

# Analog/Mixed-Signal Hardware Error Modeling for Deep Learning Inference

Angad S. Rekhi<sup>1</sup>, Brian Zimmer<sup>2</sup>, Nikola Nedovic<sup>2</sup>, Ningxi Liu<sup>3</sup>, Rangharajan Venkatesan<sup>2</sup>, Miaorong Wang<sup>4</sup>, Brucek Khailany<sup>2</sup>, William J. Dally<sup>1,2</sup>, C. Thomas Gray<sup>2</sup>

<sup>1</sup>Stanford University, <sup>2</sup>NVIDIA, <sup>3</sup>University of Virginia, <sup>4</sup>Massachusetts Institute of Technology

## ABSTRACT

Analog/mixed-signal (AMS) computation can be more energy efficient than digital approaches for deep learning inference, but incurs an accuracy penalty from precision loss. Prior AMS approaches focus on small networks/datasets, which can maintain accuracy even with 2b precision. We analyze applicability of AMS approaches to larger networks by proposing a generic AMS error model, implementing it in an existing training framework, and investigating its effect on ImageNet classification with ResNet-50. We demonstrate significant accuracy recovery by exposing the network to AMS error during retraining, and we show that batch normalization layers are responsible for this accuracy recovery. We also introduce an energy model to predict the requirements of high-accuracy AMS hardware running large networks and use it to show that for ADC-dominated designs, there is a direct tradeoff between energy efficiency and network accuracy. Our model predicts that achieving  $< 0.4\%$  accuracy loss on ResNet-50 with AMS hardware requires a computation energy of at least  $\sim 300$  fJ/MAC. Finally, we propose methods for improving the energy-accuracy tradeoff.

## KEYWORDS

Analog/mixed-signal circuits, multiply-accumulate, dot product, deep learning, inference accelerator, error modeling, ResNet-50

## 1 INTRODUCTION

Deep learning is becoming ubiquitous in modern classification and detection systems. However, state of the art deep neural networks (DNNs) can be computationally intensive; this makes inference energetically costly, which can limit the usage of DNNs in an energy-constrained setting. Any method that reduces the energy cost of DNN inference is therefore of interest to system designers.

Methods for improved energy efficiency can be broadly categorized by whether they work at the software or at the hardware level. The former includes all network-level methods, such as pruning [1, 2], distillation [3], and quantization [4], and all other approaches implemented on existing hardware. The latter deals with changing the fabric on which the inference computation is performed. The most popular hardware-level approaches for improving energy efficiency

involve the design of custom digital ASICs that are optimized for fixed-point dot product computations, with the circuitry designed so that the reuse of weights and/or activations significantly reduces the energy cost of memory access [5].

An alternative to all-digital ASICs is to use an analog approach. By encoding information in amplitude or time rather than bits, analog circuits have the potential to use less energy per operation than corresponding digital implementations, at the cost of increased susceptibility to amplitude and/or timing noise, as well as to device mismatch [6]. Interestingly, when analog approaches are used for neural network evaluation, these drawbacks can be ameliorated by the fact that deep networks are often resilient to small amounts of noise [7]. This makes analog circuits natural candidates for deep learning inference hardware.

Because data storage and movement are still most efficient in the digital domain and network inputs and outputs are often digital in nature, approaches in the literature tend to focus on analog/mixed-signal (AMS) implementations, in which the core dot product computation is performed in the analog domain while the inputs and outputs to these analog blocks are digital [6, 8].

The majority of AMS approaches to date have focused on the use of resistive arrays to perform dot product computation - weights are stored as conductances, activations are loaded in as voltages on wordlines, and their products are read out as currents on bitlines [9–21]. Another approach is to use switched-capacitor multipliers, which use passive charge redistribution to convert digital inputs into an analog output representing the scaled product of the inputs [22–24]. Both of these approaches have been shown to degrade network accuracy [19, 20, 24]. Prior work has focused on demonstrating the functionality of AMS circuits both as standalone computational units and as the hardware fabric on which neural networks are implemented [25, 26]. To date, most work has been focused on relatively small problems, such as MNIST and CIFAR classification, with some recent work starting to explore the use of AMS hardware for more complex problems, such as ImageNet classification [19, 20].

This work builds upon previous AMS-based approaches to inference by abstracting the AMS computational unit into an error-free portion with additive error, and then injecting that error at the network level to investigate the effect of AMS hardware on ImageNet classification with ResNet-50. Our method can model any system that performs digital-to-analog multiplications followed by analog-to-digital conversions, including approaches based on resistive arrays or switched capacitors. We describe our approach to error modeling and injection in Section 2. In Section 3, we present accuracy results with those error models, show that exposing the network to AMS error during retraining can recover a substantial amount of accuracy, and provide evidence suggesting that batch normalization layers are responsible for accuracy recovery. In Section 4, we introduce

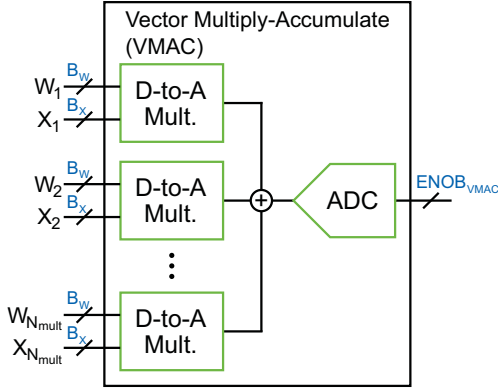
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6725-7/19/06...\$15.00

<https://doi.org/10.1145/3316781.3317770>



**Figure 1: Block diagram of the AMS VMAC unit considered in this work.**

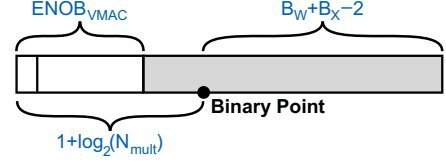
a basic energy model in order to tie accuracy to energy and use it to show that, for ADC-dominated designs, energy efficiency cannot be improved without hurting accuracy (and vice versa). Our model predicts that achieving  $< 0.4\%$  accuracy loss on ResNet-50 requires a computation energy of at least  $\sim 300$  fJ/MAC. Finally, we discuss methods to potentially improve upon the proposed model and results.

## 2 ERROR MODEL

The computational unit cell used in this work is an AMS vector multiply-accumulate (VMAC) cell; the cell’s block diagram is shown in Figure 1. This cell takes  $N_{\text{mult}}$  weight-activation pairs and outputs a single number representing the sum of the pairwise products; alternatively, the cell can be viewed as computing the dot product of two input vectors of length  $N_{\text{mult}}$ . In keeping with the dogma of digital data storage and transport, all inputs to and the output of this unit cell are digital. Input weights and activations are, respectively,  $B_W$ - and  $B_X$ -bit signed numbers (sign-magnitude representation). Each D-to-A multiplier performs a signed multiplication of its two inputs and returns the signed result in the analog domain (differentially). The sum (or average, as discussed below) of  $N_{\text{mult}}$  of these outputs is then passed to a differential analog-to-digital converter (ADC), which converts the analog dot product back to the digital domain with some effective number of bits of information ( $\text{ENOB}_{\text{VMAC}}$ ). We use  $\text{ENOB}_{\text{VMAC}}$  as an independent variable that represents all of the AMS error introduced in the VMAC cell (thermal noise and nonlinearity from the multipliers; thermal noise, nonlinearity, and quantization error from the ADC), referred to the input of the ADC.

Computing a dot product with an AMS VMAC cell entails information loss. Figure 2 represents that loss in terms of the precision of the ideal digital vector dot product. The ideal multiplication of a  $B_W$ -bit signed weight by a  $B_X$ -bit signed activation (both using sign-magnitude representation) results in a product with  $B_W+B_X-2$  magnitude bits and one sign bit. The ideal addition of  $N_{\text{mult}}$  of these numbers results in an additional  $\log_2(N_{\text{mult}})$  bits of precision. With no full-scale adjustment (see Section 4), the ADC picks out the most significant  $\text{ENOB}_{\text{VMAC}}$  of these bits.

When the VMAC cell presented here is used to compute convolutions in neural networks, outputs of multiple cells (tiled across space and/or time) will generally need to be added to arrive at a single output activation, since, in general,  $N_{\text{mult}}$  will be smaller than



**Figure 2: Representation of lost precision with AMS implementation of VMAC. White section is recovered bits (MSB is sign bit); grey portion is lost information. Note that the binary point may or may not fall within the recovered section.**

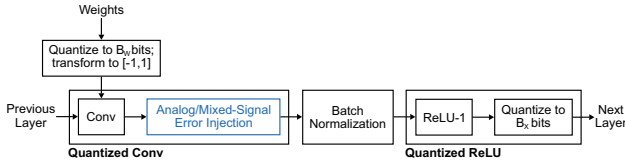
$N_{\text{tot}}$  (the total number of multiplications needed to compute a single output activation). Thus, each VMAC cell computes lossy partial products. Because our VMAC cell has digital outputs, sums of these outputs are computed digitally. Therefore, there is no extra loss in precision beyond that described by Figure 2 (i.e., beyond each AMS VMAC cell).

The loss of information in the bits of lesser significance can lead to degradation in network accuracy when our AMS VMAC cells are used to compute partial sums for convolutional layers. To quantitatively investigate this effect, we run network-level simulations with injected AMS error. We use Distiller, an open-source PyTorch-based package chiefly meant for neural network compression, to implement AMS error injection [27]. Distiller provides out-of-the-box support for several methods of quantization of weights and activations; we use DoReFa, which compresses and quantizes convolutional layer weights, clips and quantizes activations, and allows for quantization in the training loop using a straight-through estimator [28]. As opposed to the original implementation, Distiller’s version of DoReFa does not quantize gradients. Batch normalization weights and biases are also not quantized; this is acceptable because, after retraining, weights can be folded into the convolutional layer, while biases can be added digitally at little extra energy cost.

DoReFa takes care of convolutional weight and activation quantization; we build on top of DoReFa to inject AMS error. We lump this error to the output of the digital summation of multiple VMAC cell outputs and inject it there; this has the advantage of not breaking apart the convolution computation. To compute the magnitude of the error to inject, we start with the error at the output of each AMS VMAC cell,  $\mathcal{E}_{\text{VMAC}}$ . We assume that this error is additive and data-independent. By definition of  $\text{ENOB}_{\text{VMAC}}$ , even without knowing the distribution of the error, we know that it has variance  $\text{LSB}^2/12$  [29]. DoReFa’s method of quantization caps all weights and activations at 1, allowing us to easily keep track of the binary point in the ideal vector dot product (as shown in Figure 2), and thus allowing calculation of the LSB in terms of two key AMS VMAC parameters:

$$\begin{aligned} \text{Var}(\mathcal{E}_{\text{VMAC}}) &= \frac{\text{LSB}^2}{12} \\ &= \frac{(2^{1+\log_2(N_{\text{mult}})-\text{ENOB}_{\text{VMAC}})})^2}{12} \\ &= \frac{(N_{\text{mult}}2^{-(\text{ENOB}_{\text{VMAC}}-1)})^2}{12}. \end{aligned} \quad (1)$$

As discussed earlier, there is no extra error added after each AMS VMAC, as partial sums are accumulated digitally. Therefore, to compute the total error,  $\mathcal{E}_{\text{tot}}$ , we need to model the sum of multiple



**Figure 3: Block diagram of quantized convolutional layer with injected AMS error, as implemented in Distiller.**

individual errors with the variance calculated above. If  $N_{tot}$  multiplications must be performed to compute a single output activation, then  $N_{tot}/N_{mult}$  VMACs are required. Assuming that the AMS errors at the output of each VMAC are independent and identically distributed, the total error will be roughly normally distributed, with variance equal to the following:

$$\begin{aligned} Var(\mathcal{E}_{tot}) &= \frac{N_{tot}}{N_{mult}} Var(\mathcal{E}_{VMAC}) \\ &= \frac{N_{tot} (\sqrt{N_{mult}} 2^{-(ENOB_{VMAC}-1)})^2}{12}. \end{aligned} \quad (2)$$

We inject this error during only the forward pass, leaving the backward pass untouched. DoReFa-based quantization and AMS error injection together incur a roughly 50% overhead in forward pass computation time compared to the out-of-the-box FP32 network; all runs used 7 NVIDIA Tesla V100 GPUs. Figure 3 shows where AMS error is injected relative to a convolutional layer in ResNet-50<sup>1</sup>.

Before moving on to a discussion of simulation results, we highlight several important aspects of our approach to AMS error injection. First, we quantize and inject AMS error to every layer of ResNet-50, including the first and last layers. To bound the inputs to the first layer, we rescale them by the maximum input activation value so that they lie in the range  $[-1, 1]$  before quantizing; this step is unnecessary for all further layers because DoReFa replaces every activation function with a ReLU that clips at 1, therefore bounding the next layer’s input activations. Second, while we inject AMS error into every layer when using the network for evaluation (to properly model AMS hardware), we leave out AMS error injection from the last layer while training the network (all other layers still have injected error during training). We found that injecting AMS error into the last layer while training led to a loss of the network’s ability to learn, and this workaround provides a working solution.

Although the discussion thus far assumes that each VMAC implements an analog summation of the outputs of multiple D-to-A multipliers, our approach to AMS error modeling is still applicable even if these outputs are averaged rather than added. This is because the extra division by  $N_{mult}$  that averaging entails does not lead to a loss of information; it simply moves the binary point to just right of the sign bit in the partial sum. The only required change would be in the interpretation of the digital output from each VMAC; if averaging-based AMS hardware were used to implement dot product computation, the final accumulated sum would need to be scaled up by  $N_{mult}$  before being passed to the next layer. This would scale both the signal and the noise by equal amounts, so there would be no difference in the relative amount of injected error. We assume addition-based AMS hardware just to avoid having to perform this rescaling in the source code.

<sup>1</sup>Modification to Distiller source code available upon request.

### 3 ACCURACY RESULTS

Throughout this section, we compare top-1 accuracies on the ImageNet validation dataset; in our runs, top-5 accuracies generally tracked top-1 accuracies. We use “retraining” to refer to taking a pretrained FP32 network and continuing to train it after modifying the network to reflect the intended underlying hardware. All runs involving retraining use a minibatch size of 1024 with a learning rate of 0.004; all other hyperparameters are unchanged from the default for Distiller’s ResNet-50. Learning rate scheduling is not implemented here; if the validation set accuracy begins to decrease after some time, the training run is stopped and the maximum validation accuracy is reported. Each reported accuracy is the sample mean of five passes of the validation dataset through the network, with error bars showing the sample standard deviation.

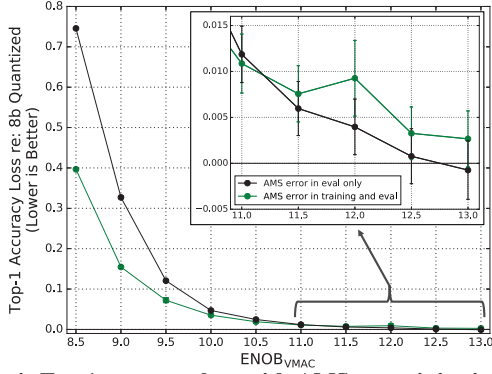
We begin by investigating the tolerance of ResNet-50 to weight and activation quantization without AMS error injection (i.e., assuming a fully digital hardware implementation) using DoReFa. These investigations provide accuracy baselines for subsequent simulations that include AMS error. Table 1 shows these results. Retraining with 8b weight and activation quantization provides full recovery of accuracy (in fact, the mean accuracy with 8b quantization is slightly higher than that of the original network, but the difference is only about one sample standard deviation away from the mean). With 6b weights and activations, the mean accuracy drops by about 2% relative to that of the FP32 network. Finally, with 6b weights and 4b activations, the mean accuracy drops by about 17% relative to that of the original network. We are interested in AMS configurations that approach the accuracy of the original network; because it is unlikely that AMS error injection can increase the accuracy beyond that of the AMS error-free quantized version of the network, we choose only the  $B_W=B_X=8$  and  $B_W=B_X=6$  configurations for further inquiry.

Figure 4 shows top-1 accuracy loss with AMS error injection relative to an 8b quantized network (i.e., relative to the second row in Table 1). For all values of  $ENOB_{VMAC}$ ,  $N_{mult}$  was set to 8, and for the runs in which AMS error was added at evaluation time only, the best epoch of the quantized retrained network for each value of  $ENOB_{VMAC}$  was used. For  $ENOB_{VMAC} \leq 11$ , including AMS error during retraining allows recovery of up to half of the accuracy lost when this error is added only at evaluation time; this effect is further investigated below. For  $ENOB_{VMAC} > 11$ , retraining with AMS error leads to slightly reduced accuracy compared to just including it at evaluation time; this may be because the AMS error is so small that including it during retraining is akin to retraining without error, which leads the network away from a previously found optimum. The accuracy first reaches within a sample standard deviation of the 8b quantized network’s accuracy for  $ENOB_{VMAC} = 12.5$ .

Figure 5 shows top-1 accuracy loss with AMS error injection relative to a 6b quantized network (the third row in Table 1). Again,

**Table 1: Top-1 accuracy for various weight and activation bitwidths after retraining with DoReFa; no AMS error added.**

| Quantization       | Top-1 Accuracy | Samp. Std. Dev. |
|--------------------|----------------|-----------------|
| FP32               | 0.778          | 6.96E-4         |
| $B_W = 8, B_X = 8$ | 0.781          | 2.82E-3         |
| $B_W = 6, B_X = 6$ | 0.757          | 9.83E-4         |
| $B_W = 6, B_X = 4$ | 0.606          | 7.03E-4         |

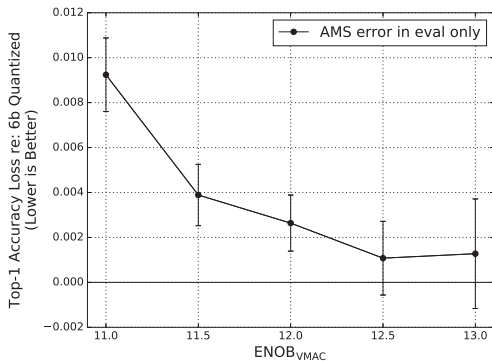


**Figure 4: Top-1 accuracy loss with AMS error injection (with  $N_{\text{mult}} = 8$ ) relative to an 8b quantized network.**

$N_{\text{mult}} = 8$  for all runs. Based on the results shown in Figure 4, for this precision we only investigated adding AMS error at evaluation time. For all values of  $\text{ENOB}_{\text{VMAC}}$ , we used the best epoch of the quantized retrained network without AMS error (as opposed to finding the best epoch for each value of  $\text{ENOB}_{\text{VMAC}}$ , as done for the data in Figure 4). We find that  $\text{ENOB}_{\text{VMAC}} = 11$  is the cutoff for  $< 1\%$  top-1 accuracy drop, while  $\text{ENOB}_{\text{VMAC}} = 12.5$  is again the cutoff for reaching within a sample standard deviation of the 6b quantized network’s accuracy.

To explore the mechanism behind the improvement in accuracy when AMS error is injected during training at relatively low  $\text{ENOB}_{\text{VMAC}}$ , we selectively froze different kinds of layers while retraining and compared the accuracy results, which are shown in Table 2. For all runs, we fixed  $\text{ENOB}_{\text{VMAC}} = 10$  and  $N_{\text{mult}} = 8$ . Freezing the convolutional layers does not significantly affect accuracy, while freezing the batch norm layers and/or fully-connected layer decreases accuracy by 4–8.5%. These results show that the batch norm layers are primarily responsible for the network’s ability to recover a fraction of the lost accuracy when retrained with AMS error injection in the loop.

We further investigated how exactly the batch norm layers help to recover accuracy by saving and visualizing activation means at the output of every convolutional layer (the location where AMS error is injected). Figure 6 shows these means, which are evaluated across the entire ImageNet validation dataset, for one of the layers, and for



**Figure 5: Top-1 accuracy loss with AMS error injection (with  $N_{\text{mult}} = 8$ ) relative to a 6b quantized network.**

**Table 2: Results of selective freezing simulations.**

| Frozen Layers | Top-1 Accuracy Loss re: 8b | Samp. Std. Dev. |
|---------------|----------------------------|-----------------|
| None          | 0.0353                     | 2.83E-3         |
| Conv          | 0.0341                     | 2.95E-3         |
| BN            | 0.0886                     | 3.24E-3         |
| FC            | 0.0774                     | 3.16E-3         |
| BN and FC     | 0.120                      | 2.96E-3         |

six different varieties of ResNet-50: FP32, retrained with 8b quantization, and retrained with 8b quantization along with varying levels of AMS noise. In 43 of the 53 convolutional layers of the network (including downsampling layers), the network appears to learn to push the means of the activations away from zero to combat added AMS noise; moreover, the larger the noise, the greater the push. This seems to be the underlying mechanism for the improvement in accuracy at relatively low  $\text{ENOB}_{\text{VMAC}}$ .

## 4 DISCUSSION

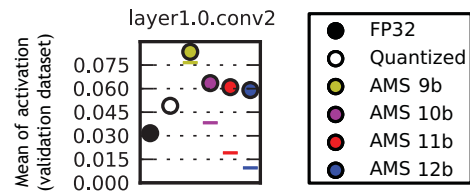
In this section, we present and explore a simple energy model for the AMS VMAC, compare the results of our models to prior work in this space, and propose ideas for extending the work presented here for improvement in accuracy and/or energy.

As a simple energy model, we assume that the VMAC energy is dominated by the energy of the ADC [19], and that  $\text{ENOB}_{\text{VMAC}} = \text{ENOB}_{\text{ADC}}$ . Our results therefore provide a lower bound on energy and an upper bound on accuracy. Our model for ADC energy as a function of resolution is based on a lower bound on the state of the art, as reported in Murmann’s ADC survey [30], and as depicted in Figure 7. This simple model results in the following bound on ADC energy and corresponding energy per MAC operation ( $E_{\text{MAC}}$ ):

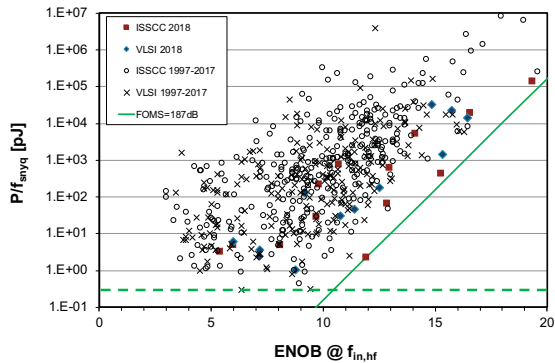
$$E_{\text{ADC}}(\text{ENOB}) \geq \begin{cases} 0.3 pJ & \text{ENOB} \leq 10.5 \\ 10^{0.1(6.02 \cdot \text{ENOB} - 68.25)} pJ & \text{ENOB} > 10.5 \end{cases} \quad (3)$$

$$E_{\text{MAC}}(\text{ENOB}_{\text{VMAC}}, N_{\text{mult}}) = \frac{1}{N_{\text{mult}}} E_{\text{ADC}}(\text{ENOB}_{\text{VMAC}}) \quad (4)$$

Figure 8 uses this energy model and the accuracy results from the previous section to relate  $\text{ENOB}_{\text{VMAC}}$ ,  $N_{\text{mult}}$ , accuracy, and energy on a single plot. Accuracy results for  $N_{\text{mult}} \neq 8$  are obtained by mapping results from  $N_{\text{mult}} = 8$  using the equation for AMS error magnitude presented in Section 2. This plot can be used as a lookup table by circuit designers to evaluate the network-level impact of circuit-level design choices, or by system designers to choose hardware based on accuracy or energy specifications.



**Figure 6: Means of activations (circles) and AMS error std. dev. (dashes) at the output of a representative convolutional layer.**



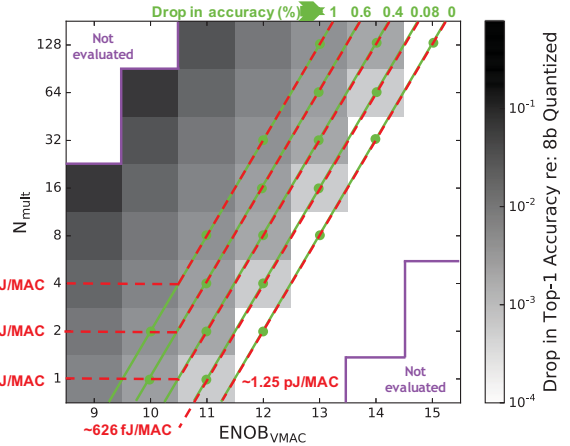
**Figure 7: Murmann’s survey of ADCs in the literature as of July 2018; adapted from [30], with slightly shifted Schreier FOM line and superimposed constant energy per sample line.**

An important conclusion to be made from Figure 8 is that, when  $ENOB_{VMAC}$  and  $N_{mult}$  are considered to be independent variables, the resultant accuracy loss and minimum energy per MAC ( $E_{MAC,min}$ ) as a function of these two variables track one another (their level curves are parallel). This occurs because our energy and error models have inverse dependencies on  $ENOB_{VMAC}$  and  $N_{mult}$ . On the error side, for each extra digitized bit, the variance of the total error drops by a factor of four; meanwhile, the effect of higher  $N_{mult}$  is to introduce quadratically greater error per VMAC but require linearly fewer VMACs, resulting in an overall linear dependence (Eq. 2). On the energy side, high-resolution ADCs are thermal noise-limited, resulting in a quadrupling of energy per conversion for each extra bit of resolution [29], while  $N_{mult}$  amortizes the energy cost of the ADC, resulting in an inverse dependence (Eq. 3-4). Thus, in the thermal noise-limited design space, increasing  $N_{mult}$  and  $ENOB_{VMAC}$  together to keep accuracy loss constant also keeps  $E_{MAC,min}$  constant.

Therefore, according to our error and energy models, accuracy loss and  $E_{MAC,min}$  have a one-to-one relationship in the design space of interest; there is no  $(ENOB_{VMAC}, N_{mult})$  pair that will improve one metric without harming the other. This relationship allows us to directly bound the energy required to achieve a certain level of network accuracy: for example, for  $< 0.4\%$  accuracy loss,  $E_{MAC,min} = \sim 313$  fJ, while for  $< 1\%$  accuracy loss,  $E_{MAC,min} = \sim 78$  fJ.

Prior work in this space has shown that the nonidealities of resistive crossbar-based hardware can lead to significant accuracy degradation for complex DNNs [20]. Here, we have shown that even if the digital-to-analog multiplier unit (whether based on resistive arrays, switched capacitors, or any other method) is error-free, the error introduced by conversion of the analog dot product back into the digital domain results in accuracy degradation with a direct energy-accuracy tradeoff. Our use of Murmann’s ADC survey extends prior work based on first-order ADC energy models [19] to reflect the current state of the art in ADC design.

Looking beyond the simple AMS hardware model that we have used in this work, there are a number of possible methods that can be pursued to lower the  $ENOB_{VMAC}$  and/or energy required for a specified maximum top-1 accuracy loss. The most pressing need is for a network-level method that minimizes the accuracy loss when AMS error is introduced; this would require no hardware-level tradeoffs in order to implement, and basically represents a “free



**Figure 8: Accuracy loss (relative to 8b quantized network) for various  $(ENOB_{VMAC}, N_{mult})$  configurations; accuracy loss and energy/MAC level curves overlaid in green and red, resp.**

lunch.” For example, Figure 4 indicates that for  $ENOB_{VMAC} \leq 10$ b, our retraining method recovers  $\sim 0.5$ b worth of accuracy, which is equivalent to a  $\sim 2\times$  reduction in  $E_{MAC,min}$ .

There are also several hardware-level methods that can be pursued. One method, based on long multiplication, is to partition each multiply into several multiplies with smaller operands, then add the appropriately shifted results in the digital domain. For example, splitting a multiplication of two 8b numbers into multiplications of 4b numbers would require computing four partial products; in general, splitting the weight into  $N_W$  parts and the activation into  $N_X$  parts would require  $N_W N_X$  multiplications of  $B_W/N_W$ -bit and  $B_X/N_X$ -bit numbers. Because the full precision of any partial product is smaller than that of the whole product, a lower-resolution ADC could be used than in the unpartitioned case while still incurring less injected error overall. This method potentially also reduces the energy per AMS MAC, as long as the energy of a lower-resolution conversion is less than  $1/(N_W N_X)$  times that of a higher-resolution conversion. Additionally, a lower-precision conversion could be performed for the partial product(s) of low significance, further saving energy. This method therefore reduces error and possibly also energy at the cost of decreased speed or increased area (depending on whether the partial products are computed in serial or in parallel, respectively).

Another method of error reduction is to subtract the quantization error incurred by the ADC in one cycle from the partial dot product computed in the next cycle. This can be shown to be equivalent to using a first-order delta-sigma modulator in place of an ADC. This method requires some level of output stationarity, as successive outputs of a single VMAC must be added for this to work correctly, and also requires the last conversion to be performed at a higher resolution than the rest. This method of error recycling reduces the total incurred quantization error, but does not change the impact of thermal noise. Reduction in error can be traded for reduction in energy by working with an ADC with nominally lower ENOB.

A third method of error reduction, implemented in [24], is to scale the ADC reference voltage with respect to the multiplier supply in order to play with the dynamic range-resolution tradeoff. By making the ADC reference voltage smaller than the multiplier supply (ignoring parasitic effects), at least one of the most significant

magnitude bits of the partial dot product is cut off (set to 0); the resolution of the ADC can then be increased (i.e., lower-significance bits can be accessed) until thermal noise considerations begin to significantly impact conversion energy. The effectiveness of this scheme is network- and data-dependent, and therefore needs to be confirmed with runs of ResNet-50 with ImageNet data.

Our consideration of AMS hardware focuses on its impact on inference accuracy and energy efficiency. Though outside the scope of this work, a full evaluation of AMS hardware must also include its performance in terms of speed and chip area, as well as the impact of variations in process, voltage, and temperature.

Finally, we suggest several ways to improve our error models. One method that would be closer to a hardware implementation would be to split up the convolution into VMAC-sized units and inject error at the output of each VMAC separately. This avoids assuming that these additive errors from separate VMACs are uncorrelated, but at the cost of slowing down the computation of each convolution. Modeling the error of the multipliers and ADC separately would allow even more fine-grained analysis of the VMAC. If the resulting slowdown is too severe, this modeling can be performed for evaluation only, if the ultimate goal is to design hardware for inference. Including non-additive and data-dependent errors (due to, for example, capacitor or resistor mismatch) would also be valuable. The resulting error models, as well as more sophisticated energy models, can be substituted into the presented framework (i.e., Figure 8) to arrive at an improved estimate of the energy-accuracy tradeoff.

## 5 CONCLUSION

We have presented an approach to modeling the error introduced by AMS dot product hardware that abstracts the computational cell into an error-free portion with additive error. We have used this model to predict top-1 ImageNet classification accuracy loss when ResNet-50 is implemented on AMS hardware, and have found that retraining with AMS error in the loop teaches batch normalization layers to push activation means away from zero, often resulting in substantial accuracy recovery. Based on a simple energy model, we have found that ADC-dominated designs exhibit a direct trade-off between energy efficiency and network accuracy; specifically, achieving  $< 0.4\%$  accuracy loss on ResNet-50 requires a computation energy of at least  $\sim 300$  fJ/MAC. Finally, we have proposed several methods for improvement, including modeling each computational unit separately, partitioning each multiplication, and recycling the quantization error. Our results can be applied to evaluate the feasibility of implementing complex DNNs on AMS hardware.

## REFERENCES

- [1] Y. Le Cun, J. S. Denker, and S. A. Solla. 1990. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*, 598–605.
- [2] S. Han, J. Pool, J. Tran, and W. J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. In *Advances in Neural Information Processing Systems 28*, 1135–1143.
- [3] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the Knowledge in a Neural Network. *arXiv*, 1–9.
- [4] S. Han, H. Mao, and W. J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*. San Juan.
- [5] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105, 12, (Dec. 2017), 2295–2329.
- [6] R. Sarpeshkar. 1998. Analog Versus Digital: Extrapolating from Electronics to Neurobiology. *Neural Computation*, 10, 7, 1601–1638.
- [7] B. Reagan, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei. 2018. Ares: a framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. San Francisco.
- [8] B. Murmann, D. Bankman, E. Chai, D. Miyashita, and L. Yang. 2016. Mixed-signal circuits for embedded machine-learning applications. In *49th Asilomar Conference on Signals, Systems and Computers*.
- [9] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang. 2015. Merging the interface: power, area and accuracy co-optimization for RRAM crossbar-based mixed-signal computing system. In *Proceedings of the 52nd Annual Design Automation Conference*. San Francisco.
- [10] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost, and D. Strukov. 2015. Efficient training algorithms for neural networks based on memristive crossbar circuits. In *Proceedings of the International Joint Conference on Neural Networks*. Killarney.
- [11] L. Xia, B. Li, T. Tang, P. Gu, X. Yin, W. Huangfu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang. 2016. MNSIM: Simulation Platform for Memristor-based Neuromorphic Computing System. In *Proceedings of the Conference on Design, Automation & Test in Europe*. Dresden.
- [12] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramanian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. Seoul.
- [13] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams. 2016. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proceedings of the 53rd Annual Design Automation Conference*. Austin.
- [14] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen. 2016. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Taipei.
- [15] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang. 2017. Binary Convolutional Neural Network on RRAM. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Chiba.
- [16] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang. 2017. Accelerator-friendly Neural-network Training: Learning Variations and Defects in RRAM Crossbar. In *Proceedings of the Conference on Design, Automation & Test in Europe*. Lausanne, 19–24.
- [17] A. Ankit, A. Sengupta, P. Panda, and K. Roy. 2017. RESPARC: A Reconfigurable and Energy-Efficient Architecture with Memristive Crossbars for Deep Spiking Neural Networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. Austin.
- [18] A. Ankit, A. Sengupta, and K. Roy. 2017. TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design. In *Proceedings of the 36th International Conference on Computer-Aided Design*. Irvine, 533–540.
- [19] Y. Kim, H. Kim, D. Ahn, and J.-J. Kim. 2018. Input-Splitting of Large Neural Networks for Power-Efficient Accelerator with Resistive Crossbar Memory Array. In *Proceedings of the International Symposium on Low Power Electronics and Design*. Seattle.
- [20] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan. 2018. Rx-Caffe: Framework for evaluating and training Deep Neural Networks on Resistive Crossbars. *arXiv*.
- [21] Y. Kim, H. Kim, and J.-J. Kim. 2018. Neural Network-Hardware Co-design for Scalable RRAM-based BNN Accelerators. *arXiv*.
- [22] Y. P. Tsvitvidis and D. Anastassiou. 1987. Switched-Capacitor Neural Networks. *Electronics Letters*, 23, 18, 958–959.
- [23] D. Bankman and B. Murmann. 2015. Passive charge redistribution digital-to-analogue multiplier. *Electronics Letters*, 51, 5, 386–388.
- [24] D. Bankman and B. Murmann. 2016. An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS. In *2016 IEEE Asian Solid-State Circuits Conference, A-SSCC 2016 - Proceedings*.
- [25] E. H. Lee and S. S. Wong. 2017. Analysis and Design of a Passive Switched-Capacitor Matrix Multiplier for Approximate Computing. *IEEE Journal of Solid-State Circuits*, 52, 1, 261–271.
- [26] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann. 2018. An always-on 3.8 $\mu$ J/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS. In *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*.
- [27] N. Zmora, G. Jacob, and G. Novik. 2018. Neural Network Distiller. (2018).
- [28] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv*.
- [29] M. Pelgrom. 2017. *Analog-to-Digital Conversion*. (3rd ed.). Springer.
- [30] B. Murmann. 2018. ADC Performance Survey 1997-2018. (2018). <https://web.stanford.edu/~murmann/adcsurvey.html>.