhead.

A similar bit-shift mechanism is used to quantize the residual connections of CNNs. Residual connections are a prominent feature of state-of-the-art CNN architecture [13, 27, 30]. With residual connections, the result of convolutions layers occurring at different levels of the CNN are added to one another before being passed to the rest of the execution. For quantized residual networks, the addition of quantized tensors using integer arithmetic can be non trivial if its operands use different quantization step sizes. The sum of two quantized tensors is the result of a sum of products:

$$\tilde{x} + \tilde{y} = x_i \cdot x_s + y_i \cdot y_s \tag{2}$$

If $x_s \neq y_s$, these step sizes cannot be factored out of the addition. As such, adding the integer component of quantized tensors using different step sizes results in an incorrect result. Therefore, a quantized addition using bit shift based rescaling is introduced to solve this issue. Given two quantized operands $\tilde{x} = x_i \cdot x_s$ and $\tilde{y} = y_i \cdot y_s$, with $x_s > y_s$, $\tilde{x}$ is scaled down to the level of $\tilde{y}$ by bit shifting its integer component x_i by the ratio x_s/y_s. Since the resulting quantized tensor $\tilde{x}' = x_i << \lfloor log2(x_s/y_s) \rceil \cdot y_s$ uses the same scale y_s as $\tilde{y}$, it allows for the following approximation:

$$\tilde{x} + \tilde{y} \approx \tilde{x}' + \tilde{y} = (x_i << \lfloor log2(x_s/y_s) \rceil + y_i) \cdot y_s \tag{3}$$

This approximation is used throughout the network to realistically model quantized residual connections.

Compared to the full-integer based quantization flow proposed here, more conventionnal AiMC based system tend to rely on high precision floating point operations for batch normalization and residual addition. To evaluate the expected energy and area impact of AiMC based neural networks inference using the proposed fully quantizated flow compared to the aforementioned systems, an evaluation was performed based on 180nm PDK using Synopsis DC at 250 MHz. Based on synthesized floating point-32 (IEEE 754 single precision float) [35] and int-8 based multiply-add units, the area and energy of the synthesized fp-32 unit are 64.5 and 178.8 times larger, respectively, than their int-8 based equivalent. The number of multiply-add operations required to implement batch normalization is proportional to the number of output channels of each convolution operation. Similarly, the number of addition operations required to implement residual connection in layers for which it is required is also proportional to the number of output channels. This implies that the proposed fully integer based quantization flow offers significant benefits over more conventional implementations relying on floating point logic.

This work concerns itself mainly with the inference of CNNs on large AiMC arrays, that do not require the matrices of single convolution operations to be split across multiple analog arrays. The reason for this choice is that addition in the analog domain is much cheaper than in digital domain. If smaller arrays are used, the energy efficiency of the array itself reduces due to higher ADC overheads and this is further compounded by the overhead of partial sums required compute each layers results. As shown in [6], at equal areas, larger arrays are more energy efficient than their smaller counterparts.

Two network architectures are of interest for this work: Resnet-20 and Resnet-18. For ResNet-20, an array size of 576 x 64 is large enough to accommodate all MVM operations without the need for splitting. As such, the 1024 x 415 AiMC demonstrated by [24] does not require any of the Resnet-20 operation to be split in any way. For Resnet-18 an arrays size of 4608 x 512 is large enough to accommodate all MVM operations without the need for splitting. Though such a large AiMC array is yet to be demonstrated, we believe that such an array size is not inconceivable.

If partial sums are explicitly required, however, the technique presented here can be extended to account for the splitting of the MVM operations. By tiling convolution operations to the size of the considered array and wrapping each MVM operation in quantization operation a proper scaling factor can be determined for the MVM operations groups corresponding to each analog convolution layer.

3.2 Hardware supported scaling

In AiMC, MVM results are encoded as voltages on summation lines (Fig. 1). These voltages are proportional to the MVM results. A unit summation voltage δv_{sum} can thus be defined, such that:

$$v_{sum} = \delta v_{sum} \cdot Wa \tag{4}$$

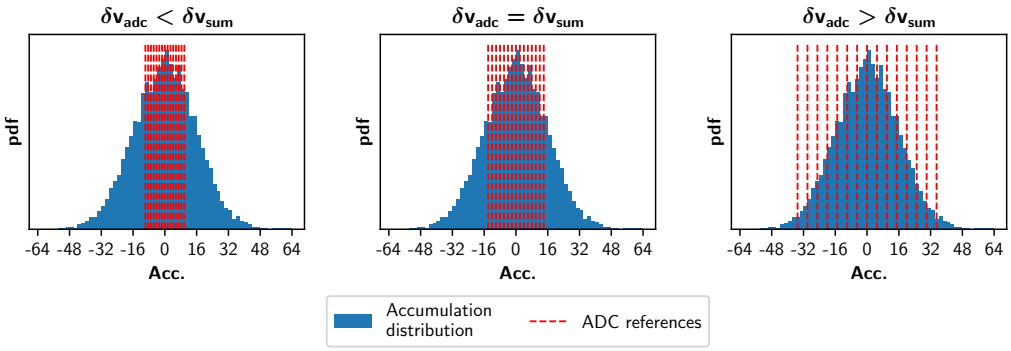

Fig. 4. Accumulation results partition by the ADC as a function of the ADC least significant bit - unit summation voltage relationship.

Analog results v_{sum} are converted to digital values o by an ADC before being passed as operands for the next operation. By noting δv_{adc} the quantization step of the ADC, the complete transfer function of the AiMC based MVM is:

$$o = \text{clamp}(\left\lfloor \frac{\delta v_{sum}}{\delta v_{adc}} \cdot Wa \right\rfloor, ADC_{min}, ADC_{max}) \tag{5}$$

The relationship between the unit summation voltage δv_{sum} and the ADC's quantization step determines the partition of the accumulation results by the ADC's quantization (Fig 5).

- If the ADC's quantization step is smaller than the unit summation voltage ($\delta v_{adc} < \delta v_{sum}$), the ADC's quantization step is unnecessarily small as more than one ADC level is spent per accumulation unit.
- If the ADC's quantization step and the unit summation voltage are equal ($\delta v_{adc} = \delta v_{sum}$), the AiMC behaves like a digital accelerator. However, as outputs are clipped to the range of the ADC, this configuration fails to capture the full accumulation distribution (Fig. 5).
- If the ADC's quantization step is larger than the unit summation voltage, ($\delta v_{adc} > \delta v_{sum}$), a wider quantization range, able to capture the accumulations distribution, is achieved. As the ADC's quantization step covers $\delta v_{adc}/\delta v_{sum}$ accumulation levels, a rescaling factor $\rho = \delta v_{sum}/\delta v_{adc}$ is applied to the accumulation by the ADC's quantization. Controlling either of the ADC's quantization step or the unit summation voltage allows to tune this scaling factor.

Control over the unit summation voltage can be introduced in hardware in multiple ways. As an example, in [7] and [24], Pulse Width Modulation (PWM) DACs are used to encode the activation. Fig. 4 illustrates how activations are converted to voltage pulses in such DACs. The time width of the encoding pulses is proportional to the activations as their product with a unit time parameter t_u. This unit time is generated using a clocking mechanism, with period t_c. By grouping a programmable number n_u of successive clocking events, the unit time is: $t_u = n_u \times t_c$. As such, the unit contribution of each activation on the summation line δv_{sum} is directly proportional to the number of clock events per unit time n_u.

Similar type of control can be enabled over the ADC's quantization step. Time scaling can be implemented by linearly controlling the unit integration time of time conversion based integrating ADCs. ADCs using programmable voltage references, such as [32] can also be used, allowing large flexibility with a linear control over the ADC's LSB, but also non-linear ADC quantization schemes.

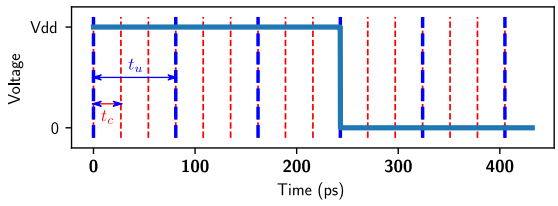

Fig. 5. Pulse width modulation scaling. A voltage pulse width is generated proportional to the input activation, 3 in the example. The time per unit t_u is a multiple of a fixed delay t_c generated by an internal clocking mechanism.

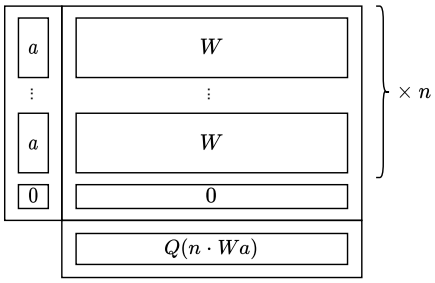

Fig. 6. Scaling unrolling scheme. The weights and activations are unrolled n times in the AiMC array, resulting in a gain factor of n.

Table 2. Numerical applications parameters

V_{dd}	$csbias$	I_{cell}	C_{line}	t_c
0.8 V	0.6 V	0.85 µA	700 fF	27 ps

Hardware based methods to control the scaling factor of AiMC are not limited to charge based implementations. Notably, the current-sensing approach based on an RRAM array introced in [34] uses a load resistance consisting of a programmable RRAM cell, to dynamically re-scale the summation line current range to a fixed voltage range. This observation, along with previously mentioned approaches, testifie that dynamic scaling can be generalized, across a wide range of device types, for both current-sensing and charge based AiMC implementations.

Not all AiMC arrays, however, implement such control schemes. For those AiMC systems, redundant weight mapping strategies are available to emulate control over the unit summation voltage. As an example, considering a convolution layer with i input channels, o output channels, and a $k \times k$ sized kernel, General Matrix Multiply (GeMM) inspired architectures unroll the convolution filters corresponding to each output channel in the columns of the accelerator. In each column, the kernels are unrolled sequentially for each input channel, resulting in $i \times k \times k$ weight vectors, multiplied by their corresponding unrolled activations. By repeating the unrolling procedure n times in each row (Fig. 6), v_{sum} is multiplied by n. Nonetheless, this approach is limited by the total number of times an operation can be unrolled in a given array, and constraints v_{sum} to multiples of its original value, since each operation must be fully unrolled to obtain correct results.

3.3 Scaling control

Subsection 3.2 showed that multiple solutions are available to enable control over the rescaling factor ρ across a wide range of AiMC based systems. However, specific AiMC system implementations restrict the number of possible values of ρ. Linear control of the unit summation voltage δv_{sum} or the ADC's quantization step δv_{adc} using a fixed precision integer n is assumed by this work. All the scaling techniques mentioned in subsection 3.2 are covered under this assumption. The values of ρ, achievable under this assumption, can be expressed as:

$$\rho_a = \begin{cases} \dfrac{n \cdot \alpha_{sum}}{\delta v_{adc}} & \text{for linear } \delta v_{sum} \\ \dfrac{\delta v_{sum}}{n \cdot \alpha_{adc}} & \text{for linear } \delta v_{adc} \end{cases} \tag{6}$$

where n is a fixed precision integer control signal.

The MVM of a quantized matrix $\tilde{W}$ with a quantized vector $\tilde{a}$, using subsection 3.1 quantization flow, can be written as:

$$\tilde{o} = Q(\tilde{W}\tilde{a}) = \text{clamp}(\left\lfloor \frac{w_s \cdot a_s}{o_s} \cdot W_i a_i \right\rceil, l_{min}, l_{max}) \tag{7}$$

In this expression, the quantization step sizes a_s, w_s and o_s are determined by the training process. Hence, the rescaling factor $\rho_d = \frac{w_s \cdot a_s}{o_s}$, is also indirectly learned during training.

In order to enable AiMC based operation-wise scaling, a control signal n needs to be found, for each operation, to approximate ρ_d using the available values of ρ_a. By expressing the control signal n as:

$$n = \begin{cases} \text{clamp}(\left\lfloor \rho_d \cdot \dfrac{\delta v_{adc}}{\alpha_{sum}} \right\rceil, n_{max}, n_{min}) \text{ for linear } \delta v_{sum} \\ \text{clamp}(\left\lfloor \dfrac{1}{\rho_d} \cdot \dfrac{\delta v_{sum}}{\alpha_{adc}} \right\rceil, n_{max}, n_{min}) \text{ for linear } \delta v_{adc} \end{cases} \tag{8}$$

the fixed precision integer n results in an analog scale ρ_a closest to the learned scale ρ_d.

However, ρ_a only approximates the learned factor ρ_d. Therefore, to properly take into account the difference between the analog and digital rescaling factors during training and inference, the output quantization step sizes of the MVM operations are adjusted to match the control signal n as:

$$o'_s = \begin{cases} w_s \cdot a_s \cdot \dfrac{\delta v_{adc}}{n \cdot \alpha_{sum}} & \text{for linear } \delta v_{sum} \\ w_s \cdot a_s \cdot \dfrac{n \cdot \alpha_{adc}}{\delta v_{sum}} & \text{for linear } \delta v_{adc} \end{cases} \tag{9}$$

As a result, the output quantization step sizes are also quantized, and can be expressed as a function of the different quantization step sizes w_s, a_s and o_s learned during training. Using these step sizes, o'_s can be rewritten:

$$o'_s = \begin{cases} \gamma_{sum} / \text{clamp}(\lfloor \gamma_{sum} / o_s \rceil) & \text{for linear } \delta v_{sum} \\ \gamma_{adc} \cdot \text{clamp}(\lfloor o_s / \gamma_{adc} \rceil) & \text{for linear } \delta v_{adc} \end{cases} \tag{10}$$

with

$$\gamma_{sum} = w_s \cdot a_s \cdot \frac{\delta v_{adc}}{\alpha_{sum}} \tag{11}$$

$$\gamma_{sum} = w_s \cdot a_s \cdot \frac{\alpha_{adc}}{\delta v_{sum}} \tag{12}$$

By using straight-through-estimation [5, 14] to differentiate the rounding function, the scale adjustment operation is fully differentiable. As such, backpropagation through this function is possible, maintaining the learnability of the quantization step sizes a_s, w_s and o_s. As the control signal n is a function of these step sizes and fixed hardware parameters, n is indirectly learned using this process. Since the step sizes are also different for individual layers, different control signals n are used throughout the network, which contrasts with previously mentioned network wide determination techniques.

From a system perspective, the method presented here introduces a single integer parameter per layer. As such, this value can be loaded with the weights in a register directly associated with the array. Taking the configuration and networks considered in section 4 as an example, Resnet-20 maps a total of 269824 weights (67.5 KB) on the array for its execution. For this same execution, 20 6-bit encoded scale factors (15B) are required, that is 0.02% of the total memory impact. For Resnet-18 the number of weight goes up to 11157504 (2.8 MB), while only 19 scales are required (15B), or 0.005% of the total memory impact. Thus, in a complete system the overheads of control and storing this value in a layer-wise fashion is expected to be negligible compared to the control and memory overhead of the array's weights.

4 EXPERIMENTS

The methods introduced in Section 3, are validated on two sets of experiments. The quantization flow introduced in subsection 3.1 is then compared to a batch norm folding based quantization flow. The scaling control pattern of subsection 3.3 is finally benchmarked against an empirical scale determination method, under realistic hardware assumptions.

4.1 Target networks and dataset

Two visual object classification datasets are used to evaluate the described methods: CIFAR-10 [20] and ImageNet [29]. CIFAR-10 is composed of 60,000 32x32 pixels colored RGB images, evenly split into 10 different classes, while ImageNet is a larger dataset composed of 1.25M high-resolution images, split across 1000 classes. In order to perform model testing on previously unseen data, both datasets are subdivided into two separate training and testing sets. For CIFAR-10 and ImageNet, the testing sets are composed of 10,000 and 50,000 images, respectively. For model selection and early stopping purposes, similarly sized validation sets are sampled from the remaining images.

Two networks of the Resnet-family [13] are targeted for this evaluation: Resnet-20 on CIFAR-10 and Resnet-18 on ImageNet. For Resnet-20, a floating point baseline is trained from scratch for 200 epochs. Training is conducted with a batch size of 128 samples, using step decay learning rate scheduling with an initial learning rate of 0.1, decaying by a factor 5 at epoch 60, 120 and 160 as used originally in [20]. Nesterov momentum is also applied to the weight update, with a momentum value of 0.9 along with a $5e-4$ weight decay value. For Resnet-18, a pretrained model is obtained directly from the Torchvision Python module.

On CIFAR-10, Resnet-20 achieves a validation accuracy of 93.0% and a test accuracy of 91.4%. On ImageNet, Resnet-18 achieves a validation accuracy of 68.3% and a test accuracy of 69.1%.

4.2 Quantization flow selection

Evaluation of the quantization flow introduced in Section 3.1 is performed by training quantized versions of Resnet-20. The accuracy achieved by these networks is evaluated on CIFAR-10's test set. To assert the relevance of this method, each quantized network is evaluated against a batch norm folded equivalent, by comparing the accuracy of both approaches under the same AiMC configurations.

Table 3. Bit width configurations summary

Configuration	Convolutions			First - Last - BN.		
	weights	inputs	outputs	weights	inputs	outputs
$Analog_8$	8 bit	8 bit	8 bit	8 bit	8 bit	8 bit
$Analog_4$	4 bit	4 bit	4 bit	4 bit	8 bit	8 bit
$Analog_2$	ternary	7 bit	6 bit	8 bit	8 bit	8 bit

Three bit width configurations are considered for evaluation. Two configurations aim at full 8 bit and 4 bit AiMC execution respectively, while the latter targets deployment on a ternary weight AiMC with 7 bit inputs and 6 bit encoded outputs, similar to [24]. In most reported quantization methods, the first and last layers of CNNs are excluded from the quantization flow [8, 22, 38]. These interface layers are included in the considered quantization flow. However, to avoid information loss at the CNN's inputs and outputs, 8 bit quantization is applied to the input and output of these interface layers. Under these considerations, 8 bit support logic is assumed for the CNN's first and last layers. As such, by assuming the same support logic can be used for batch normalization operation, 8 bit quantization is also applied to the inputs and outputs of the batch normalization layers. In addition, to relax the quantization constrains on batch normalization in the ternary configuration, 8 bit quantization is applied to the batch normalization's multiplicative and additive factors in this configuration. As the same acceleration logic is assumed for batch normalization and interface layers, 8 bit weights were also used for the input and output layers of CNNs using this configuration. A summary of these configurations is reported in Table 3.

The linear quantization described in Section 2.2 is applied following either the batch norm folded quantization flow or the quantization flow explained in subsection 3.1. To parametrize the linear quantization functions, AiMC specific constraints are taken into account. Activations in AiMCs are encoded using a sign-magnitude representation. The sign bit is used in combination with the weights to determine the direction of the activations contribution and the magnitude bits are used to determine the size of that contribution. In order to account for that representation, symmetric bounds are used to quantize the AiMC inputs ($l_{max} = -l_{min}$). On the contrary, as the first ADC reference, determining the sign of the output, is generally placed at the center of the summation line voltage range, ADCs used in AiMCs use a 2's complement representation. As such, the quantization function corresponding the ADC uses asymmetric quantization bounds ($l_{max} = -l_{min} - 1$) and a round down function to quantize the convolution outputs.

During training, straight through estimation is used to backpropagate the gradients through the rounding functions. Different strategies are applied to determine the floating point quantization range s_f. For the weights, s_f is learned as network parameter, as shown in [11], and is initialized to the absolute maximum value (AbsMax) of the floating point weights. For the activations, s_f is determined as a training hyperparameter. All quantized networks are implemented using Xilinx's Brevitas PyTorch framework [25].

Inspired by [11], different learning rates are applied to train the quantization scales and the parameters of the neural networks. In [11] the magnitude of the gradient updates are scaled as a function of number of neuron inputs and of the bit width. Here, the scaling is achieved through the use of different learning rates for the weights of the network and the ranges learned by the quantization functions. The different learning rates are determined using hyperparameter tuning. Due to the variety of bit widths used in the quantization configurations, learning rates for the convolution operations and the rest of the layers are separate, resulting in two different sets of learning rates.

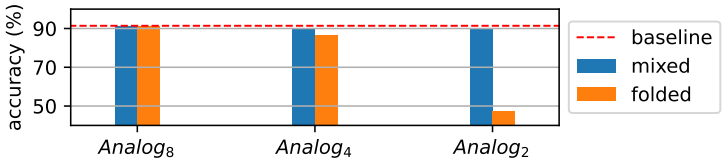

Fig. 7. Mixed signal and batch norm folded quantization flow test accuracies for Resnet-20 on CIFAR-10.

Hyperparameter tuning is performed using the Optuna framework [2]. A list of the hyperparameters tuned during this process, with the distributions each of them is sampled from, is provided in Appendix A. Each training configuration sampled during the procedure is used to train a network for a 100 epoch. To limit the number of hyperparameters tuned, cosine annealing learning rate is used in place of step decay. For each quantization configuration, the training configuration achieving the best validation accuracy is selected, trained for 200 epochs and tested on a previously unseen test set.

The classification accuracies obtained with quantization flow of subsection 3.1 and by batch norm folding, with the same quantization precision, are shown Fig. 3. Though batch norm folding is able to maintain close to floating point accuracy using full 8 bit execution, lower precision results in an observable drop in accuracy. By leveraging the higher precision batch normalization operations, the mixed signal configuration is able to maintain accuracices within 1.6% of the floating point baseline for all execution precisions.

The training configuration obtained for Resnet-20 is used to train Resnet-18 on ImageNet. In order to achieve convergence on ImageNet, however, modification to the training procedure are required. AbsMax initialization of the weight quantization scale results in the majority of the weights been quantized to zero, preventing the convergence of the network through gradient extinction. As such, weight quantization scales are initialized to one standard deviation of the floating point weights for the training of Resnet-18. Similarly, for the activations and accumulations, fixed quantization scales prevent convergence. These scales are thus set to track the absolute maximum average for the first 300,000 training batches before becoming a learnable parameter for the rest of the training.

The resulting ternary-weights Resnet-18 achieves a test accuracy of 64.7% on ImageNet. Table 4, shows that this accuracy is comparable with ternary and 2 bit quantized Resnet-18 from the literature, while also considering stricter quantization assumption for AiMC execution.

4.3 AiMC array scale control

To evaluate the hardware scaling technique exposed subsection 3.3, deployment on the AiMC accelerator introduced in [24] is targeted. Operating using ternary weights, 7 bit inputs and 6 bit outputs, the accelerator's computation pattern is illustrated Fig 8. The compute cells are composed of two SRAM memories storing ternary weights w and four switches, controlled by the cell weights w and the pulse width encoded activations a. Two long channel transistors, operating in saturation and controlled by the bias voltage $csbias$, act as current limiters to discharge summation lines.

Each cycle, two summations lines (sum_+, sum_-) are precharged to V_{dd}. Depending on the sign of each cell's local product $w \times a$, the switching circuitry selects the appropriate summation line. A charge Q_{sum}, proportional to the width of pulse encoded activation is discharged form the selected summation line. This discharge results in a voltage drop ΔV_{line} on the summation line, proportional to the DAC's unit time t_u, multiplied by the transistor's saturation current

Table 4. Quantized Resnet-18 accuracies on ImageNet test set as a function of the weights (W) activations (Act.), Accumulations (Acc.) and batch norm operation precisions. × denotes floating-point execution.

	W.	Act.	Acc.	Batch Norm	Accuracy
[37]	2 bit	×	×	×	68.0
[21]	ternary	×	×	×	61.8
[39]	ternary	×	×	×	66.6
[8]	2 bit	2 bit	×	×	64.4
[11]	2 bit	2 bit	×	×	66.7
[12]	2 bit	8 bit	×	×	67.9
Ours.	ternary	7 bit	6 bit	8 bit	64.7

I_{cell}, and inversely proportional to the summation line capacitance C_{line}.

$$\Delta V_{line} = a \times \frac{t_u \cdot I_{cell}}{C_{line}} \tag{13}$$

The accumulation of the discharge contributions on the summation lines results in a voltage difference between sum_+ and sum_-. This difference is sampled in the range $[0,\ V_{dd}]$ by a 6 bit differential SAR-ADC. Using (5), the transfer function of this AiMC array can be written as:

$$\begin{aligned} o &= \text{clamp}(\left\lfloor \frac{\delta v_{sum}}{\delta v_{adc}} \cdot aW \right\rfloor, -32, 31) \\ \delta v_{sum} &= \frac{t_u \cdot I_{cell}}{C_{line}}, \quad \delta v_{adc} = \frac{V_{dd}}{64} \end{aligned} \tag{14}$$

Simulations are performed using Pytorch-based analytical model of AiMC deployed operations, based on (14). In addition to the ADC quantization range related clipping modeled by (14), summation line level clipping is taken into account by this model. In the considered precharge discharge pattern, both positive and negative summation lines are precharged to V_{dd} at the beginning of each MVM operation. As such, the maximum voltage drop Δv_{line} supported on both positive and negative summation lines is limited by V_{dd}. This limit imposes a maximum l to the positive σ_+ and negative σ_- contributions of the AiMC's MVM operations.

$$l = \frac{C_l \cdot V_{dd}}{t_u \cdot I_{line}}, \quad \sigma_+ < l, \quad \sigma_- < l \tag{15}$$

To model this single ended summation clipping, convolution operations mapped on the simulated AiMC are decomposed as sums of their positive and negative parts. To do so weights and input activation are separated by sign.

$$\begin{aligned} W_+ &= W \circ (W > 0), \\ W_- &= W \circ (W < 0), \\ a_+ &= a \circ (a > 0), \\ a_- &= a \circ (a < 0) \end{aligned} \tag{16}$$

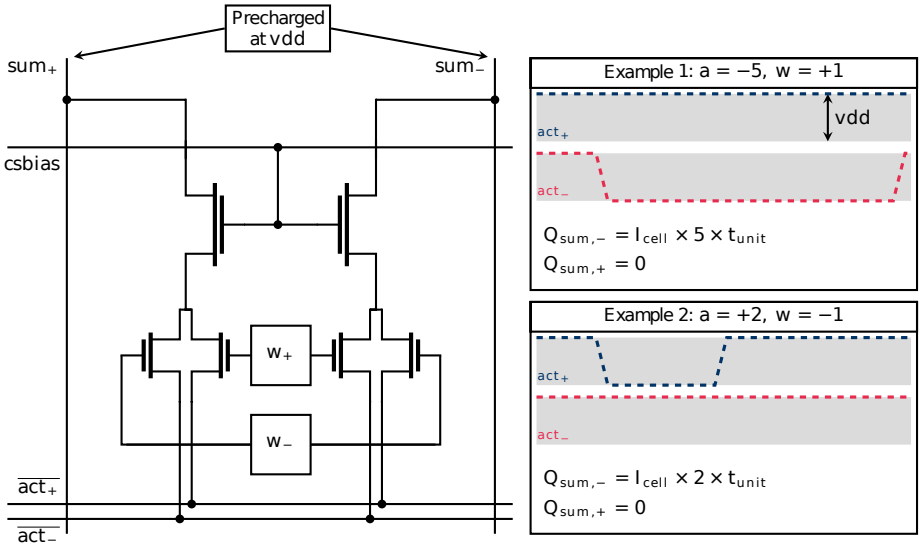

Fig. 8. Circuit schematic of compute cell concept. The weights w are stored in two SRAM cells, two long-channel transistors are utilized as current limiters, and switches determine whether to discharge sum_+ or sum_-, based on the signs of input activation a and weight w.

The positive and negative parts of the convolutions are then processed, clipped, and summed to generate the digital convolution outputs σ, as they are seen at the ADCs' inputs.

$$\begin{aligned} \sigma_+ &= clip(a_+ * W_+ + a_- * W_-, l) \\ \sigma_- &= clip(a_- * W_+ + a_+ * W_-, -l) \\ \sigma &= \sigma_+ - \sigma_- \end{aligned} \tag{17}$$

Quantization is then applied using (14) by replacing the digital MVM operation aW with the single ended clipped convolution outputs σ.

This accelerator integrates the unit time scaling proposed in Section 3.2, using a programmable delay line and encoding the number of delay events per unit time n as a 6 bit integer number, such that $t_u = n \times t_c$. To determine this number of events per unit time, the scale-based, operation-wise, determination pattern detailed in Section 3.3 is compared to sweeping-based, network-wide, empirical determination technique. To do so, trained ternary networks are deployed using the Pytorch simulated AiMC system and evaluated on their corresponding validation set. All possible, network-wide, delay events count values n are evaluated and the value resulting in the highest validation accuracy is selected for comparison. The test accuracy using this value network-wide is compared to the test accuracy obtained using scale-determined values for individual operation.

The validation and test accuracies obtained by sweeping the range of all possible counts are illustrated in Fig. 9. Different optimum counts per unit are observed for each network. An optimal of 12 counts per unit is found for Resnet-20 and of 7 counts for Resnet-18. This observation can be explained by the wider distribution of the convolution outputs, before ADC quantization, observed in Resnet-18 (Fig. 10). To cover the wider distribution, a smaller unit time is necessary. Fig. 11 compare the rescaling factor resulting from these unit times to the distribution of the operation-wise

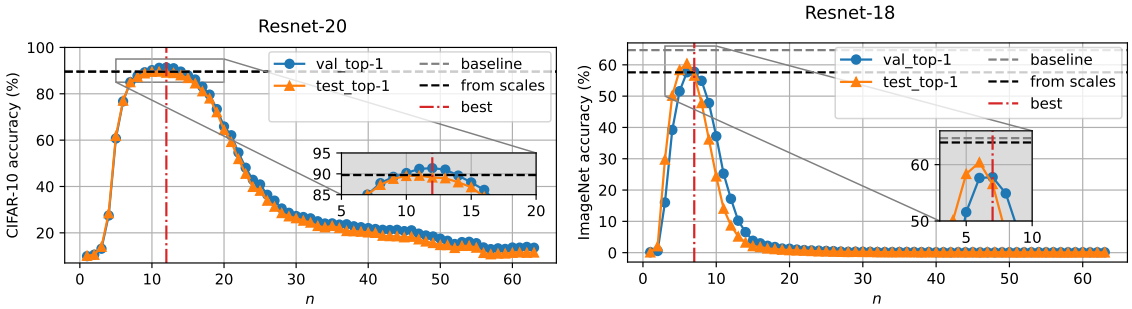

Fig. 9. Network test and validation accuracy vs delay events count per unit time n, for Resnet-20 on CIFAR-10 and Resnet-18 on ImageNet. Gray dashed line is the baseline accuracy of the ternary neural network. Red dashed line is the network wide delay events count resulting in the maximum validation accuracy. Black dashed line is the test accuracy obtained determining n from the scales of each operation.

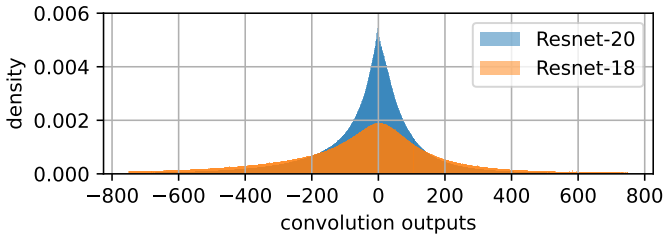

Fig. 10. Ternary convolution outputs distributions for Resnet-20 and Resnet-18 on CIFAR-10 and ImageNet.

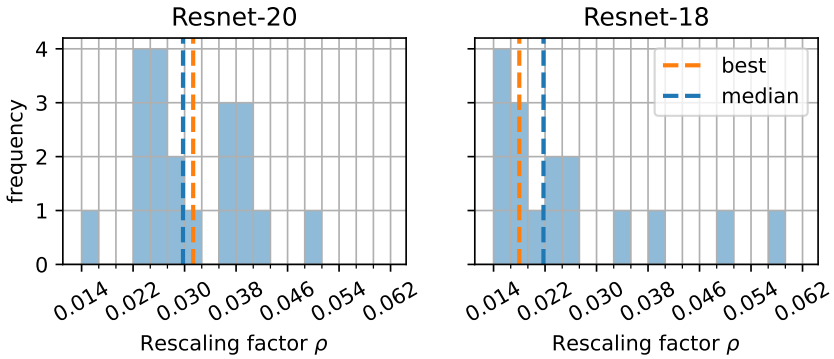

Fig. 11. Rescaling factors distribution across networks convolution operations. Each bin corresponds to a single value of n. Rescaling factor found using value sweeping is marked in red. Median rescaling factor is marked in blue.

factors deduced from the learned quantization scales. The factors selected by the sweeping methods are neighboring the median of the operation-wise deduced ones.

Thought accuracies close to the ternary baseline are achieved by both the network-wide and operation-wise methods for Resnet-20, significant accuracy loss is observed on Resnet-18 for both methods. To close this accuracy gap, additional

Table 5. Simulated hardware accuracies of ResNet-20 and ResNet-18 on CIFAR-10 and ImageNet

Network	Dataset	Method	Accuracy [%]		Baseline
			Raw	Retrained	
Resnet 20	CIFAR-10	Net wide ($n = 12$)	89.1	89.6	89.8
		Layer wise	89.5	**90.1**	
Resnet 18	ImageNet	Net wide ($n = 7$)	56.5	62.9	64.7
		Layer wise	57.6	**63.4**	

training is performed with the modeled accelerator. The networks are retrained for 5 epochs, lowering the previously determined learning rates by a factor 1000 for Resnet-20 and 10 for Resnet-18.

Table 5, reports all the accuracies obtained with both methods, before and after retraining. Operation wise scale control achieves 90.1% accuracy on CIFAR-10 with Resnet-20 and 63.4% on ImageNet, surpassing the network wide determination by 0.5% in both cases. Inspection of the summation lines clipping events frequency during the execution of the networks, reveals that this difference of accuracy can be attributed to a drastic reduction in the number of clipping events when operation-wise scaling is used. On Resnet-20 and Resnet-18, the number of clipping events is reduced by 58.8% and 40.9% respectively, when operation-wise scaling is used instead of network-wide scaling.

4.4 AiMC variability impact

The results presented in subsection 4.3 advocate for operation wise scale control over network wide scale control. However, as subsection 4.3 relies on a fully deterministic model of the arrays operation, these results fail to account for the inherent vulnerability of the analog circuits. Indeed, due to the analog nature of AiMC operation, process variations in their manufacturing process can cause variability in the results of the analog MVM operation [10].

To account for such variations, a modification for the model of subsection 4.3 is proposed. In this updated model, sources of stochasticity are modeled through the injection of noise to the analog values before quantization is applied by the ADC. This noise takes the form of a random variable ϵ such that:

$$o = clamp\left(\left\lfloor \frac{\delta v_{sum}}{\delta v_{adc}} \cdot \sigma \ + \ \epsilon \right\rfloor, -32, 31\right) \tag{18}$$

Where $\epsilon \sim \mathcal{N}\left(0, p \cdot \frac{\delta v_{sum}}{\delta v_{adc}}\right)$ is a normally distributed random variable whose standard deviation is proportional to the step size of the quantized MVM operation.

[24] reports that for a similar configuration to the one considered by this work, the variability observed at the output of the array should correspond to 52% of the ADC step size. To assert the relevance of the considered model, a comparison is established between analog MVM results obtained using the aforementioned model with $p = 0.52$ and results obtained from the measurements reported by [24] under an equivalent context. Fig. 12, illustrates the results from this comparison. Under the same configurations, the proposed updated model seem to match the observations reported by [24].

To evaluate the impact of AiMC variability on operation wise scale control compared to network wide scaling, the trained networks obtained subsection 4.3 are evaluated under different noise regimes with the updated AiCM model. Results for Resnet-18 on ImageNet and Resnet-20 on CIFAR-10 are reported in Fig 13. For each noise level, the considered

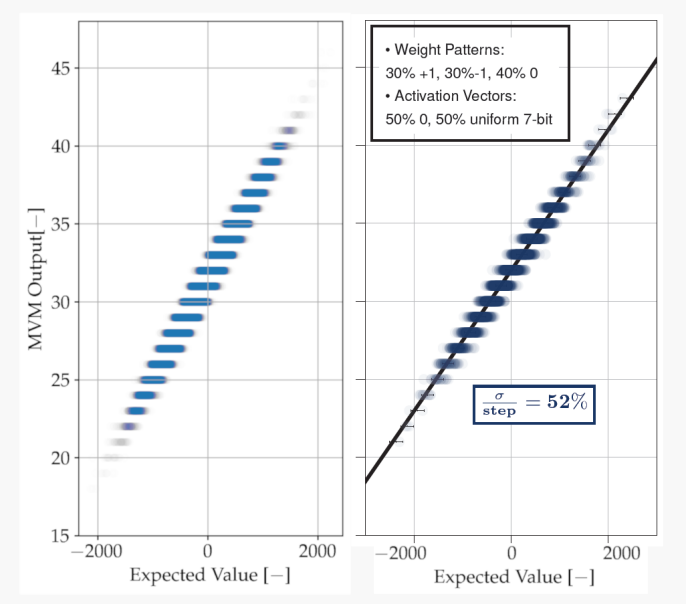

Fig. 12. Simulated analog MVM output for $p = 0.52$ as a function of the true MVM results (left) vs measured MVM result as measured by [24] (right).

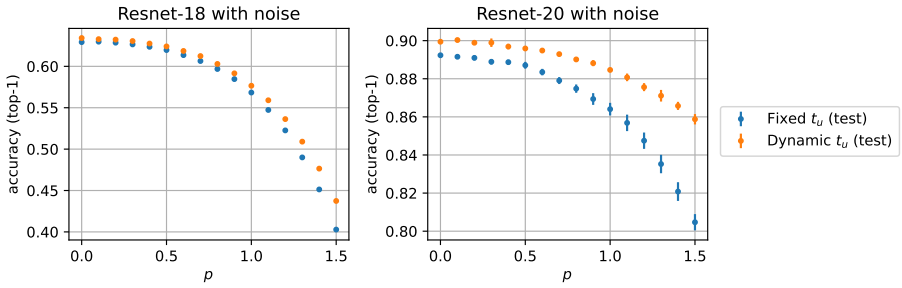

Fig. 13. Resnet-18 accuracy on Imagenet and Resnet-20 accuracy on CIFAR-10 as a function of the applied noise. In blue, accuracies obtained using network wide scaling. In orange, accuracies obtained using operation wise scaling.

network is evaluated 5 times with different random seeds. The average accuracy is reported as well as the standard deviation of the 5 runs. Though an accuracy degradation is observed for increasing noise levels for both networks and scaling patterns, across all of the reported noise magnitudes, operation-wise control continues to outperforms network wide scaling. Moreover, the accuracy gap between operation-wise and network-wide scaling widens with increasing noise levels. For Resnet-18 the accuracy gap increases from 0.5% without AiMC variability to 3.5% for $p = 1.5$. For Resnet-20 this same gap increase from 0.5% to 5.4%. These results further accentuate the advantages of dynamic over static scaling in AiMC.

5 CONCLUSION

This work addresses the discrepancies between the scaling assumptions of AiMC based neural networks accelerators and those of modern neural networks quantization methods. An AiMC compatible quantization flow is introduced and demonstrated with relevant residual networks on the CIFAR-10 and ImageNet datasets. By leveraging higher precision batch normalization while maintaining AiMC execution compatibility, the proposed quantization flow can limit the accuracy degradation caused by quantization. Ternary versions of Resnet-20 and Resnet-18 are obtained, achieving accuracies of 89.8% and 64.7% on CIFAR-10 and ImageNet respectively. AiMC scale control patterns are compared by deploying these networks on simulated AiMC arrays. Compared to standard sweep based AiMC scale determination methods, operation-wise scale control is shown to result in neural networks that are both more accurate and more robust to the inherent variability of the analog based MVM operation.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2623–2631.

[3] Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. [n.d.]. AI and Compute. https://openai.com/blog/ai-and-compute/. Accessed: 2021-05-04.

[4] Chaim Baskin, Natan Liss, Yoav Chai, Evgenii Zheltonozhskii, Eli Schwartz, Raja Giryes, Avi Mendelson, and Alexander M Bronstein. 2018. Nice: Noise injection and clamping estimation for neural network quantization. *arXiv preprint arXiv:1810.00162* (2018).

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

[6] Debjyoti Bhattacharjee, Nathan Laubeuf, Stefan Cosemans, Ioannis Papistas, Arindam Mallik, Peter Debacker, Myung Hee Na, and Diederik Verkest. 2021. Design-Technology Space Exploration For Energy Efficient AiMC-based Inference Acceleration. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[7] Avishek Biswas and Anantha P Chandrakasan. 2018. CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. *IEEE Journal of Solid-State Circuits* 54, 1 (2018), 217–230.

[8] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[9] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. 2018. Serving dnns in real time at datacenter scale with project brainwave. *iEEE Micro* 38, 2 (2018), 8–20.

[10] Jonas Doevenspeck, Peter Vrancx, Nathan Laubeuf, Arindam Mallik, Peter Debacker, Diederik Verkest, Rudy Lauwereins, and Wim Dehaene. 2021. Noise tolerant ternary weight deep neural networks for analog in-memory inference. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[11] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2020. Learned Step Size quantization. In *ICLR*. OpenReview.net.

[12] Cheng Gong, Yao Chen, Ye Lu, Tao Li, Cong Hao, and Deming Chen. 2020. Vecq: Minimal loss dnn model compression with vectorized weight quantization. *IEEE Trans. Comput.* (2020).

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

[14] Geoffrey Hinton. 2012. Neural networks for machine learning. Coursera (video lectures).

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

[16] Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. 2020. Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 112–128.

[17] Zhewei Jiang, Shihui Yin, Jae-Sun Seo, and Mingoo Seok. 2020. C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism. *IEEE Journal of Solid-State Circuits* 55, 7 (2020), 1888–1897.

[18] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[19] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

[20] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[21] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711* (2016).

[22] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. 2018. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191* (2018).

[23] Boris Murmann. 2020. Mixed-signal computing for deep neural network inference. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 1 (2020), 3–13.

[24] Ioannis Papistas, Stefan Cosemans, Bram Rooseleer, Jonas Doevenspeck, Myung-Hee Na, Arindam Mallik, Peter Debacker, and Diederik Verkest. 2021. A 22nm, 1540 $TOP/s/W$, 12 : 1 $TOP/s/mm^2$ in-Memory Analog Matrix-Vector-Multiplier for DNN Acceleration. In *2021 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE.

[25] Alessandro Pappalardo. [n.d.]. *Xilinx/brevitas*. https://doi.org/10.5281/zenodo.3333552

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[27] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10428–10436.

[28] Rajat Raina, Anand Madhavan, and Andrew Y Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*. 873–880.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[31] Abu Sebastian, Irem Boybat, Martino Dazzi, Iason Giannopoulos, V Jonnalagadda, Vinay Joshi, Geethan Karunaratne, Benedikt Kersting, Riduan Khaddam-Aljameh, SR Nandakumar, et al. 2019. Computational memory-based inference and training of deep neural networks. In *2019 Symposium on VLSI Technology*. IEEE, T168–T169.

[32] Xin Si, Jia-Jing Chen, Yung-Ning Tu, Wei-Hsing Huang, Jing-Hong Wang, Yen-Cheng Chiu, Wei-Chen Wei, Ssu-Yen Wu, Xiaoyu Sun, Rui Liu, et al. 2019. 24.5 a twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 396–398.

[33] Dave Steinkraus, Ian Buck, and PY Simard. 2005. Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 1115–1120.

[34] Hanbo Sun, Zhenhua Zhu, Yi Cai, Xiaoming Chen, Yu Wang, and Huazhong Yang. 2020. An energy-efficient quantized and regularized training framework for processing-in-memory accelerators. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 325–330.

[35] Rudolf Usselmann. 2018. *Opencores Floating Point Unit*. https://opencores.org/projects/fpu

[36] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. 2020. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits* 55, 6 (2020), 1733–1743.

[37] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 365–382.

[38] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[39] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).

A HYPERPARAMETERS TUNING

Table 6. Hyperparameter tuning configurations and results for ResNet-20 on CIFAR-10

Parameter	Distribution	Domain	Folded			Mixed		
			$Analog_8$	$Analog_4$	$Analog_2$	$Analog_8$	$Analog_4$	$Analog_2$
Digital act scale	uniform	$\{2^n \mid n \in [-1..3]\}$	2	8	2	2	2	4
Analog input scale	uniform	$\{2^n \mid n \in [-1..3]\}$	4	1	2	8	2	4
Analog output scale	uniform	$\{2^n \mid n \in [-1..3]\}$	1	1	8	0.5	1	4
Digital param. lr	log uniform	$[1e-5, 1e-1]$	$3.79e-2$	$2.48e-3$	$2.20e-2$	$5.03e-3$	$7.92e-2$	$8.24e-3$
Digital scale lr	log uniform	$[1e-5, 1e-1]$	$1.48e-5$	$1.03e-4$	$9.79e-2$	$2.05e-2$	$4.47e-4$	$3.64e-2$
Analog param. lr	log uniform	$[1e-5, 1e-1]$	$4.91e-4$	$5.53e-5$	$8.32e-2$	$2.31e-3$	$2.66e-4$	$2.90e-3$
Analog scale lr	log uniform	$[1e-5, 1e-1]$	$2.29e-5$	$6.84e-2$	$4.88e-3$	$3.05e-3$	$1.63e-3$	$5.48e-4$
Weight decay	log uniform	$[1e-6, 1e-3]$	$1.38e-5$	$5.71e-6$	$6.59e-4$	$5.58e-6$	$1.03e-6$	$1.91e-6$
Momentum	uniform	$[0.0, 0.99]$	0.87	0.19	0.37	0.58	0.57	0.50
Nesterov	categorical	$\{True, False\}$	$True$	$True$	$False$	$False$	$True$	$True$
Warm up steps	uniform	$\{250 \cdot n \mid n \in [0..8]\}$	1000	1500	1500	1250	1750	750
Test Accuracy (%)			91.3	86.2	47.1	90.9	89.7	89.8