Robust Processing-In-Memory With Multibit ReRAM Using Hessian-Driven Mixed-Precision Computation

Saurabh Dash, Yandong Luo, Anni Lu, Shimeng Yu, *Senior Member, IEEE*, and Saibal Mukhopadhyay, *Fellow, IEEE*

***Abstract*—This article presents an algorithmic approach to design reliable deep neural networks (DNNs) in the presence of stochastic variations in the network parameters induced by process variations in the bit cells in a processing-in-memory (PIM) architecture. We propose and derive a Hessian-based sensitivity metric that can be computed without computing or storing the full Hessian to identify and protect the "important" network parameters while allowing large variations in unprotected parameters. We also show that this metric can be used to aggressively quantize unprotected network parameters in the PIM for improved inference efficiency and compute density. Experiments on modern DNNs like ResNet, MobileNetv2, and DenseNet on CIFAR10 using measured RRAM device data shows the effectiveness of our approach.**

***Index Terms*—Deep learning, processing-in-memory (PIM), robustness, variation.**

I. Introduction

MIXED-SIGNAL vector-matrix-multiplication (VMM) using processing-in-memory (PIM), in particular using nonvolatile memory devices, such as resistive random access memory (ReRAM) or ferroelectric FET (FeFET), have shown potential for high energy efficiency in deep neural network (DNN) inference [1]–[5]. However, a key challenge is that the process variations in the memory bit cells degrades accuracy of PIM-based DNN accelerators. Hence, in recent years there have been effort to design DNN models that are more robust to variations in the parameters. For example, Long *et al.* [6] has shown that, including ReRAM variation as noise during training a DNN can improve its robustness. However, training-based approaches are effective only when noise introduced during training closely matches the noise experienced during inference. Hence, techniques that can improve robustness without retraining based on exact prior knowledge of variation is desired for PIM.

Manuscript received November 26, 2020; revised February 6, 2021 and March 18, 2021; accepted April 25, 2021. Date of publication May 7, 2021; date of current version March 21, 2022. This work was supported in part by the E2CDA Program of National Science Foundation under Grant CCF 1740197, and in part by the Semiconductor Research Corporation under Grant NCORE 2762.003. This article was recommended by Associate Editor Y. Wang. *(Corresponding author: Saurabh Dash.)*

The authors are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: saurabhdash@gatech.edu).

Digital Object Identifier 10.1109/TCAD.2021.3078408

With the advent of specialized hardware for efficient DNN inference, there has been interest to use mixed-precision computation for increased energy efficiency [7]. It has been shown that different DNN layers have different redundancy and by preferentially computing some layers at higher precision, there is significant gains in efficiency with minimal degradation in classification accuracy. However, these methods seek to only optimize for mixed-precision computation at the resolution of layers which leads to an important dilemma—if there are only a few critically important parameters in a given layer, the entire layer still needs to computed at a higher precision resulting in diminishing gains. We instead argue that if one can identify only the important parameters in a model to computed preferentially at a higher cost with digital multiply and accumulate (MAC) operations, while the rest of the model is aggressively quantized and computed in an unreliable yet significantly computationally efficient PIM architecture, one can obtain significant gains in DNN inference efficiency. Similar ideas have been pursued in the domain of digital signal processing where Karakonstantis *et al.*, have demonstrated that low frequency components in the DCT of an image are more significant compared to the high frequency components and thus can be computed preferentially at the cost of higher power dissipation [8].

Such an aggressive quantization scheme in the PIM is also motivated by the recent advances in multibit NVM devices [9], [10] that enable storage and computation of multi-bit parameters in a single device to increase the computational density of the PIM architecture—leading to a larger number of VMM operations being performed in a PIM.

As modern DNN models are heavily over parameterized, "protecting" a small fraction of the important network parameters—i.e., shielding them from variation and computing them at a higher precision can significantly improve robustness against process variation while leading to high energy efficiency. Such unequal protection of DNN weights have been explored for quantization and compression of weights [11], [12]. These methods primarily used heuristics based on gradients as an indicator to determine which weights must be protected. However, the gradient-based heuristic requires protection of a relatively large number of weights to minimize accuracy loss under appreciable variations.

In this article, we present a significance-driven unequal error and precision protection for hardware-mapped DNNs,

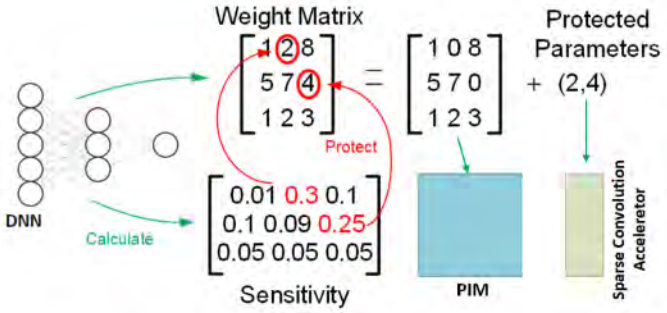

Fig. 1. Overview of the proposed approach.

where Hessian of the weight matrix, instead of the gradient, is used to select a set of critical parameters. The noncritical parameters are aggressively quantized and mapped in the unreliable memory array to perform (error-prone) mixed-signal VMM while accurate digital MAC operations are performed on the critical parameters (Fig. 1) at a higher precision. Although, Hessian-based approaches have been used for pruning DNNs [13], [14], they require storing and computing on Hessian of a DNN (or layer) which is computationally intractable for large (1 million or more parameter) networks. In contrast, we eliminate calculation, storage and inversion of the full exact Hessian and perform one-shot estimation of sensitivities to provide an upper bound on the accuracy degradation given the ability to protect a certain fraction of parameters. This article does not present a new PIM accelerator, rather demonstrates a new method for selection of critical (protected) parameters to be shielded from variation, computed at higher precision, and presents a strong case for a Hybrid PIM architecture by performing a high-level analysis of the energy-consumption through detailed simulations and architectural discussions.

We make the following key contributions.

1) We propose and derive a Hessian-based sensitivity metric for each parameter of a DNN model to determine critical/noncritical parameters. This sensitivity metric is agnostic to device variation.
2) We develop an algorithm to perform sensitivity estimation and parameter selection without computing or storing the exact Hessian of large networks.
3) We demonstrate the validity of the proposed method in the presence of actual measured device noise (for high and low write cycles) for both single-bit and multibit devices.
4) We show that the obtained sensitivity metric can be used to aggressively quantize the noncritical parameters for improved DNN inference efficiency and compute density.
5) We include architectural discussions to demonstrate the feasibility of such a Hybrid PIM design.

We perform algorithm-level experiments on image classification on state-of-the-art modern deep networks like ResNet [15], MobileNet [16], and DenseNet [17] on CIFAR10 [18] to demonstrate the effectiveness of our approach. We observe that, by only protecting 14%–17% of the network parameters there is $<$ 2% degradation in the classification accuracy even with binarization of unimportant parameters. In particular, our approach shows significant improvement over baselines, namely, gradient-based protection [11]. We evaluate overheads of Hessian-driven parameter protection considering existing PIM and digital accelerators for sparse convolution [19], [20]. The analysis shows 1.33× and 1.28× improvement in power-efficiency compared to a pure PIM Implementation for ResNet18 and DenseNet121, respectively. We also observe an increase in compute density by 2.64×. We believe this opens up a new way of looking at energy efficient DNN inference—by using noisy low precision hardware for bulk DNN computation augmented by precise high precision MAC operations is sometimes a better strategy for energy efficient inference.

II. Background

A. Device Variation and PIM-Based VMM

A generic mixed-signal PIM architecture for VMM uses a bit-cell array to store the parameters where each parameter is stored as multibit number [4], [6], [20]–[23]. The bit cells can be realized in many technologies, including ReRAM, FeFET, or SRAM. The input bits are fed in serially via rows (eliminates digital-to-analog converter, DAC). The bitline current summation and digitization (ADC), followed by shift-and-add operation is used to perform MAC. The stochasticity in the device conductance can cause current variations in bit cells. For example, conductance variation in can be due to stochasticity in filament formation in ReRAM [6], [24], [25], threshold voltage variation of access transistors in SRAM [26]–[28], or stochasticity in the threshold voltage for FeFET [29]. Ultimately conductance variation leads to inaccurate VMM operation [6]. Long *et al.* [6] showed that for a multibit PIM, the effect of Gaussian distribution in conductance of individual devices can be approximately modeled by adding a Gaussian noise with standard deviation proportional to the value of the parameter ($x_i \sim \mathcal{N}(0, \sigma w_i)$, $\mathbb{E}[x_i x_j] = 0$). The standard deviation of the parameter noise (σw_i) can be determined from the process variation in individual bitcells.

B. Intuition Behind Hessian-Based Protection

The Hessian represents the rate of change of the jth component of the gradient along w_i. It thus gives the amount of curvature of the loss manifold at a given point. The eigenvalues of the Hessian matrix give the amount of curvatures along the direction of the eigenvectors. Previous works have shown that for overparameterized networks, the bulk of the eigenvalues are close to 0 with only a few outliers, indicating that there are only a few directions of large curvatures [30]. Nguyen [31] has also shown that for a class of such overparameterized networks, all the minima lie in a large global valley. We argue that for such models, if variation in parameters is constrained along this valley by minimizing the perturbation along the directions of the large curvatures, we can reduce the degradation in the classification accuracy (Fig. 2).

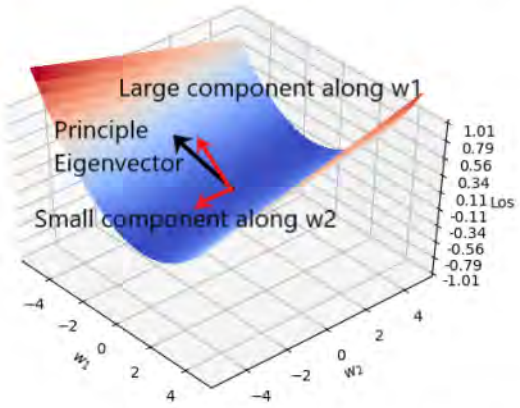

Fig. 2. Example loss surface showing the intuition behind Hessian-based protection.

C. Related Work on Hessian-Based Pruning

We are motivated by prior works like OBS [13] and its extension L-OBS [14] that have used the Hessian to prune DNNs. Given a trained network with final parameters $\boldsymbol{w}$, OBS uses the Taylor expansion of the error $\mathcal{L}$ with respect to $\boldsymbol{w}$ of to obtain

$$\delta\mathcal{L} \approx \nabla_{\boldsymbol{w}}\mathcal{L}\,\delta\boldsymbol{w}^T + \frac{1}{2}\delta\boldsymbol{w}^T\mathbf{H}\delta\boldsymbol{w} \tag{1}$$

where the Hessian ($\mathbf{H} \in \mathbb{R}^{N\times N}$) with respect to the weights ($\boldsymbol{w} \in \mathbb{R}^N$) is defined as

$$H_{ij} = \partial_{w_i}\partial_{w_j}\mathcal{L}(\boldsymbol{w}). \tag{2}$$

The goal is to set one parameter (w_q) to 0 iteratively, that minimizes $\delta\mathcal{L}$ in each pruning iteration. For a trained network, the first order term vanishes resulting in an equivalent optimization problem as

$$\min_q \delta\boldsymbol{w}^T\mathbf{H}\delta\boldsymbol{w} \quad \text{s.t} \quad \mathbf{e}_q^T\delta\boldsymbol{w} + w_q = 0 \tag{3}$$

$\mathbf{e}_q$ is a unit vector whose qth element is 1 otherwise 0. The closed form solution to the above optimization problem is given by

$$\delta\mathbf{w} = -\frac{w_q}{[H^{-1}]_{qq}}\mathbf{H}^{-1}.\,\mathbf{e}_q \;\text{ and }\; L_q = \frac{1}{2}\frac{w_q^2}{[H^{-1}]_{qq}} \tag{4}$$

L_q gives the sensitivity of the error with respect to w_q, H^{-1} denotes the inverse of the Hessian. The memory requirement of the Hessian of a network with n parameters grows as $\mathcal{O}(N^2)$. Hence, Hessian of a network with 1 million parameters has 1 trillion parameters making storage and computation H^{-1} computationally intractable.

To circumvent this issue, Dong *et al.* [14] introduced L-OBS which induces sparsity in each layer (l) using OBS to minimize the error (Frobenius norm) between the actual and reconstructed features in layer ($l+1$). The memory complexity of layerwise Hessian inversion is tractable, but the process is computationally intensive for wide fully connected layers due to its $\mathcal{O}(N^3)$ complexity. Furthermore, as pruning in the deeper layers depends on the pruning in the earlier layers, an sequential computation of layers is necessary. This overhead of sequentially processing layers becomes significant as the state-of-the-art networks get deeper and introduce more dependencies between layers with skip connections.

D. Our Contribution

In contrast to OBS [13] and L-OBS [14] where the goal is compute H^{-1} to estimate important parameters, we relax and reformulate the problem to provide an upper bound on the degradation given the ability to protect a certain fraction of the parameters. We show that this allows us to circumvent the calculation, storage and inversion of the full exact Hessian; utilizing space in the order of $\mathcal{O}(N)$ instead of $\mathcal{O}(N^2)$. We are also able to find the sensitivity for all the parameters in the DNN in one-shot unlike L-OBS. In summary, although we build on the concept of basing a sensitivity metric on the Hessian, we present a problem formulation and associated algorithmic approach to obtain good enough but computationally feasible solutions for improving robustness of DNN inference considering stochastic variations in parameters.

III. Proposed Methodology

A. Problem Formulation

Given a trained DNN $\boldsymbol{w}$, consider a binary mask M ($M \in \{0,1\}^N$) of that network where 1 and 0 represents protected and unprotected parameters, respectively. Given an acceptable expected degradation Δ_{acc}, ideally we would like to find a M that has the minimum number of entries that are 1. This can be formulated as the optimization problem

$$\arg\min_M \;\; \|M\|_0 \quad \text{s.t} \;\; \Delta \leq \Delta_{\text{acc}}. \tag{5}$$

Modifying OBS, one can solve this problem by iteratively finding out a parameter w_q

$$\arg\min_q \;\; \mathbb{E}\left[\boldsymbol{w}^T\mathbf{H}\delta\boldsymbol{w}\right] \quad \text{s.t.} \;\; \mathbb{E}\left[x_q^2\right] = 0 \tag{6}$$

where x_q is the variation in the parameter w_q. After a certain number of iterations, the degradation Δ is less than the tolerable degradation Δ_{acc}, resulting in the final mask M. However, unlike pruning that minimizes a deterministic objective, the objective function here is the expected value of a stochastic variable making the optimization problem harder to solve.

Instead of finding M that minimizes $\|M\|_0$, we relax the problem to find an M which has $\|M\|_0$ less than some acceptable value p_{acc}

$$M \quad \text{s.t} \;\; \Delta \leq \Delta_{\text{acc}} \quad \text{and} \quad \|M\|_0 \leq p_{\text{acc}}. \tag{7}$$

We show that the degradation Δ is bounded by a function of a sensitivity vector (s) that can be calculated for the entire DNN network at once and $\|M\|_0$. This bound can be made tighter to make $\Delta \leq \Delta_{\text{acc}}$ by protecting the parameters corresponding to the largest values of s.

B. Mathematical Motivation

Let us consider the loss $\mathcal{L}(\boldsymbol{w})$ where $\boldsymbol{w} \in \mathbb{R}^N$ are the model parameters. N is the total dimension of the parameter space. Let the small variation in parameters around this point be given by $\Delta\boldsymbol{w} = \alpha\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon}$ is a random noise vector where the standard

deviation of the noise is proportional to the parameter ($\epsilon = w \odot x$, $x \sim \mathcal{N}(0, I_{N\times N})$). $|\cdot|$ is the absolute value function. $x \odot y$ represents the Hadamard product of two vectors x, y and $x^2 = x \odot x$. Using Taylor's expansion and neglecting the third order and higher terms

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \approx \mathbb{E}\left[\left|\alpha\epsilon^T \nabla_w \mathcal{L}(w) + \frac{1}{2}\alpha^2 \epsilon^T \mathbf{H}\epsilon\right|\right]. \tag{8}$$

As w is a minima, $\nabla_w \mathcal{L}(w) = 0$ and $\mathbf{H}$ can be written as $Q^T \Lambda Q$, since $\mathbf{H}$ is a Real Symmetric Matrix, where the rows of Q are the eigenvectors and Λ is a diagonal matrix with the eigenvalues as the diagonal entries in descending order

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \approx \mathbb{E}\left[\left|\frac{1}{2}\alpha^2 \epsilon^T Q^T \Lambda Q \epsilon\right|\right] \tag{9}$$

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \approx \mathbb{E}\left[\frac{1}{2}\alpha^2 |(Q\epsilon)^T \Lambda (Q\epsilon)|\right]. \tag{10}$$

Let $\tilde{\epsilon} = Q\epsilon$, then

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \approx \frac{1}{2}\alpha^2 \mathbb{E}\left[\left|\tilde{\epsilon}^T \Lambda \tilde{\epsilon}\right|\right]. \tag{11}$$

As Λ is a diagonal matrix, (11) can be written as

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \approx \frac{\alpha^2}{2}\mathbb{E}\left[\left|\sum_{i=1}^{N} \tilde{\epsilon}_i^2 \lambda_i\right|\right] \tag{12}$$

where $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_N$. Using the triangle inequality

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2}\sum_{i=1}^{N} |\lambda_i| \, \mathbb{E}\left[\tilde{\epsilon}_i^2\right]. \tag{13}$$

Since, $\tilde{\epsilon} = Q\epsilon$ and the rows of Q are eigenvectors (q_i^T) of H^w, $\tilde{\epsilon}_i = q_i^T \epsilon$. Thus (13) can be written as

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2}\sum_{i=1}^{N} |\lambda_i| \, \mathbb{E}\left[\left(q_i^T \epsilon\right)^2\right]. \tag{14}$$

Expanding the inner product

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2}\sum_{i=1}^{N} |\lambda_i| \, \mathbb{E}\left[\sum_j \left(q_{ij}^T \epsilon_j\right)^2\right] + \sum_j \sum_k q_{ij}^T q_{ik}^T \, \mathbb{E}[\epsilon_j \epsilon_k]. \tag{15}$$

As, $\mathbb{E}[\epsilon_j \epsilon_k] = 0$ the above equation can be rewritten as

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2}\sum_{i=1}^{N} |\lambda_i| \left(q_i^2\right)^T \mathbb{E}\left[\epsilon^2\right]. \tag{16}$$

After the end of training, bulk of the eigenvalues are close to 0, only a few outliers remain [30] ($\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n \ggg \lambda_{n+1} \cdots \lambda_N$), where $n \ll N$. Since, ϵ is proportional to the parameters, (16) can be written as

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2}\left(\left(\sum_{i=1}^{n} |\lambda_i| \left(q_i^2\right)^T\right) \odot \left(w^2\right)^T\right) \times \mathbb{E}\left[x^2\right]. \tag{17}$$

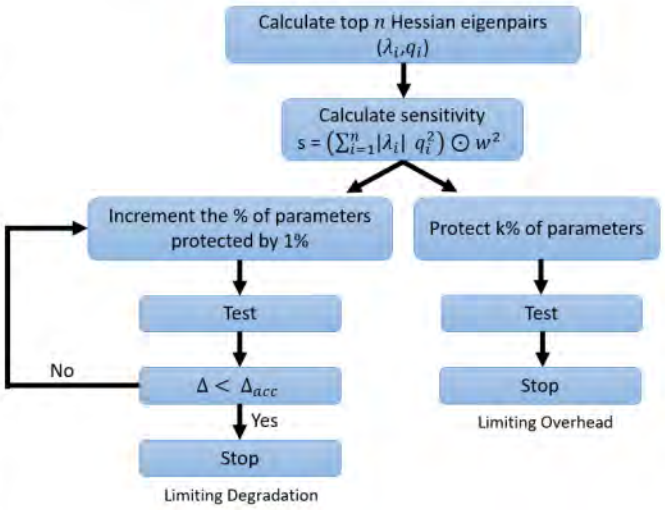

Fig. 3. Overall Hessian-based protection flow.

Denoting, $\sum_{i=1}^{n} |\lambda_i| \, (q_i^2)^T \odot (w^2)^T \equiv s^T$, $s \in \mathbb{R}^N$, we obtain

$$\mathbb{E}[|\mathcal{L}(w+\Delta w) - \mathcal{L}(w)|] \leq \frac{\alpha^2}{2} s^T \, \mathbb{E}\left[x^2\right]. \tag{18}$$

As evident from (18), if the variance of noise along the larger components for s is 0, there is minimal degradation in the loss.

Quantization can be interpreted to be a form of noise added to the true value of the weights. Although we derived the expression with certain assumptions (added noise as 0 mean), which do not hold for quantization noise, we use this metric to preferentially quantize the identified unimportant parameters to a very low precision as well. Through our experiments, we empirically show that our sensitivity metric also performs well in this additional task.

C. Proposed Algorithm

The inputs to this algorithm are a trained DNN parameters w with N parameters, maximum tolerable degradation Δ_{acc} and the precision for unimportant-parameters B. The hyper-parameter in the algorithm is n which represents the number of Hessian eigenpairs used to approximate the sensitivity metric. The output is a binary mask M ($M \in \{0,1\}^N$) whose indices that are 1 denote the parameters that are protected. Algorithm 1 summarizes the steps to identify the "important" parameters and Fig. 3 shows the algorithmic flow. It is important to note that, the proposed approach does not involve retraining. Moreover, the eigenpair computation does not require prior information on the extent of stochastic variation on the parameters nor the precision of quantization performed. In other words, our approach is agnostic to underlying device technology (and its process variation) used in the DNN hardware.

The first step is to compute the n Hessian eigenpairs. Our goal is to compute the n largest eigenpairs without access to the full exact Hessian. A well established method to calculate the principle eigenpair of matrix A is the power iteration

Algorithm 1 Mixed-Precision Protection

Require: Trained DNN Model Parameters w
(Calculate the first n Eigenvalues and Eigenvectors)
for $i = 1$ to n **do**
$(\lambda_i, q_i) = DeflatedPowerIter(w, i)$
end for
(Calculate the Sensitivity Vector s)
$s \leftarrow \mathbf{0}$
for $i = 1$ to n **do**
$s \leftarrow s + |\lambda_i| \, ((q_i^2) \odot (w^2))$
end for
(Find the top k% important weights to protect)
$k \leftarrow 0$
while $\Delta > \Delta_{acc}$ **do**
$k \leftarrow k + 1$
$idx_k \leftarrow argmax(s, k)$
$M(idx_k) \leftarrow 1$
$\Delta \leftarrow Test(w, M, B)$
end while
return M

method that computes the following until convergence:

$$b_{k+1} = \frac{Ab_k}{\|b_k\|} \quad \hat{\lambda} = \frac{\|b_{k+1}\|_\infty}{\|b_k\|_\infty}. \tag{19}$$

Once the principle eigenpair is obtained, Hotelling's deflation can be used to give rise to a matrix A' that has the same eigenpairs as the initial matrix, except the principle eigenpair $(\lambda, \mathbf{q})$

$$A' = A - \lambda \mathbf{q}\mathbf{q}^T. \tag{20}$$

Power iteration followed by deflation can be repeatedly applied to compute the top n eigenpairs [32]. Note, computing the eigenvectors using the above process does not require access to the full Hessian, which is intractable for large networks, rather one needs to only perform computation of an Hessian-vector product. This can be done efficiently using the automatic differentiation engine of PyTorch as shown by Golmant *et al.* [33]. Calculating the Hessian-vector product involves invoking the backpropagation function twice. However, the automatic differentiation engine of PyTorch can only effectively calculate these Hessian-vector products for a minibatch. These minibatch Hessian-vector products are averaged to get an estimate over the entire dataset which is then used for the eigendecomposition. Power iteration method can be used to obtain eigenpairs ϵ close to the true eigenpairs by increasing the number of iterations [32]. This guarantees that our algorithm always converges, and is thus stable.

Once the n eigenpairs are computed, the next step is to calculate the sensitivity metric s, where s_i gives the sensitivity of parameter w_i is calculated using

$$s \equiv \left(\sum_{i=1}^{n} |\lambda_i| \, q_i^2 \right) \odot w^2 \tag{21}$$

λ_i's are the eigenvalues and q_i's are the corresponding eigenvectors. $(\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n \ggg \lambda_{n+1} \cdots \lambda_N)$, where $n \ll N$.

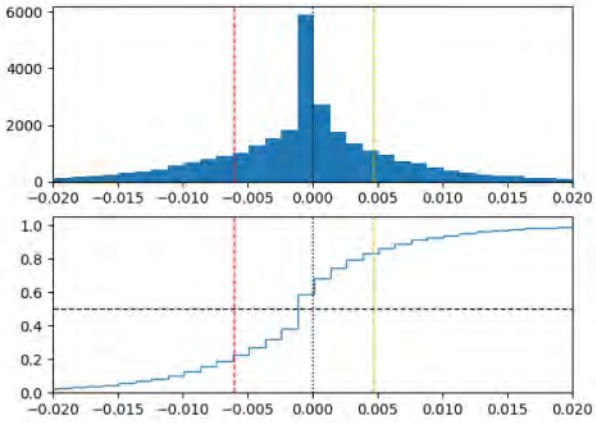

Fig. 4. (a) Distribution of parameters in a MobileNetv2 layer. (b) CDF split across the median (black vertical line). Red line (left) and yellow line (right) show the 1-bit quantized values.

The accuracy of the estimate of s depends on the number of top n eigenpairs computed. If $\lambda_1 \gg \lambda_2$, only the first eigenvector can be used. In this case, s simplifies to just the Hadamard product of the principle eigenvector q_1 and weight vector w.

Once the parameter sensitivities are obtained the next step is to compute the minimum percentage of parameters to be protected to meet acceptable accuracy loss (Δ_{acc}). As the accuracy drop is a monotonous function of the amount of protection, we can employ either iterative search or binary search on the percentage of protected parameters k ($0 \leq k \leq p_{acc}$) to find the minimum percentage of parameters to be protected. In our experiments, we implemented iterative search. The parameters which were protected were also computed at a higher precision, while the rest of the parameters were quantized to a certain precision B. We increased the percentage of parameters protected in increments of 1% until the accuracy drop was within the desired limit Δ_{acc}. Once we find the minimum percentage of protected parameters, we generate the associated binary mask M to indicate the indices of the protected parameters.

D. Aggressive Quantization of Unimportant parameters

In this article we explore binarization and 3-bit quantization of the unimportant parameters stored using a 3-bit ReRAM device [9]. The quantization of important parameters is kept at 8 bit as it has been observed that most networks can be compressed to have 8-bit parameters without substantial degradation compared to the floating-point counterparts. All the activations are also quantized to 8 bit.

1) Binarization: To quantize the unimportant weights to 1 bit, we borrow ideas from rate-distortion theory [34]. Let the distribution of parameters in a layer be given by $p(x)$ as shown in Fig. 4(a). We take a monotone transformation of the distribution (compander) $f(x)$. Here, we choose the compander to be the cumulative distribution function (CDF) of the parameter distribution $p(x)$. We perform quantization by splitting the parameter distribution along the median (black vertical line in Fig. 4) and mapping the mean of all weights smaller than the median (red vertical line in Fig. 4) and mean of all weights

greater than the median (yellow vertical line in Fig. 4) to be the quantized values.

2) 3-bit Quantization: To take advantage of the 3-bit ReRAM device, we perform 3-bit uniform affine quantization as described by Krishnamoorthy [35]. The parameters of each layer are quantized with a uniform scale to one of 2^3 values.

E. Runtime Analysis

The runtime of the proposed approach is associated with computing the eigenpair(s) for the Hessian of the entire dataset. For ResNet18 with 11M parameters on CIFAR10, if the power iteration convergence steps are fixed to be 10, each eigenvector can be calculated in 1 h on 2 NVIDIA GTX1080ti GPUs. In comparison, it takes 4 h for the same network to be trained for 400 epochs from scratch. Although, the time to compute the protected parameters is substantial, however, this process has to be carried out only once after reaching the final trained weights. Moreover, the eigenpair computation in the proposed approach does not need to be repeated depending on underlying device technology.

IV. Simulation Results on Device Data

A. Device Details

The conductance distribution of 3-bit RRAM device is obtained from the experiment data in [9], which is measured from the 256 × 256 1-transistor 1-resistor (1T1R) RRAM array fabricated with Winbond's 90*nm* HfO_2 RRAM process [10]. For the eight states in a 3-bit weight, the state 0 and 7 are mapped to the high resistance state (HRS, 500 kΩ) and low resistance state (LRS7, 6.4 kΩ), respectively. The rest 6 states are uniformly inserted between HRS and LRS7.

To tighten the conductance distribution, a write-verify scheme with different numbers of write-verify cycles is used. In this article, the conductance distributions with single write-verify cycles and multiple write-verify cycles are considered. The number of write cycles in the multiple write-verify cycles case was chosen to be 5. We collected conductance data for various number of write cycles ranging from 1 to 30 and we observed that 5 write cycles ensured a tight distribution as shown in Fig. 6(a). Any additional write cycles beyond this would only lead to an increase in write time and hence decrease in the throughput.

B. Setup and Assumptions

In this section, we present the results for the classification accuracy of a hybrid PIM architecture on modern state-of-the-art deep networks like ResNet18 [15], MobileNetv2 [16], and DenseNet121 [17] on the CIFAR10 dataset. As most of the PIM-designs use reduced-precision fixed-point representation [4], [6], [20]–[23], our simulations are also performed considering quantized weights and activation. Computations of parameters are deemed important are stored and computed at a higher precision of 8 bits. The protected parameters have no variation in the parameters. We aggressively quantize the unimportant parameters to 3 bit/1 bit. We quantize all the networks to have uniform 8-bit fixed-point activation quantization [36]. The batch-normalization layers were quantized

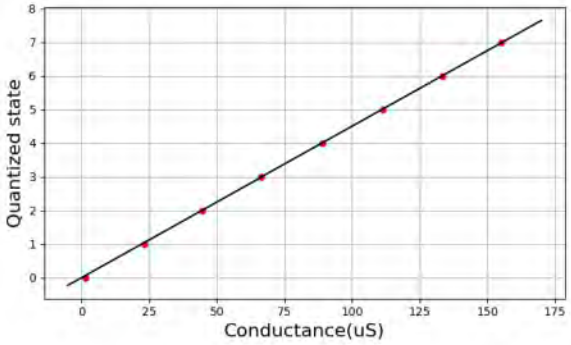

Fig. 5. Mapping from conductance to quantized state.

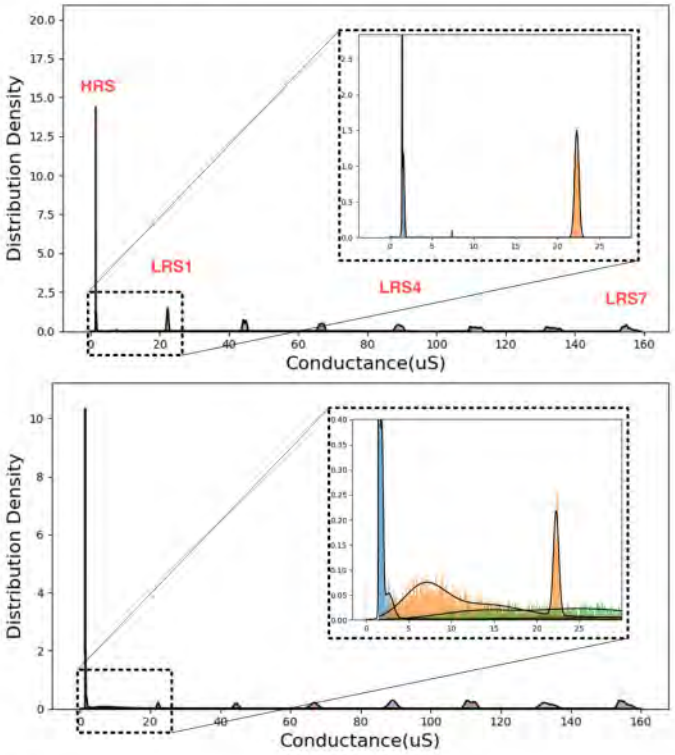

Fig. 6. ReRAM conductance distribution after multiple write cycles (above) and single write cycle (below).

to 12 bits as we observed substantial degradation in the classification accuracy of DenseNet121 when the precision was reduced further. As mentioned before, the unimportant parameters are assumed to be mapped to a PIM leading to variation in the parameters. To model parameter variation using device measurement data, we fit a Gaussian mixture model on the device conductance data to obtain a parameterized conductance distribution that is sampled to obtain conductance values for our simulations (Fig. 6). To map the conductance values to the quantized states, we use linear regression on the modes of the obtained Gaussian mixture model as shown in Fig. 5. Using this linear fit, the variation in the conductance values can be now translated into variation in the quantized network parameters.

Apart from using the device to store 3-bit parameters, we also perform simulations where the device is used as a 1-bit device, i.e., unimportant parameters are quantized to 1-bit and

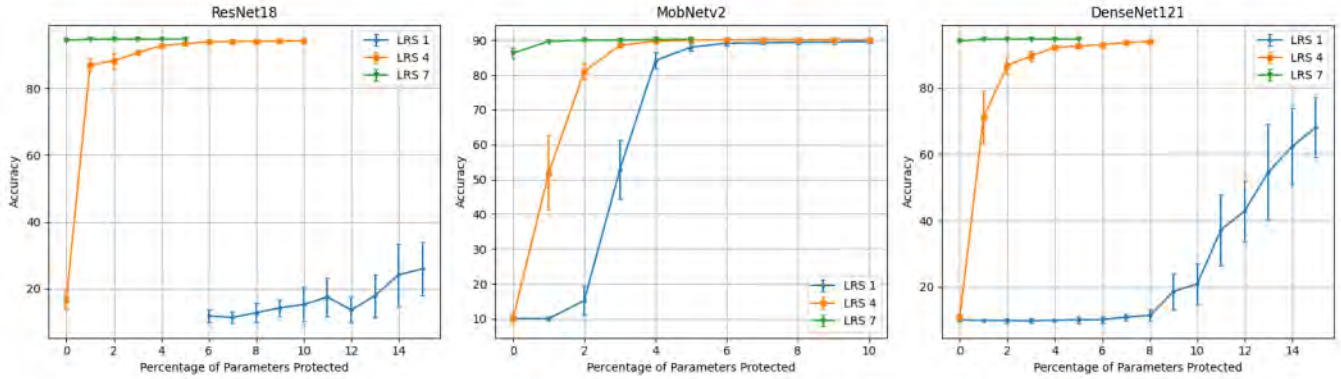

Fig. 7. Classification accuracy with different protection percentages of 8-bit PIM.

mapped to the device. The lowest conductance state [or highest resistance state (HRS)] is used as the "0" state of the parameter while the "1" state of the parameter can be mapped to any of the other seven high conductance states [low resistance states (LRS)]. We perform experiments by mapping the "1" state to the first low conductance (LRS1) state, fourth low conductance state (LRS4) and the highest conductance state (LRS7) denoted as 1 bit(1), 1 bit(4), and 1 bit(7), respectively, further on in the manuscript. It is desirable that the "1" state is mapped to lowest high conductance state of the device to keep the output current minimal—reducing the power dissipation in the PIM. We perform experiments on conductance distributions after a single write cycle and multiple write cycles. The DNN models are implemented using the PyTorch framework [37] and are run on an Nvidia GTX-1080Ti.

C. Robustness of 8-Bit Unprotected Design

For a baseline, we perform simulations to test for the robustness of classification in a PIM with 8-bit parameters and 8-bit activation quantization. Each parameter is stored and computed using 8 1-bit devices. Table I shows the classification accuracy when the "1" state is mapped to LRS1, LRS4 and LRS7 for both single and multiple write cycles. We see that only LRS7 is resistant to variation as the standard deviation is low compared to the mean value of the conductance. PIM using LRS1 and LRS4 suffer from a large drop in classification accuracy for both single cycle and multicycle write routines.

D. Robustness of Proposed Design With 8-Bit PIM

As discussed above, to mitigate the drop in classification accuracy, we protect a small fraction of the parameters from variation. The hyperparameter in our algorithm is the number of eigenpairs n to be considered to approximate the sensitivity vector. In this section, we choose n to be 5. Fig. 7 shows the improvement in classification accuracy for single write cycle. We see that for ResNet18 LRS4 shows a dramatic improvement in classification accuracy when just 1% of parameters are protected as matches the classification accuracy of LRS7 with 6% protection. Similar trends are observed for MobileNetv2 and DenseNet121. LRS1 also shows improvement in classification accuracy despite the presence of large amounts of variation, but at the cost of significantly higher protection.

TABLE I
CLASSIFICATION ACCURACY OF BASELINE 8-BIT PIM

Model	Write Cycles	LRS	Accuracy (%)
ResNet18	Single	1	10.21
		4	16.66
		7	94.41
		4 + 6% protection	94.38
	Multiple	1	10.63
		4	93.03
		7	94.44
		4 + 1% protection	94.43
DenseNet121	Single	1	10.00
		4	10.77
		7	94.17
		4 + 6% protection	94.20
	Multiple	1	10.44
		4	89.24
		7	94.38
		4 + 1% protection	94.42
MobileNetv2	Single	1	10.09
		4	10.16
		7	86.31
		4 + 5% protection	90.09
	Multiple	1	10.06
		4	69.59
		7	89.29
		4 + 3% protection	90.02

E. Robustness of Aggressively Quantized Unprotected Design

To improve the efficiency of the PIM, we truncate the models to have 3-bit parameters and 8-bit activations. We observe the classification accuracy degrade to random guessing when the model is aggressively quantized to have 3-bit parameters after training and device variation is injected during inference, suggesting the need to augment the unreliable PIM with reliable digital hardware to perform critical computations without any variation at a higher precision.

F. Robustness of Proposed Design

We plot the classification accuracy for various percentages of the parameters protected for both the cases—when inference is performed using devices in which the parameters are written using a single write cycle and using multiple write cycles shown in Fig. 8. In this section, we choose the number of eigenpairs to estimate the sensitivity vector, n to be 5.

1) Single Write Cycle: Data written into ReRAM device with a single write cycle suffers for large conductance variation as shown in Fig. 6(bottom). The classification accuracy for

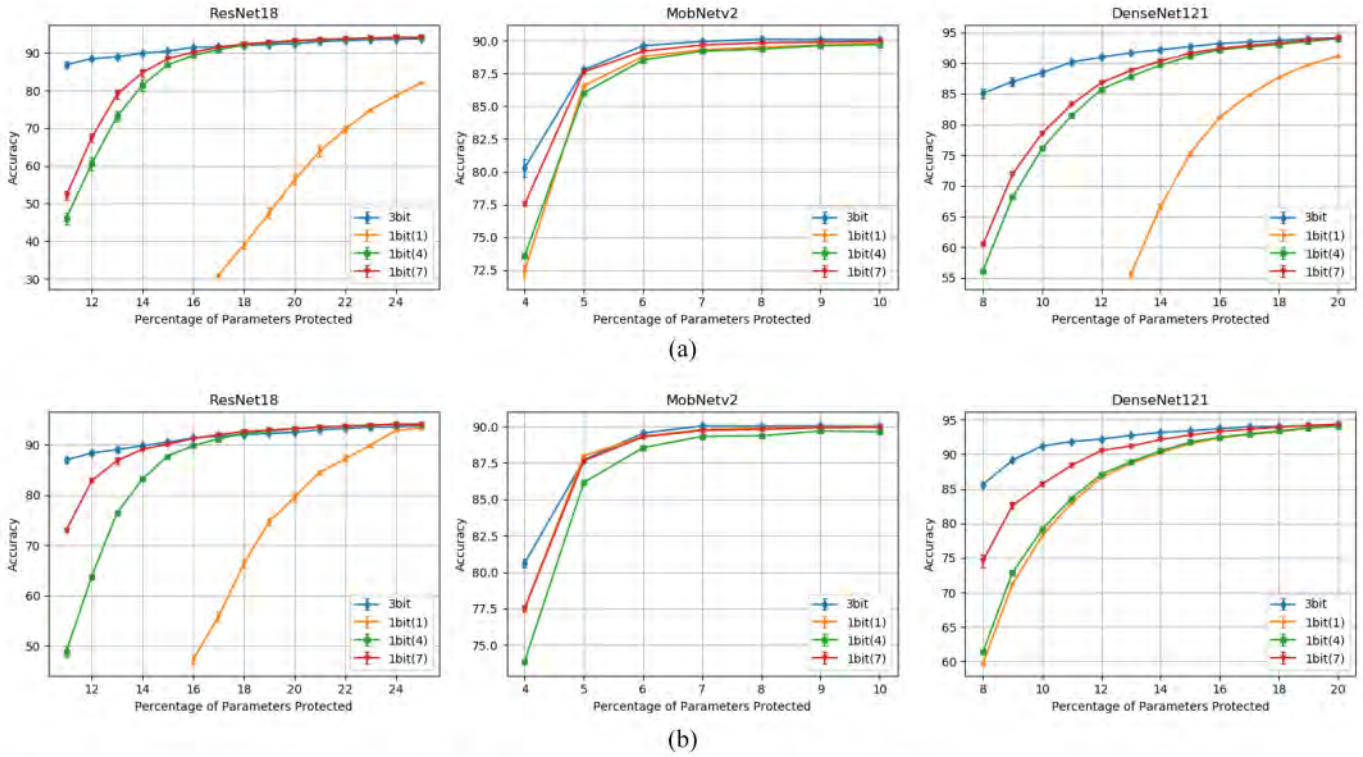

Fig. 8. Classification Accuracy for various models with (a) single write cycle and (b) multiple write cycles.

various models is shown in Fig. 8(a). We can see that models with 3-bit quantized parameters need much lower number of parameters to be protected. For large networks like ResNet18 and DenseNet121, the classification accuracy using the lowest conductance states [1 bit(1)] suffer from a large drop due to the large overlap between conductance states 0 and 1, as shown in Fig. 6(below). The highest and the fourth highest conductance states [1 bit(7) and 1 bit(4)] perform as good as 3-bit parameters when a higher fraction of parameters are protected however, using a 3-bit device allows us to aggressively reduce the fraction of parameters protected, reducing the computational overhead and allowing greater than 90% classification accuracy with 14% and 11% protection for ResNet18 and DenseNet121, respectively. When the fraction of parameters protected are reduced to 11% for ResNet18 and 8% for DenseNet121, 3-bit quantization is able to achieve a classification accuracy of 86.81% and 85.1% compared to 52.13% and 60.38% using 1 bit(7), respectively. For MobileNetv2, the 3-bit parameters also outperform 1-bit counterparts by 1% across the board.

It is desirable to have multiple write cycles to program the ReRAM device, but this is accompanied by an increase in per-layer computation time. This time adds up to be significant in modern DNN architectures that can possibly have hundreds of layers.

2) Multiple Write Cycles: When multiple cycles are used to write data to the ReRAM device, the conductance distributions are tight, as seen in Fig. 6(top) and the variation in conductance values is minimal. Here, unlike the previous case, using the lowest conductance state 1 bit(1) allows for comparable accuracy compared to the 1 and 3 bit counterparts at high protection percentages, however we still observe that 1 bit(1) is very sensitive to the percentage of parameters protected. With the distributions tightening, the gap between the accuracies of 3 bit and 1 bit(7) reduces. Compared to the single write cycle case, the classification accuracy of 1 bit(7) increases by 21% at 11% protection for ResNet18 and by 14% at 8% protection for DenseNet121. Moreover, we see a decrease in the standard deviation of the classification accuracy across the board due to the decrease in parameter variation.

G. Distribution of Sensitivity

We see from Fig. 8 that the number of parameters required to be protected are the least for MobileNetv2, followed by DenseNet121 and ResNet18. This can be explained by plotting the distribution of the sensitivity values as shown in Fig. 9. We see that MobileNetv2 has a small number of parameters with high sensitivity (to the right), once these parameters are protected, the network accuracy increases quickly and is now immune to degradation. On the other hand, the distribution of sensitivity is spread out for DenseNet121 and ResNet18. Thus, a greater number of parameters need to be protected in these networks for robust inference.

H. Effect of Number of Eigenpairs

As the time overhead increases in the order of $\mathcal{O}(n)$, it is desirable to study how the estimate of sensitivity improves with the number of eigenpairs considered. Fig. 10 shows the reduction in the classification accuracy for DenseNet121 when

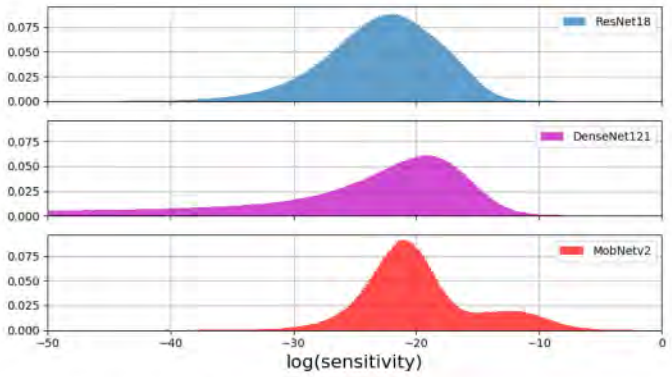

Fig. 9. Distribution of nonzero sensitivity values.

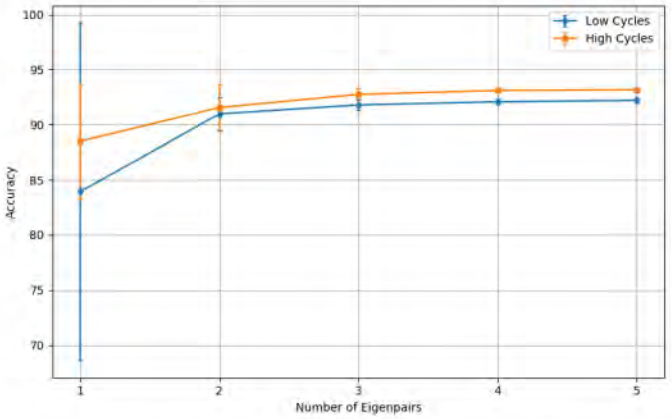

Fig. 10. Classification accuracy for DenseNet121 with different number of eigenpairs considered.

14% parameters are protected to ensure the degradation in mean classification accuracy is $< 2\%$ of the ideal 32-bit floating-point network. We observe that increasing the number of eigenpairs considered to approximate the sensitivity increases the mean classification accuracy, more importantly it leads to a massive decrease in the standard deviation of the classification accuracy which implies that the inference is much more robust in nature. Similar trends were observed for MobileNetv2 and ResNet18.

I. Comparison to Previous Work

As an alternative to using Hessian, one can use gradient as the measure of sensitivity to select protected parameters. Such gradient-based protection have been implemented by Ko *et al.* [11] and Kim *et al.* [12] to identify the weights that have higher gradient with respect to loss and select them for reduced precision or higher compression. We compare our method to such gradient-based protection. For a fair comparison, the precision of both protected and unprotected parameters were identical for both the methods. Only the criteria to select the parameters for protection was varied. We perform experiments to quantize the unimportant parameters to 3 bit and 1 bit [using the 1 bit(7) and 1 bit(4)] states using both the methods. We also perform the experiments for both single write cycle and multiple write cycles to study the affect

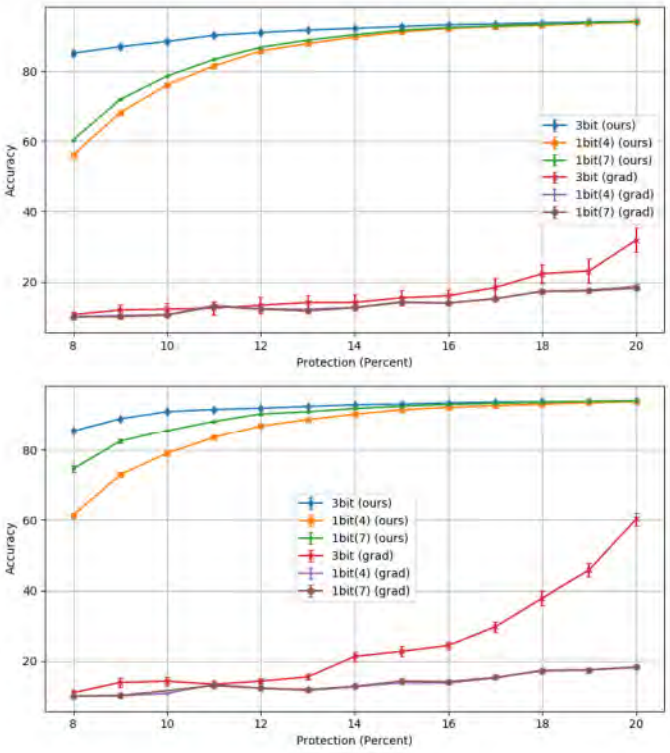

Fig. 11. Protection using this method versus protection using gradients [11] for single write cycle (top) and multiple write cycles (bottom).

of device variation on the performance of the methods. The comparison is shown for DenseNet121 in Fig. 11.

We see that Hessian-based protection greatly improves over using the gradient as the metric for protection. Gradient-based method fails to correctly identify the parameters to be quantized to 1 bit for both single and multiple write cycles leading to poor accuracy. For the unimportant parameters quantized to 3 bits, gradient-based methods need a much higher fraction of parameters to be protected for acceptable degradation in classification accuracy. We also see the performance of gradient-based protection to be highly susceptible to the amount of device noise. On the other hand, our sensitivity metric is able to limit the degradation in classification accuracy to less than 2% with 14% of the parameters protected. We posit that gradient-based method fails because to identify the important parameters as it is unable to incorporate higher order information in the geometry of the loss landscape.

V. Power Consumption Analysis

We perform a high-level estimation of efficiency overhead of Hessian-based protection using the concept of a hybrid PIM architecture (Fig. 12). We map un-protected parameters and computations to PIM-based VMM engines (bulk computation). The "protected" computations are performed using digital MAC engines. As "protected" computations are sparse, we can use a sparse convolution accelerator like SIGMA [19]. To estimate the efficiency and compute density for the PIM, we use DNN+Neurosim [38]. A detailed design of the hybrid PIM is beyond the scope of this article,

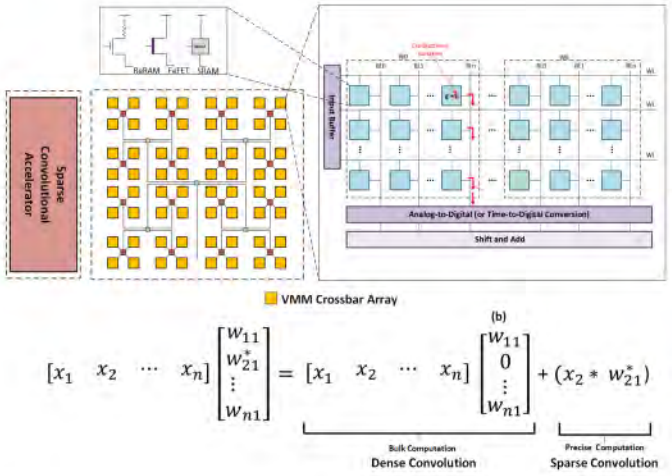

Fig. 12. Overview of the system architecture with a generic VMM engine and computational partitioning.

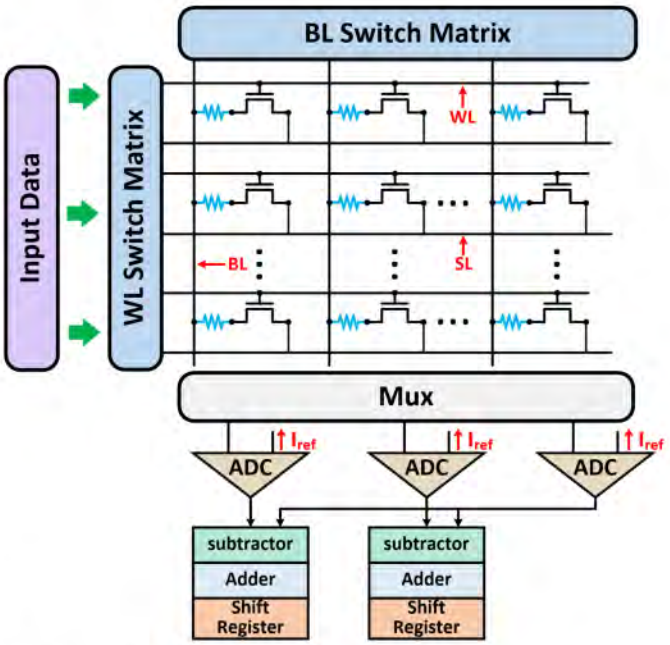

Fig. 13. PIM Architecture considered in NeuroSim simulations.

instead we perform a high-level analysis of the efficiency to make a strong case for pursuing a Hybrid PIM architecture. We report numbers for different PIM configurations. First, we perform 1-bit quantization and use the 3-bit device to store the 1-bit parameters. We report the efficiency for 1 bit(1) (44.84 kΩ), 1 bit(4)(11.21 kΩ), and 1 bit(7)(6.41 kΩ). The next case considered was using the 3-bit device to store 3-bit model parameters. The batch size of the input was set to 1.

To compute the energy efficiency of the hybrid PIM, we first calculate the number of computations performed using the protected parameters, these parameters are computed using the sparse convolutional accelerator and thus incur a high energy-per-compute penalty. We use the energy efficiency number for SIGMA reported in [19] to compute the total energy overhead of using this addition digital compute block. We add this overhead to the estimated power consumed by the PIM to compute the energy efficiency of the new hybrid PIM.

A. NeuroSim Assumptions and Setup

The architecture level simulation is conducted with DNN+NeuroSim [38], which considers the sparsity of the input data. The subarray design with 1T1R bit cell is shown in Fig. 13. The N-bit input data is first binarized and then applied at the word line (WL) in N cycles. As a result, no DAC is needed. In each cycle, the partial sum current along bit line (BL) is sensed by 8-bit SAR ADC at the periphery of the array. The read voltage applied at BL is assumed to be 0.5 V. To reduce the area overhead of ADC, 4 columns share one ADC through the mux. The transmission gate size in the mux is sized up to reduce the voltage drop due to large partial sum current. To represent negative weight, a dummy column with all the cells programmed to $[(G_{\max} + G_{\min})/2]$ is used. The partial sum from the regular columns are subtracted by the partial sum from the dummy column. For multibit input, shift-and-add is needed.

In the simulations, the chip size is constraint to 40 mm^2. The batch size is set to 1. The weights are loaded from DRAM and programmed to the RRAM array for layer-by-layer computation using the weight reloading mechanism proposed in [39]. The DRAM technology is LPDDR4 and the access energy used is 1.3 pJ/bit [40].

Two performance metrics: 1) energy efficiency in terms of *TOPS*/*W* and 2) compute density in terms of GOPS/mm^2 are considered. When counting the total number of operations, each MAC operation is regarded as two operations.

B. Results

Here, we make the design decision to keep the degradation in classification accuracy $> 90\%$ for both ResNet18 and DenseNet121 when data is written to the device with a single write cycle. We protect the fraction of parameters accordingly. The results are shown in Table II. We use the term baseline to refer to any design (fully PIM or hybrid) that uses 8-bit parameters as it is our goal to compare how hybrid designs with aggressive weight quantization schemes compare to their 8-bit counterparts. The baseline designs we compare the quantized hybrid designs to, are a fully PIM design that uses LRS7 and a hybrid PIM design that uses LRS4 and contains 6% of the parameters protected which are processed using a sparse accelerator.

We observe that, compared to a fully Digital implementation like SIGMA that has an efficiency of 480 GOPS/W, and a baseline fully PIM design that uses LRS7, a hybrid approach with aggressive weight quantization is able to improve the efficiency to 1.24 TOPS/W and 1 TOPS/W leading to an improvement of 1.33× and 1.28×, respectively, over the baseline fully PIM design. We also observe a compute density increase of 2.64× and 2.56×, respectively.

The results paint an interesting picture that the most efficient candidate for robust DNN inference might not always need to use "precise hardware." If we can tolerate a minor

TABLE II
EFFICIENCY AND COMPUTE DENSITY FOR VARIOUS HYBRID PIM IMPLEMENTATIONS

Model	Quantization	LRS	Protection (%)	Accuracy	Compute Density (GOPS/mm^2)	Final Efficiency (TOPS/W)
ResNet18	8 bit (Baseline)	4	6	94.38	8.72	0.79
		7	-	94.41	7.16	0.78
	1 bit	1	30	90.34	18.91	0.81
		4	17	91.17	18.91	1.00
		7	16	91.24	17.31	0.95
	3 bit	-	14	90.79	14.57	0.88
DenseNet121	8 bit (baseline)	4	6	94.20	2.7	0.99
		7	-	94.17	2.36	0.93
	1 bit	1	20	91.44	6.05	1.24
		4	15	91.74	5.12	1.14
		7	14	92.10	4.47	0.96
	3 bit	-	11	92.72	4.32	0.98

degradation, instead using devices that have inherent variation to reduce the power consumption and adding a small amount of precise computations to offset the errors introduced by these variations seems to be the optimal route. In the PIM, as the batch size is 1, there are frequent changes to the weights stored in the devices. In such a scenario, The DRAM access energy for weight reloading dominates the compute energy. Compared to 8-bit PIM, using 1-bit PIM, we see a reduction in weight reloading energy by 5.3× due to significant reduction in DRAM storage/access energy and interconnect energy. We see a reduction in compute energy by 6.5× due to reduction in the memory array energy. On the flipside, the relatively higher protection required for 1-bit parameters leads to a high penalty. The result of these two effects combined being that hybrid 1-bit PIMs increase the efficiency over baseline 8-bit PIMs using LRS7 by 1.33× for ResNet18.

We have not shown results for MobileNetv2 as DNN+Neurosim does not support depthwise separable convolutions. However, recent works [41], [42] have shown improved efficiency in mapping depthwise separable convolutions into PIM architectures with minimal overhead. As depthwise separable convolution leads to unused memory cells, the operation unit (OU) row compression approach by Yang *et al.* [42] can be used to process this sparse model. Alternatively, Zheng *et al.* [41] proposed MobiLattice which enables both analog and digital mode operations in PIM crossbar. The digital mode improves the computational efficiency of depthwise convolution layers by mitigating the redundant memory cells in the crossbars. Even in these modified PIM architectures, error in the computations still persist and hence there remains a need to mitigate the degradation using a precise digital computation block, as proposed. Hence, we decided to only present the algorithmic results for MobileNetv2 (Section IV) but not the efficiency results as our current iteration of Neurosim+DNN does not support it.

VI. ARCHITECTURAL CONSIDERATIONS

In this section, we propose an overview of a potential architecture to implement a hybrid PIM architecture. The design has two major compute components: 1) the PIM for bulk computation and 2) a sparse accelerator for precise digital computations.

A. PIM System

The PIM design uses multiple VMM arrays interconnected using a hierarchical network-on-chip (H-NoC) as proposed by Long *et al.* [20]. The VMM arrays perform mixed-signal computations and partial summations from multiple VMM arrays are accumulated in the router of the H-NoC. The reduction tree-based design of the H-NoC with in-router accumulation and data-forwarding allows high degree of parallelism. The architecture is illustrated in Fig. 12.

B. Sparse Accelerator

As the precise computations contain unstructured sparsity, the precise digital computation engine needs to be able to handle arbitrary degrees of sparsity with high efficiency—thus we propose using SIGMA. SIGMA uses bitmap-based encoding to smartly route nonzero weights to the right processing engines (FlexDPE) to ensure high degree of compute utilization. A novel FAN-based Adder topology is also used to handle variable sized dot-products efficiently [19].

C. Weight and Input Loading

As mentioned in Section V-A, we consider layer-by-layer processing of DNNs—this means that the appropriate parameters and activations have to be loaded from DRAM and routed to the correct destination for each layer in the DNN.

As important/ nonimportant parameters are differently processed, we propose to solve this problem by storing an extra bit alongside the parameters in the DRAM to signify the importance of a parameter. This importance bit is "1" for the important parameters and "0" for the nonimportant ones and is used by the PIM and SIGMA in different ways.

Alongside the parameters, the inputs to a given layer should also be fed into both the PIM and the Sparse Accelerator. Inputs to a given layer are the outputs from the previous layer hence, these need not be offloaded into the DRAM and can be stored in some temporary memory on-chip.

The data loading process of the PIM and SIGMA are independent of each other, once the data has been transferred from the DRAM as the allocation in controlled by their respective controllers. An overview of the dataflow is shown in Fig. 14.

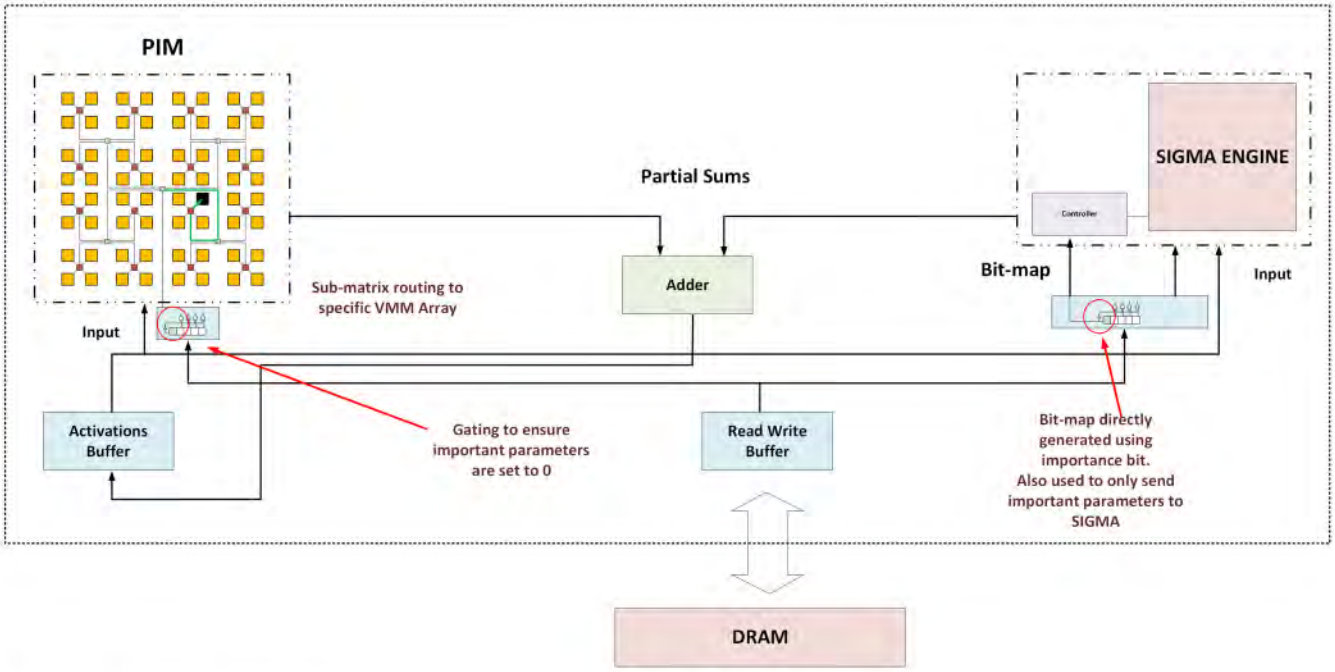

Fig. 14. Data flow in a hybrid PIM.

1) PIM: Parameters are loaded from the DRAM and large matrices are broken down into chunks that can be fit into the VMM engines. The H-NoC routes these chunks to the appropriate VMMs using an addressing mechanism [20]. As the important parameters are mapped to a 0 in the PIM, the importance bit is used to implement a gating mechanism to ensure this behavior. The nonimportant parameters are not affected and are simply routed.

The inputs stored in a temporary on-chip memory are also routed to the appropriate VMMs by the H-NoC using the same addressing mechanism.

2) Sparse Accelerator: SIGMA requires only the important parameters along with bit-map metadata. The importance bit included with the parameters is directly the bit-map required by SIGMA. The importance bit is also used as a gate to only allow important weights through.

The inputs are routed by the SIGMA controller to the appropriate flexDPE based on the parameter bit map. Here, we assume the only the parameter matrix is sparse while the input vector is dense. The multiplication between elements is dictated by the parameter bit map. However, as modern DNNs contain ReLU nonlinearity, intermediate activations also contain a certain degree of sparsity. Generating input bit map before feeding it into SIGMA could lead to reduction in SIGMA computations.

D. Compute

Once the parameters and inputs are fed into both the compute modules, the computations are independent of each other. The time taken by the PIM to compute is determined by the number of rows in the VMM engines and the number of router levels in the H-NoC and hence is deterministic in nature [20]. On the other hand, the time taken by SIGMA depends on the number of nonzero elements in the parameter bit map [19].

Compared to a baseline purely PIM design, additional latency is introduced only when the computation time for SIGMA is greater than the computation time for the PIM for a given layer. In our experiments, as the number of protected computations was just a fraction of the total computations, we observed the PIM compute time to be the latency bottleneck. However the accommodate cases where the compute time for SIGMA is greater than that of the PIM, we propose using an additional flag for each of modules that can be raised once their respective computations are finished. These flags can be used to synchronize the two compute modules and control the summation of the partial sums generated by both the compute units to give rise to the final activations of the layer. This activation can now be stored on the temporary on-chip memory to be used as an input for the next layer.

We would like to emphasize that this is just an overview of a feasible architecture that could be used to implement a Hybrid PIM and not the most optimal way to design such a system. We believe that the precise nature of how to go about designing a hybrid PIM architecture capable of achieving this goal is an interesting and open problem.

VII. Conclusion

We have presented an inference-time approach to improve robustness of PIM-based DNN accelerators under process variations in bitcells. We have developed a Hessian-based sensitivity vector to identify a set of critical parameters in a DNN that must be protected from process variation. Furthermore, the parameters designated nonimportant by our algorithm can also be quantized aggressively without appreciable loss in classification accuracy. A concept architecture is presented that

couples PIM and digital logic-based accelerators to incorporate Hessian-based protection in a DNN accelerator to obtain up to 1.33× increase in the energy efficiency compared to state-of-the-art PIM architecture. The promising results from our algorithmic and efficiency analysis suggest the need for future work on detail microarchitecture design of hybrid PIM. The special emphasis is necessary to consider the layer-to-layer nonuniformity in protected parameters.

REFERENCES

[1] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. HPCA*, 2017, pp. 541–552.

[2] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proc. ISCA*, 2016, pp. 27–39.

[3] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ISCA*, 2016, pp. 14–26.

[4] Y. Long *et al.*, "A ferroelectric FET based power-efficient architecture for data-intensive computing," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, p. 32.

[5] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. ISCA*, 2018, pp. 383–396.

[6] Y. Long, X. She, and S. Mukhopadhyay, "Design of reliable DNN accelerator with un-reliable ReRAM," in *Proc. DATE*, 2019, pp. 1769–1774.

[7] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. CVPR*, 2019, pp. 8612–8620.

[8] G. Karakonstantis, G. Panagopoulos, and K. Roy, "HERQULES: System level cross-layer design exploration for efficient energy-quality tradeoffs," in *Proc. ISLPED*, 2010, pp. 117–122.

[9] C. Ho *et al.*, "Integrated HfO2-RRAM to achieve highly reliable, greener, faster, cost-effective, and scaled devices," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2017, pp. 2.6.1–2.6.4, doi: 10.1109/IEDM.2017.8268314.

[10] Y. Luo, X. Han, Z. Ye, H. Barnaby, J.-S. Seo, and S. Yu, "Array-level programming of 3-bit per cell resistive memory and its application for deep neural network inference," *IEEE Trans. Electron Devices*, vol. 67, no. 11, pp. 4621–4625, Nov. 2020.

[11] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay, "Adaptive weight compression for memory-efficient neural networks," in *Proc. DATE*, 2017, pp. 199–204.

[12] D. Kim, J. Kung, and S. Mukhopadhyay, "A power-aware digital multilayer perceptron accelerator with on-chip training based on approximate computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 2, pp. 164–178, Apr.-Jun. 2017.

[13] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe, "Optimal brain surgeon: Extensions and performance comparisons," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Denver, CO, USA, 1993, pp. 263–270.

[14] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," 2017. [Online]. Available: arXiv:1705.07565.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. CVPR*, 2015, pp. 770–778.

[16] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," 2018. [Online]. Available: arXiv:1801.04381.

[17] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," 2016. [Online]. Available: arXiv:1608.06993.

[18] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10,"

[19] E. Qin *et al.*, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2020, pp. 58–70.

[20] Y. Long *et al.*, "A ferroelectric FET-based processing-in-memory architecture for DNN acceleration," *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 5, no. 2, pp. 113–122, Dec. 2019.

[21] J. Wang *et al.*, "14.2 a compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 224–226.

[22] C. Xue *et al.*, "24.1 a 1mb multibit ReRAM computing-in-memory macro with 14.6ns parallel MAC computing time for CNN based ai edge processors," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 388–390.

[23] W.-H. Chen *et al.*, "A 65nm 1mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *Proc. ISSCC*, 2018, pp. 494–496.

[24] M.-Y. Lin *et al.*, "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, p. 31.

[25] H. Li *et al.*, "Variation-aware, reliability-emphasized design and optimization of RRAM using spice model," in *Proc. Design Autom Test Europe Conf. Exhib. (DATE)*, 2015, pp. 1425–1430.

[26] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 12, pp. 1859–1880, Dec. 2005.

[27] S. K. Gonugondla, M. Kang, and N. R. Shanbhag, "A variation-tolerant in-memory machine learning classifier via on-chip training," *IEEE J. Solid-State Circuits*, vol. 53, no. 11, pp. 3163–3173, Nov. 2018.

[28] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[29] K. Ni, W. Chakraborty, J. Smith, B. Grisafe, and S. Datta, "Fundamental understanding and control of device-to-device variation in deeply scaled ferroelectric FETs," in *Proc. Symp. VLSI Technol.*, 2019, pp. T40–T41.

[30] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou, "Empirical analysis of the hessian of over-parametrized neural networks," 2017. [Online]. Available: arXiv:1706.04454.

[31] Q. Nguyen, "On connected sublevel sets in deep learning," in *Proc. ICML*, vol. 97, 2019, pp. 4790–4799.

[32] P. Xu, D. He, C. De Sa, I. Mitliagkas, and C. Ré, "Accelerated stochastic power iteration," in *Proc. AISTATS*, vol. 84, 2018, pp. 58–67.

[33] N. Golmant *et al.*, "pytorch-hessian-eigenthings: efficient pytorch hessian eigendecomposition."

[34] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, NY, USA: Wiley, 2006.

[35] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018.

[36] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018. [Online]. Available: arXiv:1806.08342.

[37] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Assoc. Inc., pp. 8024–8035, 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[38] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, 2019, p. 32.

[39] A. Lu, X. Peng, Y. Luo, and S. Yu, "Benchmark of the compute-in-memory-based DNN accelerator with area constraint," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1945–1952, Sep. 2020.

[40] D. Skinner, "Lpddr4 moves mobile," *JEDEC Mobile*, 2013.

[41] Q. Zheng *et al.*, "MobiLatice: A depth-wise DCNN accelerator with hybrid digital/analog nonvolatile processing-in-memory block," in *Proc. 39th Int. Conf. Comput.-Aided Design* New York, NY, USA, 2020, pp. 1–9.

[42] T.-H. Yang *et al.*, "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proc. 46th Int. Symp. Comput. Architect. (ISCA)*, New York, NY, USA, 2019, p. 236–249.

Saurabh Dash received the B.S. and M.S. degrees in electronics and electrical communications engineering from IIT Kharagpur, Kharagpur, India, in 2019, where he received the Order of Merit. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Georgia Institute of Technology, Atlanta, GA, USA, where he was awarded the ECE Fellowship.

His research interests include robust machine learning and learning in discrete-event dynamical systems.

Yandong Luo received the B.Eng. degree from Zhejiang University, Hangzhou, China, in 2016, and the M.S. degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2018. He is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA.

His current research interests include in-memory computing with emerging nonvolatile memory device and deep learning accelerator design.

Anni Lu received the B.S. degree in electronic information engineering from Tianjin University, Tianjin, China, in 2019. She is currently pursuing the Ph.D. degree in electrical and computer engineering with Georgia Institute of Technology, Atlanta, GA, USA.

Her current research interests include deep learning algorithm and hardware co-design.

Shimeng Yu (Senior Member, IEEE) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2009, and the M.S. degree and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2011 and 2013, respectively.

He is currently an Associate Professor of electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA, USA. From 2013 to 2018, he was an Assistant Professor with Arizona State University, Tempe, AZ, USA. His research expertise is on the emerging nonvolatile memories for applications, such as deep learning accelerator, in-memory computing, 3-D integration, and hardware security.

Saibal Mukhopadhyay (Fellow, IEEE) received the B.E. degree in ETCE from Jadavpur University, Kolkata, India, in 2000, and the Ph.D. degree in ECE from Purdue University, West Lafayette, IN, USA, in 2006.

He is currently a Joseph M. Pettit Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA.