Variability-Aware Training and Self-Tuning of Highly Quantized DNNs for Analog PIM

1st Zihao Deng
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin TX, USA
zihaodeng@utexas.edu

2nd Michael Orshansky
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin TX, USA
orshansky@utexas.edu

***Abstract*—DNNs deployed on analog processing in memory (PIM) architectures are subject to fabrication-time variability. We developed a new joint variability- and quantization-aware DNN training algorithm for highly quantized analog PIM-based models that is significantly more effective than prior work. It outperforms variability-oblivious and post-training quantized models on multiple computer vision datasets/models. For low-bitwidth models and high variation, the gain in accuracy is up to 35.7% for ResNet-18 over the best alternative.**

We demonstrate that, under a realistic pattern of within- and between-chip components of variability, training alone is unable to prevent large DNN accuracy loss (of up to 54% on CIFAR-100/ResNet-18). We introduce a self-tuning DNN architecture that dynamically adjusts layer-wise activations during inference and is effective in reducing accuracy loss to below 10%.

I. Introduction

Analog hardware accelerators for DNNs, based on the processing in memory (PIM) architectures, perform computation within a memory array that dramatically reduces data movement, eliminates the von Neumann bottleneck, and the associated costs in energy and latency [1]. A key compute kernel needed for DNN inference is a matrix-vector multiplication (MVM) array. Analog PIM accelerators rely on the possibility of efficient dot-product calculation using analog-based current summation and have been shown for multiple device technologies, including Resistive RAM, Flash, and MRAM [2]–[7]. DNN weights are represented by analog conductances (resistances) of memory cells. With the wordline voltages representing activations, the output current in a bitline is proportional to the activation-weight dot product [8].

Though, in theory, a continuous range of conductance is possible, in practice, there is a discrete set of feasible conductance values. The number of feasible conductance levels depends on technology. RRAM and Flash allow more than two states per cell. For example, there is currently a production-ready 5-bits per cell Flash technology available [9]. At the same time, the precision of activations is limited by the resolution of DACs (producing wordline voltages/input activations) and ADCs (quantizing the dot-product results).

Conventional DNN training is based on single-precision representation of weights and activations. Aggressive quantization of both activations and weights means that PIM-based DNNs need to rely on quantization-aware training (QAT) that directly incorporates quantization error into training [10]–[14]. As we demonstrate, deploying models on highly quantized platforms without QAT, or using naive post-training quantization, leads to drastic quality loss.

Yet currently available QAT algorithms focus on issues around digital quantizaiton. They ignore another significant challenge for practical adoption of analog PIM DNN accelerators: how to handle the non-ideality of weight representations as physical variables, e.g., conductances. The use of bit-cell conductance as a non-binary quantity intrinsically makes it subject to being impacted by the randomness of semiconductor fabrication. Nanometer scale semiconductor devices exhibit large variability across multiple spatial scales [15]. As a result, the programmed conductances of memory cells will deviate from their intended values. These deviations in memory cells' conductances are then translated into variations of weights inside DNNs once they are deployed on PIM devices. Such parameter variation significantly degrades the performance of DNNs.

The main focus of the paper is the development of improved variability-aware training (VAT) methods and DNN network-structure innovation that mitigate such degradation, specifically, in the context of highly-quantized deployments. We describe a training approach for the joint quantization-aware and variability-aware training (QAVAT). Training quantized neural networks involves finding the derivative of a piecewise constant (threshold) function whose gradient vanishes almost everywhere. Similar to prior QAT works [11]–[14], QAVAT uses the straight-through estimator to back-propagate gradients through the hard threshold function as though it were an identity function. To train robust networks against physical variability, the practical strategies explored by most researchers use implicit robustification. These techniques inject noise during the forward pass of the backpropagation algorithm to simulate the effect of parameter variation [2], [3], [16]. In every forward propagation, a noise vector is sampled from a pre-defined distribution and applied to the parameters to calculate loss.

Our training algorithm also relies on implicit robustification through injection of variability during training. However, we advance the state-of-the-art in several aspects. Prior work has not addressed the question of an optimal sampling strategy. We describe a sampling mechanism ensuring that the gradient estimates are not biased and have reduced variance, using the

978-3-9819263-6-1/DATE22/©2022 EDAA

techniques of reparameterization and multi-sampling.

Further, we point out that earlier work in VAT has not made a critical distinction in its treatment of variability structure. It assumed independence of variability, failing to capture its spatial structure. In real manufacturing, chip-to-chip variation is typically responsible for a sizable contribution to overall variability. We show that in the presence of spatially correlated variability, training-time only solutions fail to prevent large DNN accuracy loss. To effectively deal with correlated variability, we introduce a *self-tuning DNN architecture* realized via the insertion of light-weight self-tuning modules into DNN models. Our self-tuning DNN architecture is general and different from circuit-specific solutions [17], [18].

In summary, the contributions of this work are:

- A novel implementation of joint quantization- and variability-aware DNN training (QAVAT). It significantly outperforms variability-oblivious QAT and post-training quantized VAT (PTQ-VAT). For ternary-weight ResNet-18 on CIFAR-100, on average, QAVAT achieves 42% and 14% higher accuracy than PTQ-VAT and QAT.
- For realistic variability with spatial correlations, training is unable to prevent large accuracy loss, e.g. of 50% for ResNet-18/CIFAR-100.
- A general self-tuning architecture, suitable for an arbitary DNN, that dynamically adjusts its layer outputs during inference and is very effective in reducing accuracy loss under correlated variability (from 50% to below 10% for ResNet-18/CIFAR-100).
- A design space exploration of self-tuning including its size-quality trade-off.

II. Quantization- and Variation-Aware Training: Algorithm and Models

A. Quantization- and Variation-Aware Gradient Estimation

Our algorithm utilizes Monte Carlo-style injection of uncertainty during network training as a way of making the resulting network robust against physical variations. The Straight-Through Estimator (STE) is used jointly with uncertainty sampling to estimate the descending direction. As a way to formalize key aspects of such training, we introduce a variational computational graph to describe the computations of QAVAT, Fig. 1. The main innovative component of the graph is its specification of the variability sampling process.

The key aspect of the sampling-based gradient-descent algorithm is the injection of variability. It is important to ensure that the gradients computed for weight updates are unbiased with respect to variability sampling. Formally, we seek a graph such that the gradient (g) computed on each batch of training samples satisfies $\mathbb{E}(g) = \nabla_w \mathbb{E}_{x,y,\delta_w}[L(M_{w+\delta_w}(x), y)]$, where $M_w(\cdot)$ represents an arbitrary DNN model parameterized by w, L is the loss function, and the distribution of loss depends on data x, labels y, and weight variation δ_w. Note that the distribution of δ_w usually depends on w.

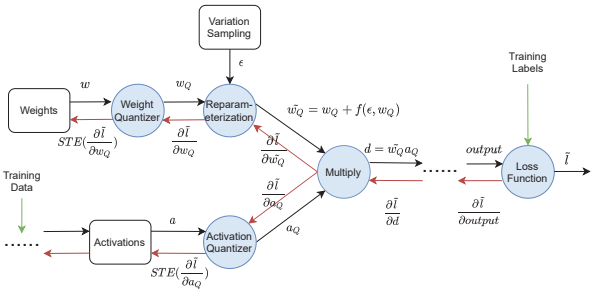


Fig. 1: Computational graph of joint quantization- and variation-aware training. Key elements: reparameterization and multi-variation Monte Carlo sampling for variability, STE for quantization. Multi-variation sampling is done by accumulating the gradients of multiple forward (black arrows) and backward (red arrows) passes through the computational graph.

The choice of a sampling distribution for variability is critical for computing an unbiased estimate of the gradient. When using an auto-differentiation system, the simplest approach is to sample variations δ_w numerically and add them onto weights. Unfortunately, this procedure computes a biased estimator of the true gradients. In Eq. 1, the expected value of the gradient on the left side is what the procedure yields in backpropagation. However, it does not capture the impact of w on the distribution of δ_w. The true gradient is on the right side.

$$\mathbb{E}_{x,y,\delta_w}[\nabla_w L(M_{w+\delta_w}(x), y)] \neq \nabla_w \mathbb{E}_{x,y,\delta_w}[L(M_{w+\delta_w}(x), y)] \tag{1}$$

To overcome this problem, we reformulate the computational graph using reparameterization [19], which allows computing an unbiased estimator of true gradient.[1] Reparameterization assumes that there exists a differentiable function f such that $f(w, \epsilon)$ generates the same distribution as δ_w with a new random variable ϵ that is independent of w. As an example, $f(\epsilon, w) = \epsilon w$, with $\epsilon \sim N(0, \sigma^2)$, is a valid reparameterization of $\delta_w \sim N(0, \sigma^2|w|^2)$. With reparameterization, we have:

$$\begin{aligned} &\nabla_w \mathbb{E}_{x,y,\delta_w}[L(M_{w+\delta_w}(x), y)] \\ &= \nabla_w \mathbb{E}_{x,y,\epsilon}[L(M_{w+f(w,\epsilon)}(x), y)] \\ &= \mathbb{E}_{x,y,\epsilon}[\nabla_w L(M_{w+f(w,\epsilon)}(x), y)] \end{aligned} \tag{2}$$

Unlike Eq. 1, in the last step of Eq. 2 the gradient operator can be safely moved inside the expectation because the reparameterized variability ϵ is now independent of w. Therefore, the back-propagated gradient $\nabla_w L(M_{w+f(w,\epsilon)}(x), y)$ is unbiased.

It is also important to compute a low-variance estimator of the gradient. The variance of the estimator is partly determined by how many ϵs in Fig. 1 are sampled per optimization step. Without discussing the importance of using a low-variance estimator, prior work uses a single sample per update [2], [3], [16]. As we show in experiments, a measurable benefit can be achieved when multiple samples are used. We refer to this as multi-sampling.

[1] We have seen no other work on variability-aware training that has described the need for reparameterization.

The injection of variability, as shown in Fig. 1, is applied after the quantization of weights. We now describe our quantization scheme and introduce the use of STE and show how gradients are estimated when variability is present. We use the uniform symmetric quantizer [20] for both activations and weights. Let x represent the tensor to be quantized (weight or activation), k be the bitwidth, and Δ be the scaling factor. Let x_Q be the quantized tensor and x_D be its dequantized value. The quantization strategy is summarized in Eq. 3.

$$\begin{aligned} x_Q &= clip(\lfloor \frac{x}{\Delta} \rceil, -2^{k-1}+1, 2^{k-1}-1) \\ x_D &= x_Q \cdot \Delta \\ \tilde{x_D} &= x_D + f(\epsilon, x_D) \end{aligned} \tag{3}$$

In the above equation, when x stands for weights, f and ϵ are its variability reparameterization. $\tilde{x_D}$ represents the variability-impacted weights after dequantization. When x stands for activations, f is a dummy function that always returns zero because no explicit variability is assumed on activations. $\tilde{x_D}$ in this case is equal to x_D.

We use the minimum mean squared error (MMSE) heuristic [21] to compute the scaling factors of weights. By default, we compute them only at the beginning of training. We found that more frequent updates only improve results marginally. We also use static method for activation quantization in which the scaling factors are fixed during training. Dynamic methods [10], [22], without extensive fine-tuning (e.g., without adding regularization of clipping thresholds in [22]), have shown degraded performance compared to a static estimation scheme.

To compute the gradients of quantized tensors under variability, STE is used:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \tilde{x_D}} \frac{\partial \tilde{x_D}}{\partial x_D} \frac{\partial x_D}{\partial x} \overset{\text{STE}}{\approx} \frac{\partial L}{\partial \tilde{x_D}} (1 + \frac{\partial f(\epsilon, x_D)}{\partial x_D}) \tag{4}$$

L denotes the training loss and x denotes weights and activations. The entire QAVAT is described in Algorithm 1.

Algorithm 1 Multi-Variation Sampling QAVAT

Given
  n: number of variation samples
  w_t: network parameters at step t
  T: number of iterations, η: step size
  $L(w, B)$: loss of network w on mini-batch B
  ∇: gradient estimation using STE
procedure
  for t in $0 : T-1$ **do**
    Sample a mini-batch of training data B
    $l \leftarrow 0$
    for i in $1 : n$ **do**
      Sample ϵ_i and $l \leftarrow l + L(f(w_t, \epsilon_i), B)$
    $w_{t+1} \leftarrow w_t - \eta \nabla_{w_t} l$
  return w_T

B. Joint Within- and Between-Chip Variability Modelling

Prior work in variability-aware training has not addressed an important issue: realistic patterns of variability found in semiconductor manufacturing have pronounced and significant structure. Instead, earlier work assumes that variability is generated by independent zero-mean Gaussian sources [2], [3], [16]. These models account for only one of the prevalent types of variation, commonly referred to as within-chip variation. Within-chip variation, which we denote by ϵ_W, captures the spatially uncorrelated stochastic nature of variability patterns that impact individual memory cells on a given chip.

In reality, there is a significant component of variation that occurs on a chip-to-chip basis (referred to as the between-chip variation). It captures the variability observed between the mean values of all cell characteristics per chip. It is typically responsible for a sizable contribution to the overall variability. The two patterns of variation can be modeled using an additive model. We introduce a fully-correlated Gaussian variable ϵ_B to represent the between-chip variation. In the realistic case, where both types of variations exist, the additive effect of ϵ_W and ϵ_B leads to partially correlated variations of different network parameters. We use σ_W and σ_B to denote the standard deviation of ϵ_W and ϵ_B, respectively. The total variance of variations is $\sigma_{tot}^2 = \sigma_W^2 + \sigma_B^2$.

The sampling of variations is done in their reparameterized variable space. To sample a single variation $\delta_w \in \mathbb{R}^d$ on weights $w \in \mathbb{R}^d$, we construct its reparameterization variable ϵ in the following way. First, a random variable $\epsilon_B \sim N(0, \sigma_B^2)$ is sampled to capture the between-chip variability. Then, d iid random variables $\epsilon_{W,i} \sim N(0, \sigma_W^2)$ are sampled to capture the within-chip variability. The i-th entry of δ_w is calculated as $\delta_{w,i} = f(\epsilon_i, w_i)$, where $\epsilon_i = \epsilon_B + \epsilon_{W,i}$. f is a reparameterization of δ_w.

Following prior work, we model weight variation δ_w as $\delta_w \sim N(0, \sigma(w))$. Further, we explore two models of $\sigma(w)$ [2], [17], both being of practical interest. In the first model [2], the standard deviation is proportional to the magnitude of a weight: $\sigma(w) = \sigma|w|$. We refer to it as the "weight-proportional variance" model. In the second model [17], all variations in the same layer (l-th) share a fixed standard deviation σ^l, which only depends on the largest weight w_{max}^l in that layer: $\sigma(w) = \sigma|w_{max}^l|$. We refer to it as the "layer-fixed variance" model. We use reparameterization $f(\epsilon, w) = \epsilon w$ for the former, and $f(\epsilon, w) = \epsilon w_{max}^l$ for the latter.

III. Self-Tuning DNNs

A. Better Training is Not Enough against Realistic Variation Patterns

In the presence of significant between-chip variation, training-time methods of producing a robust DNN model appear to fail, even when they succeed at rectifying the detrimental effect of within-chip variation (shown later in our experiments).

It is clear that within- and between-chip variations impact the network accuracy differently. We can qualitatively understand the difference in behavior using the following model. For any given chip, between-chip variation shifts all weights and, by extension, all dot-products and all activations, in the same positive/negative direction. This is in contrast to the

impact of within-chip variation where, on any given chip, some weights are shifted positively while others negatively, effectively creating an averaging effect in the computed dot-products/activations.

B. Self-Tuning DNNs

We have developed a novel general method for mitigating the impact of the between-chip variation that achieves robustness via insertion of small modules in networks.[2] *The proposed self-tuning (ST) architecture is suitable for an arbitary DNN and dynamically adjusts its layer outputs during inference.*

The architecture estimates layer-wise activation deviations and corrects for them. Suppose the analog PIM array implements a MVM of inputs $x \in \mathbb{R}^{d_{in}}$ and weight matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$ to produce $y \in \mathbb{R}^{d_{out}}$. Let W_{max} be the largest value in W. We initially let $\sigma_W = 0$ and show that the structure of self-tuning modules depends on the form of variability.

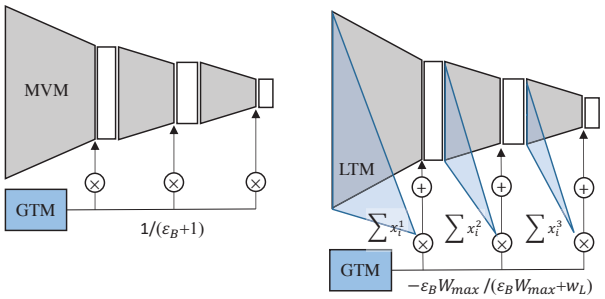


Fig. 2: Self-tuning architectures for variability with weight-proportional variance (left) and layer-fixed variance (right).

If the variation follows the weight-proportional variance model, the noisy components of output are $\tilde{y}_i = \sum_{j=1}^{d_{in}} (W_{i,j} + \epsilon_B W_{i,j}) x_j = (1 + \epsilon_B) y_i$. Thus, the correction term that needs to be applied to every component of y is the same: $1/(1+\epsilon_B)$. We introduce a Global Tuning Module (GTM) to estimate ϵ_B. As shown in Fig. 3, the GTM is a single column of the analog PIM array with n cells. The fixed inputs x_G (voltages) and weights w_G (conductances) are selected based on circuit implementation. The variation-free value of the GTM output is $y_0 = n w_G x_G$ and is stored digitally. The true output of GTM module, under variation, is $y_{GTM} = y_0(1 + \epsilon_B)$. By measuring y_{GTM} and dividing it by y_0 in the digital domain, we get the exact value of ϵ_B. When within-chip variability is present ($\sigma_w \neq 0$), GTM is an unbiased estimator of ϵ_B.

The self-tuning of variations with layer-fixed variance is more costly. In this case, estimating ϵ_B is insufficient to recover each layer's outputs as their error also depends on input activations. Under the layer-fixed variance model, the noisy output is $\tilde{y} = (W + \epsilon_B W_{max}) x$. The error in each output dimension is $(\tilde{y}_i - y_i) = \epsilon_B W_{max} \sum_{j=1}^{d_{in}} x_j$. ϵ_B can be estimated using GTM. W_{max} is stored digitally. The remaining term $\sum_{j=1}^{d_{in}} x_j$ is estimated on a per-layer basis.

[2]Though we limit our formal treatment to fabrication-time between-chip variation, the proposed self-tuning architecture can be generalized to compensate for any correlated weight variation, e.g., due to temperature drifts or aging.

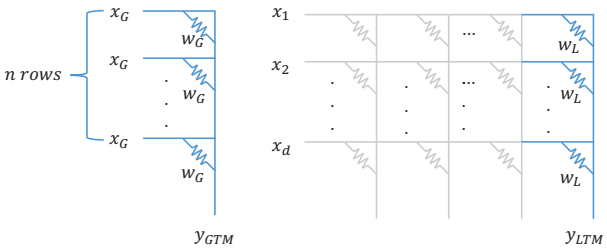


Fig. 3: Circuit implementation of GTM (left) and LTM (right). GTM estimates the model-wise between-chip variation and LTM estimate the layer-wise input activation sums.

To estimate the sum of activations, we introduce a new Layer Tuning Module (LTM) that is added to every layer of the model, Fig. 2 (right). The circuit implementation of LTM is a single column of the PIM memory array with weights (conductances) of all cells set to the same value w_L, Fig. 3 (right). w_L is stored digitally. Under variability, the output of LTM is $y_{LTM} = (w_L + \epsilon_B W_{max}) \sum_{j=1}^{j=d_{in}} x_j$. The tuning is completed in the digital domain where the correction factor $(\epsilon_B W_{max}/(w_L + \epsilon_B W_{max}) \cdot y_{LTM})$ is subtracted from every component of $\tilde{y}$. Allocating more LCM columns further reduces estimation variance and improves results. We use LCM=n to denote the use of LCMs with n columns.

The area overhead of self-tuning architectures is small. LTMs are instantiated per MVM array, adding columns to each array. Assuming a single array of size 512×512 [8], the overhead is 0.2% if LCM=1, and is 3.1% if LCM=16. The overhead from GTM is negligible because only one GTM is needed per chip. The largest number of GTM cells we experiment with is 10^5. This is less than 0.1% of reported demonstrations of analog PIM architectures in hardware [8]. When deployed in a self-tuning mode, the network, without the self-tuning modules, is first trained using QAVAT, with Monte Carlo sampling capturing only the within-chip variation. The self-tuning modules are then appended to the trained model.

IV. Experiments

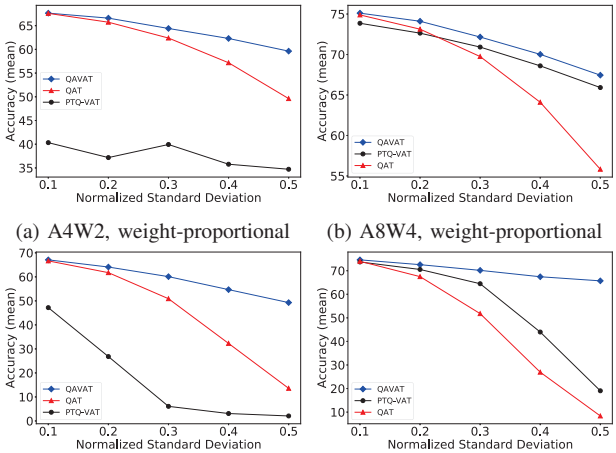
We study the effectiveness of QAVAT on several widely-used models and testsuits: LeNet-5 on MNIST, VGG-11 on CIFAR-10, and ResNet-18 on CIFAR-100. We compare QAVAT with post-training quantized VAT (PTQ-VAT) models and variability-oblivious QAT models. These models are trained under 5 different values of variability standard deviation: $\sigma_W \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The notation "AxWy" represents a model with x-bit activations and y-bit weights. We explore a number of low-bitwidth combinations: A2W2 for LeNet-5, A4W2 and A8W4 for VGG-11 and ResNet-18. By default, we use a single-sample version of QAVAT. The resulting quality (robustness) of each trained model under variability is evaluated using a testing run involving 2000 samples of the variability vectors. The mean test accuracy of the resulting 2000 models is reported. For consistency, the scaling factors of activations in PTQ-VAT are calibrated by the moving average of min-max

values on batches of training data [10]. The scaling factors of weights are computed using MMSE [21].

Experiments are conducted under two scenarios of variability structure. The first scenario represents the case when between-chip variability is negligible and only within-chip variation is present. The second scenario studies a more realistic setting in which both between-chip and within-chip variations exist.

A. Scenario 1: Within-chip Variation is Dominant

This set of experiments assumes within-chip variations only. Table I presents the results at the lowest and the highest level of variation (for the layer-fixed variance model). Fig. 4 shows the complete results of ResNet-18/CIFAR-100 experiment.



(a) A4W2, weight-proportional (b) A8W4, weight-proportional

(c) A4W2, layer-fixed (d) A8W4, layer-fixed

Fig. 4: QAVAT outperforms PTQ-VAT and QAT.

Overall, QAVAT is able to produce models that show only modest degradation as the variation increases and for different bitwidths. In contrast, QAT models degrade substantially at higher variances. This degradation is more severe in higher bitwidth models (A8W4): the models quantized more aggressively appear to be intrinsically more robust against variations. For A8W4, the maximum improvement produced by QAVAT is 24.9% on VGG-11 and 11.6% on ResNet-18, under weight-proportional variance.

		$\sigma = 0.1$			$\sigma = 0.5$		
Model	A/W	VAT	QAT	QAVAT	VAT	QAT	QAVAT
ResNet-18	4/2	47.18	66.65	67.08	2.08	13.58	49.28
	8/4	73.71	74.00	74.61	19.05	8.37	65.70
VGG-11	4/2	53.76	87.10	87.21	29.72	68.36	79.65
	8/4	88.91	88.42	89.00	77.70	37.88	83.09
LeNet-5	2/2	62.75	98.21	98.33	53.82	90.03	96.38

TABLE I: QAVAT at the lowest and the highest level of variability (within-chip variability, layer-fixed variance variance).

Models produced by PTQ-VAT exhibit significant quality degradation at low bitwidths. Across all variability levels, the advantage of QAVAT over PTQ-VAT is much larger for A4W2. For A4W2, on average, across 5 variation levels, QAVAT produces models that are better than PTQ-VAT models by 21.8% on VGG-11 and by 24.9% on ResNet-18, under weight-proportional variance.

We also study the impact of multi-sampling in QAVAT, Fig. 7a. Under both A8W4 and A4W2 configurations, mean accuracy of model is improved (by $\sim 0.9\%$ at $\sigma = 0.3$ and by $\sim 1.3\%$ at $\sigma = 0.5$). The gains saturate around 5 samples.

B. Scenario 2: Equal Within-chip and Between-chip Variations

We next study a more realistic setting in which both between-chip and within-chip components of variation are of equal magnitude: $\sigma_B = \sigma_W$. We call this the "mixed-type" variation. In this case, QAVAT is no longer able to produce robust models, as shown in Fig. 5.

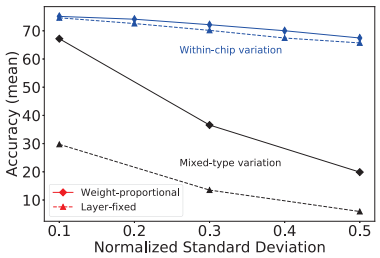
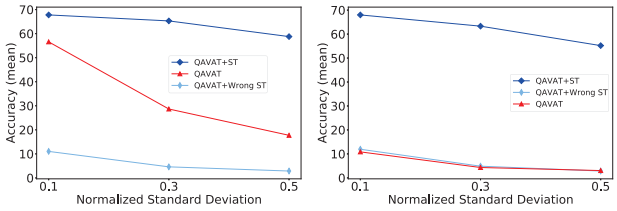


Fig. 5: ResNet-18 on CIFAR-100: QAVAT under two scenarios: (1) within-chip variation only, and (2) mixed-type variation. On both variability models (layer-fixed and weight-proportional), degradation in (2) is much more destructive.

The self-tuning structure is extremely effective in producing deployable models of much higher accuracy. Under $\sigma_{tot} = 0.5$ (weight-proportional variance), the best QAVAT-trained models has 36% accuracy loss on VGG-11 and 54% accuracy loss on ResNet-18 compared to a variation-free case. Under the same setting, self-tuning reduces the accuracy loss on VGG-11 and ResNet-18 to 9.1% and 9.3%, respectively.



(a) Weight-proportional (b) Layer-fixed

Fig. 6: Proper application of self-tuning (ST) prevents quality loss: A4W2 ResNet-18, mixed-type variation.

Fig. 6 shows the performance of a self-tuning architecture for A4W2 ResNet-18. Table II present the results for A8W4 models. By default, QAVAT+ST uses 10^3 GTM cells (per model/chip) and 1 LTM column (per layer). For the layer-fixed variance model at $\sigma_{tot} = 0.3, 0.5$, QAVAT+ST uses 10^5 GTM cells and 16 LTM columns to achieve the results shown. As presented in Fig. 2, the layer-fixed variance requires the GTM+LTM configuration of ST, while the weight-proportional

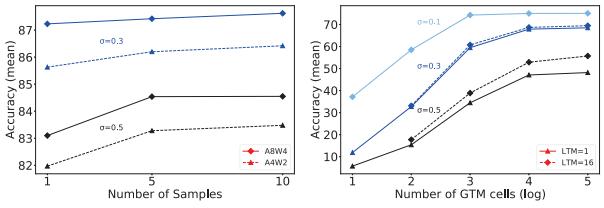
variance requires only a GTM. We show that applying the wrong type of self-tuning, i.e., using a GTM-only ST for layer-fixed variance, and vice-versa, is not helpful and leads to drastic degradation ("QAVAT+Wrong ST" in the figure and table).

The computation overhead of self-tuning is the ratio of inference-time FLOPs in all tuning modules (GTM+LTMs) compared to the base model. On ResNet-18, assuming 10^5 GTM cells are used, the overhead is $\sim 0.3\%$ when LTM=1, $\sim 2.2\%$ when LTM=8, and $\sim 4.4\%$ when LTM=16.

	VGG-11			ResNet-18		
σ_{tot}	0.1	0.3	0.5	0.1	0.3	0.5
QAVAT	88.59	70.75	54.70	67.19	36.58	19.89
QAVAT+ST	90.05	88.09	81.90	75.35	73.39	66.58
QAVAT+Wrong ST	44.70	23.06	17.33	14.32	5.26	3.78

TABLE II: Proper application of self-tuning (ST) prevents quality loss: mixed-type variation, weight-proportional variance.

Performance of self-tuning improves with increasing number of cells in GTM, Fig. 7b (layer-fixed variance). However, there is a diminishing return in model improvement. Larger variance requires a higher number of GTM cells before improvements diminish. Similar trade-off works for variability with weight-proportional variance. For self-tuning of variability with layer-fixed variance, increasing the number of LTM columns is an orthogonal way of reducing estimation variance and improving results. As shown in the figure, the usefulness of more LTMs is clear under the highest variance level ($\sigma = 0.5$).



(a) Impact of multi-sampling (VGG-11, within-chip variation). (b) Impact of ST size (ResNet-18, mixed-type variation).

Fig. 7: Impact of QAVAT and ST parameters on model quality.

V. Conclusion

We present first a general variability- and quantization-aware training algorithm (QAVAT) to mitigate the degradation of DNN performance under analog PIM non-idealities. QAVAT is more effective at reducing the degradation than alternatives on three computer vision networks and datasets and a wide range of variability and quantization levels. We also demonstrate a novel lightweight self-tuning architecture that can be used on arbitrary DNNs. It dynamically adjusts layers' outputs to dramatically reduce degradation under correlated variations.

References

[1] S Cosemans, B Verhoef, J Doevenspeck, IA Papistas, F Catthoor, P Debacker, A Mallik, and D Verkest. Towards 10000tops/w dnn inference with analog in-memory computing–a circuit blueprint, device options and requirements. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 22–2. IEEE, 2019.

[2] Yun Long, Xueyuan She, and S. Mukhopadhyay. Design of reliable dnn accelerator with un-reliable reram. *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1769–1774, 2019.

[3] G. Charan, Abinash Mohanty, Xiaocong Du, Gokul Krishnan, R. Joshi, and Y. Cao. Accurate inference with inaccurate rram devices: A joint algorithm-design solution. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6:27–35, 2020.

[4] Myeonggu Kang, Hyeonuk Kim, Hyein Shin, Jaehyeong Sim, Kyeonghan Kim, and L. Kim. S-flash: A nand flash-based deep neural network accelerator exploiting bit-level sparsity. *IEEE Transactions on Computers*, pages 1–1, 2021.

[5] Linfeng Tao, Rui Xu, Teng Tian, Zikun Xiang, Yifei Li, X. Jin, Jun Ren, Z. Li, and Chenxia Li. Cint – an energy-efficient mixed-signal in-memory cnn accelerator based on nor flash memory (poster). *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, 2019.

[6] Hongjie Wang, Yang Zhao, Chaojian Li, Yue Wang, and Yingyan Lin. A new mram-based process in-memory accelerator for efficient neural network training with floating point precision. *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020.

[7] Ameya D. Patil, Haocheng Hua, Sujan Kumar Gonugondla, Mingu Kang, and Naresh R Shanbhag. An mram-based deep in-memory architecture for deep neural networks. *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2019.

[8] I. Chakraborty, M. Ali, Aayush Ankit, Shubham Jain, Sourjya Roy, S. Sridharan, Amogh Agrawal, A. Raghunathan, and K. Roy. Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges. *Proceedings of the IEEE*, 108:2276–2310, 2020.

[9] tom'sHARDWARE. Toshiba's talk on 5-bit-per-cell flash, 2019.

[10] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv*, abs/1806.08342, 2018.

[11] M. Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[12] Fengfu Li and Bin Liu. Ternary weight networks. *ArXiv*, abs/1605.04711, 2016.

[13] Matthieu Courbariaux, Yoshua Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[14] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. In *BMVC*, 2019.

[15] Brian E Stine, Duane S Boning, and James E Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):24–41, 1997.

[16] C. Zhou, Prad Kadambi, Matthew Mattina, and P. N. Whatmough. Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation. *ArXiv*, abs/2001.04974, 2020.

[17] V. Joshi, M. Le Gallo, Simon Haefeli, I. Boybat, S. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou. Accurate deep neural network inference using computational phase-change memory. *Nature Communications*, 11, 2020.

[18] M. Le Gallo, A. Sebastian, G. Cherubini, H. Giefers, and E. Eleftheriou. Compressed sensing with approximate message passing using in-memory computing. *IEEE Transactions on Electron Devices*, 65:4304–4312, 2018.

[19] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[20] Steve Dai, Rangharajan Venkatesan, Haoxing Ren, B. Zimmer, W. Dally, and B. Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *ArXiv*, abs/2102.04503, 2021.

[21] Yoni Choukroun, Eli Kravchik, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018, 2019.

[22] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, P. Chuang, V. Srinivasan, and K. Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *ArXiv*, abs/1805.06085, 2018.