Neural Network Training With Stochastic Hardware Models and Software Abstractions

Bonan Zhang, *Student Member, IEEE*, Lung-Yen Chen, *Student Member, IEEE*, and Naveen Verma, *Member, IEEE*

***Abstract*— Machine learning inference is of broad interest, increasingly in energy-constrained applications. However, platforms are often pushed to their energy limits, especially with deep learning models, which provide state-of-the-art inference performance but are also computationally intensive. This has motivated algorithmic co-design, where flexibility in the model and model parameters, derived from training, is exploited for hardware energy efficiency. This work extends a model-training algorithm referred to as Stochastic Data-Driven Hardware Resilience (S-DDHR) to enable statistical models of computations, amenable for energy/throughput aggressive hardware operating points as well as emerging variation-prone device technologies. S-DDHR itself extends the previous approach of DDHR by incorporating the statistical distribution of hardware variations for model-parameter learning, rather than a sample of the distributions. This is critical to developing accurate and composable abstractions of computations, to enable scalable hardware-generalized training, rather than hardware instance-by-instance training. S-DDHR is demonstrated and evaluated for a bit-scalable MRAM-based in-memory computing architecture, whose energy/throughput trade-offs explicitly motivate statistical computations. Using foundry data to model MRAM device variations, S-DDHR is shown to preserve high inference performance for benchmark datasets (MNIST, CIFAR-10, SVHN) as variation parameters are scaled to high levels, exhibiting less than 3.5% accuracy drop at 10× the nominal variation level.**

***Index Terms*— Statistical computing, in-memory computing, circuit reliability, deep learning, fault tolerance.**

I. Introduction

MACHINE learning, especially deep learning, is of interest in a wide range of applications, and increasingly in embedded and sensor applications, due to the high inference performance achieved on input streams such as vision and speech. However, deep learning inference is computationally expensive, especially with state-of-the-art models of interest in these application domains. This pushes hardware resource requirements (amount of memory, energy consumption) beyond the limits of the platforms in many such applications [1]. For instance, object classification using AlexNet on smartphones can run for less than an hour [2], and face detection can run for less than 40 minutes on today's smart glasses [3].

A distinctive characteristic of machine learning algorithms is their statistical nature. It has been widely recognized that this affords opportunities for addressing computational resource requirements [4]. This is because allowing operations to be statistical enables relaxation of the operations and/or the resources they required. This has been indicated at all levels of the computing stack, from devices [5], to circuits [6], to micro/architectures [7].

Algorithm-hardware co-design is required in order to harness opportunities at the algorithmic level for addressing physical resource requirements at the hardware levels. This, in turn, requires forming abstractions of the statistical hardware operations that can be propagated up to the algorithmic level. It has been shown that the fidelity of those abstractions, namely in how they represent the statistics of the underlying operations, critically determines the possible gains. Simply treating statistical operations as noisy computation and leveraging inherent noise tolerance of statistical algorithms yields limited gains, while fully representing the statistics of operations and incorporating these within statistical optimization algorithms yields much greater gains, ultimately set by information-theoretic limits [8]. This implies trade-offs between the complexity, the breadth of applicability, and system-level gains possible from statistical abstractions.

This work extends an algorithm referred to as Stochastic Data-Driven Hardware Resilience (S-DDHR), an approach to training machine-learning models incorporating variations of the hardware used to execute inference. In S-DDHR, which itself can run on conventional hardware (e.g., GPUs), training is performed once, to learn inference-model parameters generalized to the statistics across instances of a hardware architecture, whose computations are affected by device-level variations between the different hardware instances of the architecture. This improves on previous approaches, where training is required on an instance-by-instance basis of a hardware architecture [8]–[10].

The specific contributions of this work are as follows:

1) We extend the S-DDHR algorithm, which was initially presented in short form [11]. In this work, we generalize it, specifically showing how the underlying statistical computations can be robustly composed into more complex computations via their abstractions, as required for architectural scalability and application mapping. A realization of the S-DDHR algorithm taking advantage of such composability for deep learning systems

Manuscript received August 12, 2020; revised November 20, 2020 and December 22, 2020; accepted January 8, 2021. Date of publication January 26, 2021; date of current version March 8, 2021. This work was supported by the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under Agreement FA8650-18-2-7866. This article was recommended by Associate Editor A. James. *(Corresponding author: Bonan Zhang.)*

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: bonanz@princeton.edu; lungyenc@princeton.edu; nverma@princeton.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3052981.

Digital Object Identifier 10.1109/TCSI.2021.3052981

is demonstrated, along with different approaches to the statistical models and corresponding abstractions employed.

2) We evaluate the S-DDHR algorithm on an MRAM-based in-memory computing (IMC) architecture, using statistical physics-based MRAM device models for a foundry technology. Unlike the previous report of S-DDHR [11], the architecture employs multi-bit computations through algorithmic and architectural extensions, exploiting and validating composability of the abstractions. S-DDHR demonstrates restoration of high inference performance at different precision levels, even as hardware variations are scaled to high levels.
3) We demonstrate the integration of the statistical abstractions necessary for S-DDHR into domain-specific design frameworks for deep learning systems (e.g., PyTorch is used in this work). This provides an end-to-end connection from the device/circuit hardware level, where computation statistics originate, to the application-development level, where algorithmic co-design can be exploited.

The remainder of this paper is organized as follows. Section II presents background, both on previous approaches that leverage statistical operations to reduce resource requirements, and on statistical algorithmic methods that are leveraged in this work to extend these previous approaches. Section III describes details of the S-DDHR approach. Section IV presents the MRAM-based IMC architecture used for demonstration and evaluation. Section V describes the statistical models and abstractions constructed for S-DDHR, and the software required for executing neural networks on the IMC architecture. Section VI presents the evaluation approach and results. Finally, Section VII provides conclusions.

II. Background

The following subsections describe how trade-offs emerge between resource-efficiency and the statistics of hardware architectures, and how these have been exploited previously. Finally, an algorithmic approach to neural network training is described, which is leveraged in this work for enabling stochastic hardware.

A. IMC and Energy/Throughput-Vs.-SNR Trade-Offs

The resource requirements for deep learning inference have motivated a wide range of research in the design of specialized hardware accelerators [12]. Generally, hardware acceleration addresses compute resources by mitigating the overheads of programmability. However, data-intensive operations in deep learning inference, in the form of high-dimensionality matrix-vector multiplications (MVMs), also make data-movement a dominant concern. This has led to a focus on spatial architectures, where hardware processing engines (PEs) are specifically arranged in a two-dimensional array, to align with and efficiently address the dataflow in MVMs. A wide range of different dataflow architectures have been researched to exploit this alignment [13]–[15].

IMC is a spatial architecture that exploits such alignment aggressively, by using a two-dimensional arrangement of dense bit cells as PEs. The objective in IMC is to access a computation result over multiple stored bits, rather than accessing individual stored bits one at a time, as done in standard memory. This instates a trade-off, where data communication costs (energy, delay) are amortized, but data dynamic range requirements are increased. In turn, accommodating the resulting dynamic range within existing high-density memory circuits leads to reduced signal-to-noise ratio (SNR), and thus fundamental energy/delay vs. SNR trade-offs in IMC [16], [17]. This makes IMC an exemplary architecture for exploiting statistical operations to address resource requirements [18], [19].

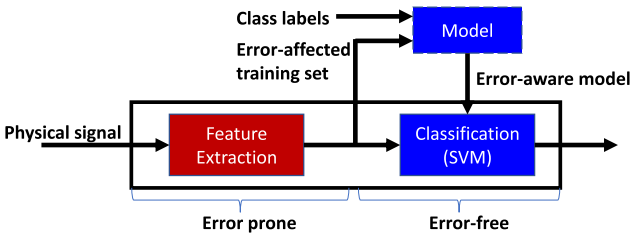


Fig. 1. Illustration of the DDHR System [8].

The computation noise in IMC architectures can arise from multiple sources. Typically, fitting computation within dense bit cells leverages analog operation, whereby devices can provide functionality not restricted to simple switch-based models of computation. This raises susceptibility to all forms of analog non-idealities, including process/temperature/aging variations, device non-linearities, and electronic noise [20], [21]. Especially with the opportunities presented by emerging nanoscale memory technologies, process variations, which worsen with device scaling [22], are a major concern.

B. Hardware-Algorithm Co-Design

The benefits of hardware-algorithmic co-design have been widely recognized in deep learning systems [23]. A particularly successful example is quantized neural networks, where aggressive quantization of weights and activations introduces a source of hardware noise (quantization noise), but which can be handled algorithmically by modeling and incorporating it into the statistical optimization employed for parameter training [24]–[26]. A key to this approach is the ability to robustly model the hardware quantization effects at the algorithmic level.

Beyond quantization, modeling the diverse sources of noise arising from hardware non-idealities (variations, non-linearities) is a key challenge. Nonetheless, to avoid modeling challenges and illustrate the potential that co-design can have in enabling low-SNR, resource-aggressive hardware, several approaches have used the specific instance of hardware at hand directly within the parameter-training process. One example is Data-Driven Hardware Resilience or DDHR [8]. As shown in Fig. 1, the feature-extraction stage in a sensor-inference system is assumed to cause computation noise due to hardware non-idealities. However, by training a classification stage using the computed feature vectors, the statistics of the computation noise are properly represented in the training data set. This leads to model parameters optimized to the resulting statistics, referred to as an *error-aware model*, and subsequently used for inference. DDHR has been analyzed extensively using an FPGA platform to controllably emulate hardware non-idealities (static faults) [8]. It was found that

such co-design enables performance not limited by error rates or magnitudes, but rather more fundamentally by the level of mutual information retained between the error-affected feature vectors and the data classes.

Works have extended DDHR in a number of ways, especially relevant for IMC. First, the original presentation of DDHR addresses hardware non-idealities in the stages preceding the application of a data-driven model. Error-Adaptive Classifier Boosting (EACB) overcomes hardware non-idealities within the stage applying the data-driven model itself [9]. This is achieved by leveraging the machine learning algorithm Adaptive Boosting (AdaBoost) [27], where multiple weak classifiers are trained iteratively and combined to achieve an overall strong classifier. This enables iterative training wherein weak-classifier outputs from previous iterations, which are also affected by hardware non-idealities, are employed to suitably adapt the parameters of subsequent weak classifiers. EACB has the added benefit that the use of weak classifiers, enables implementation within low-energy IMC hardware. This was demonstrated on a fully-row/column parallel IMC design [28].

Second, further works have implemented adaptive training of a data-driven machine learning model implemented in IMC, by leveraging Stochastic Gradient Decent (SGD). Like AdaBoost, SGD employs iterative training, affording the opportunity to incorporate hardware non-idealities into the parameter-updating optimization process. For example, previous work used SGD to train an IMC implementation of a deep neural network, which spanned across four chips [29]. Another example used SGD to train an IMC implementation of a Support Vector Machine (SVM), with hardware for parameter updating also incorporated in the architecture [30].

In all of these cases, the hardware non-idealities are represented by a specific instance of hardware, by using that instance within the parameter-training algorithm. While this demonstrates the promise of hardware-algorithmic co-design for addressing computation noise due to hardware non-idealities, instance-by-instance training will be infeasible in many systems. The proposed approach overcomes this by using a statistical model to represent the distribution over all hardware instances. Rather than a specific instance of hardware, i.e., a sample from the distribution, parameter training is now generalized to the entire distribution, overcoming the need for instance-by-instance training. This leverages the approach of stochastic training described next.

C. Stochastic Training

For a neural network, a batch of training data is passed through a network model to compute an aggregate error metric (loss function) at the network output, with respect to the training-set ground truth. The gradient of each model parameter with respect to the error is then determined and adjustments are propagated back to each model parameter, referred to as backpropagation. In stochastic training, stochastic noise is introduced in the forward path of a neural network in various ways. Often, generic noise distributions are selected. One example is *whiteout* [31], where additive Gaussian noise is introduced in the forward path. In another example, multiplicative noise is introduced from a Bernoulli distribution to the hidden nodes in the network [32]. This previous research has motivated stochastic training as a means of speeding up the convergence of the backpropagation algorithm [33] or as a regularizer to improve the generalization (reduce overfitting) of the model and parameters [34]. Recent research has started to explore stochastic training in the context of noisy computations during inference. For instance, motivated by analog neuromorphic computing, stochastic training is applied on a long short-term memory (LSTM) network to enhance the robustness of the network model [35], once again to a particular instance of hardware, but which can intrinsically exhibit transient noise due to electronic sources. Like previous research, a generic distribution for noise is used, rather than a distribution designed to model the statistics across instances of underlying hardware architecture. In the proposed work, a statistical model is derived specifically for the distribution across instances of underlying hardware architecture. Different algorithmic approaches as well as different computation mappings to the architecture are investigated to evaluate the efficiency and fidelity of parameter training, across different instances of the hardware architecture.

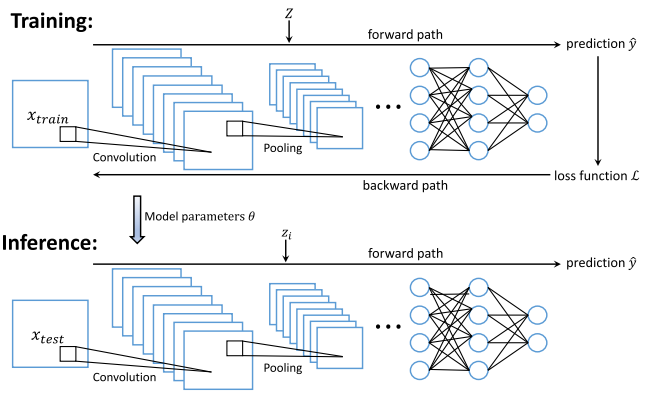


Fig. 2. Stochastic Data-Driven Hardware Resilience (S-DDHR) during training and inference [11].

III. Stochastic DDHR (S-DDHR) Overview

The proposed approach is referred to as Stochastic Data-Driven Hardware Resilience (S-DDHR). It extends the previous idea of DDHR [8], by employing a statistical model that represents computations across instances of the underlying hardware architecture that are affected by statistical variations. Specifically, the use of such a model is demonstrated for algorithmic co-design of deep neural network inference models.

A. Overview

A visualization of S-DDHR across both the training and inference phases is shown in Fig. 2. Z is a random variable, which is activated in the forward path of the training process. Its distribution is designed to represent the statistics of computations performed across instances of the hardware architecture. Without loss of generality, Z can be regarded as computation noise on top of a nominal, deterministic model of the underlying computation performed. To help concretize, and better relate to stochastic-training approaches that have been proposed for deep learning training, such noise can be thought of as affecting all layers of a neural network, which will then lead to a net effect on parameter updates used for backpropagation.

Algorithm 1 Training Process for S-DDHR. J Is the Number of Layers, Z Is the Random Variable Representing Stochastic Hardware, $\mathcal{L}$ Is the Loss Function, $f_j(\cdot)$ Is the Forward Function for j^{th} Layer, λ Is the Decay Factor for Learning Rate

Require: previous model parameters θ, previous learning rate η, input to j^{th} layer a_j ($x_{train} = a_0$)

Ensure: statistics of hardware variations (Z) is captured by loss function $\mathcal{L}$, and carried to the updated parameters θ_j^{t+1}

{**Forward propagation**}

for $j = 1$ to J **do**

$a_j \leftarrow f_j(a_{j-1}, \theta_j, Z)$

end for ▷ $\mathcal{L}$ is a function of $f(x_{train}, \theta, Z)$ and y_{train}

{**Backward propagation**}

for $j = J$ to 1 **do**

$\frac{\partial \mathcal{L}}{\partial \theta_j} \leftarrow \frac{\partial \mathcal{L}}{\partial f_j(a_{j-1}, \theta_j, Z)} \frac{\partial f_j(a_{j-1}, \theta_j, Z)}{\partial \theta_j}$

end for

for $j = 1$ to J **do**

$\theta_j^{t+1} \leftarrow \theta_j - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}$

$\eta^{t+1} \leftarrow \lambda \eta$

end for

With regards to designing the distribution of Z to represent the statistics of hardware computations, two types of variation can be considered: (1) static variations, e.g., due to limited engineering control on critical device-level parameters [20]; and (2) transient variations, e.g., due to electronic noise sources, temperature variations, etc. S-DDHR can, in general accommodate both types of variations. However, with the focus of the demonstration in this work on IMC, especially leveraging emerging nanoscale memory technologies, static variations are the primary concern, particularly as they become more prominent with dimensional scaling and emerging technologies [22]. In this case, after model parameters are learned through the training process, inference is performed via computations modeled as a sample z_i of Z. Our interest is in the inference performance thus achieved. Incorporating transient variations simply requires modeling Z as the sum of two random variables $Z = Z^s + Z^t$, where Z^s represents static variations, while retaining the random variable Z^t for transient variations (possibly as a function of time, temperature, etc. to represent aging, temperature coefficients, etc.), then modeling inference computations by a sample z_i of Z, Hence the predicted output $\hat{y}$ can be expressed as

$$\hat{y}_{train} = f(x_{train}, \theta, Z) \tag{1}$$

$$\hat{y}_{test} = f(x_{test}, \theta, z_i) \tag{2}$$

where $\hat{y}_{train}$, $\hat{y}_{test}$ represent the training and inference outputs respectively, x_{train}, x_{test} are training and inference inputs, respectively, and θ represents the model parameters. This training process is described in Algorithm 1 for one epoch.

B. Hardware Model Integration in Training

S-DDHR incorporates the stochastic hardware model into the training process during evaluation of the forward loss function employed for parameter optimization. Specifically, in this work, cross-entropy loss functions are used, with the form $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^t$, where $\mathcal{L}^t$ denotes the loss function for the t^{th} training sample, and T denotes the total number of samples.

For quantitative evaluation of S-DDHR, several different loss functions, including ones incorporating the statistical hardware model, are defined and used for comparison. The loss functions are defined as below, where x and y represent the input and output of the neural network, $f(\cdot)_c$ is the c^{th} element of the neural network output represented by its corresponding forward function, and C denotes the number of classes:

- **Ideal Variation-Free Hardware.** We define a model that presumes ideal variation-free hardware. In this case, the loss function is independent of Z, having the form

$$\mathcal{L}^t_{Ideal} = -\sum_{c=1}^{C} y_c^t \log f(x^t, \theta)_c \tag{3}$$

- **DDHR Representation of Hardware.** We define a model which uses a specific sample of stochastic hardware z_i within both the training and inference processes. This is the approach taken in DDHR [8], and is expected to yield high inference performance. However, it imposes a high training cost, requiring instance-by-instance training of hardware. Its loss function has the form

$$\mathcal{L}^t_{DDHR} = -\sum_{c=1}^{C} y_c^t \log f(x^t, \theta, z_i)_c \tag{4}$$

- **Per-sample Representation of Hardware.** Considering the training complexity of DDHR, an alternate naive approach is considered, where different samples of stochastic hardware are used for the training and inference processes. Specifically, a sample $z_\emptyset$ of Z is used for training, while different samples z_i are used for inference. Since the sample $z_\emptyset$ does not adequately represent the distribution Z, this approach is expected to yield low inference performance. Its loss function has the form

$$\mathcal{L}^t_{Fixed} = -\sum_{c=1}^{C} y_c^t \log f(x^t, \theta, z_\emptyset)_c \tag{5}$$

- **S-DDHR Representation of Hardware.** As mentioned, the proposed approach of S-DDHR models the distribution of stochastic hardware, via the random variable Z. As we describe later, two different models will be considered for the specific demonstration example employed, with one approximating the statistics. This will enable preliminary evaluation of the fidelity required of the stochastic hardware model. For both cases, the loss function has the form

$$\mathcal{L}^t_{S\text{-}DDHR} = -\sum_{c=1}^{C} y_c^t \log f(x^t, \theta, Z)_c \tag{6}$$

IV. Demonstration Architecture

While S-DDHR has the potential to apply to a broad range of hardware architectures affected by statistical behaviors, its feasibility ultimately depends on the ability to adequately model the statistics via hardware and software abstractions, as well as the ability to model the statistics of computations composed from those abstractions. This, in turn, depends on the properties of the statistical behaviors themselves (ergodicity, stationarity, etc.), which need to be assessed for hardware architectures. To evaluate S-DDHR, we focus on an

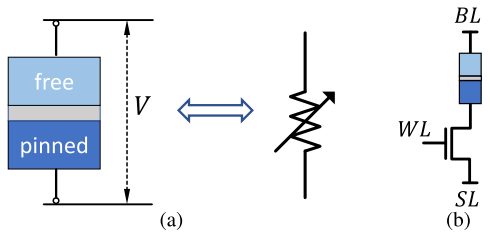


Fig. 3. MRAM bit-cell. (a) The MTJ unit consists of a free layer, a pinned layer, and a tunneling oxide in between. (b) The MRAM bit-cell is comprised of an MTJ unit connected with an access transistor in series.

IMC architecture based on the emerging non-volatile memory technology MRAM, whose device-level variations exhibit properties amenable for statistical modeling. Below, details of the MRAM technology are described first, followed by the overall IMC architecture.

A. Magnetic RAM (MRAM)

MRAM is being integrated into VLSI technologies for its potential to achieve much higher density than SRAM as well as its ability to provide non-volatile storage, which is particularly beneficial in low-power, low-duty-cycle applications. This work considers a foundry embedded MRAM technology [36], based on a perpendicular magnetic tunneling junction (MTJ). The MTJ uses a very thin layer of magnetic material that can pass a tunneling current, which, at specific biases, changes the polarity of the magnetic spin, resulting in an effective change in the resistance through the layer [37]. Specifically, the magnetic material consists of a free component, where the magnetic orientation can be switched, a pinned component, where the magnetic orientation is fixed, and a tunneling oxide in between. By changing the orientation of the free component, relative to the pinned component, the MTJ behaves like a variable resistor. When the free component is switched to an orientation parallel with the pinned component, the MTJ exhibits a low resistance, denoted by R_P. On the other hand, when the free component is switched to an orientation anti-parallel with the pinned component, the MTJ has a high resistance, denoted by R_{AP}. The complete MRAM bit-cell is shown in Fig. 3. It includes an access transistor in series with the MTJ device, where the access transistor and MTJ connect across the column-wise source line (SL) and bit line (BL), respectively, and the access-transistor gate connects with a row-wise word line (WL).

Proper MTJ operation requires a very thin oxide layer for reasonable tunneling current. However, such a layer is an important contributor to the variation of both the resistance in each state and the current required for changing the state [38], [39]. Particularly the resistance variations result in IMC computation noise, which will be targeted for correction via S-DDHR.

B. MRAM-Based In-Memory Computing

Fig. 4 shows the MRAM-based IMC architecture considered in this work, corresponding to a $2M \times N$ array of bit cells, where M denotes the number of computing units per column and is half the number of rows. The architecture implements matrix-vector multiplication (MVM), $y = Wx$, where $W = (w_{i,j}) \in \{-1, 1\}^{M \times N}$, and $x = [x_0, x_1, \cdots, x_{m-1}]^T \in \{-1, 1\}^M$. The input-vector elements x_m (neural network input activation) are driven in parallel on the word lines (WL_i), to be multiplied by matrix elements $w_{m,n}$ (neural network weights) stored in the MRAM bit cells.

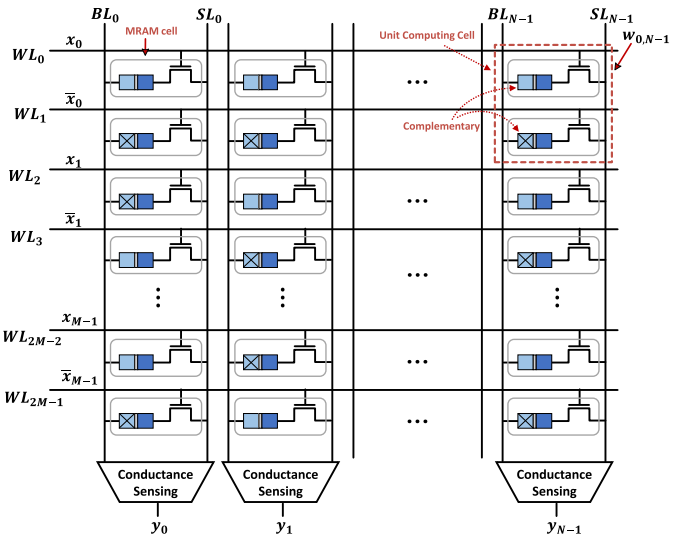


Fig. 4. MRAM-based in-memory computing architecture.

In this architecture, the input-vector and matrix elements can be multi-bit, as will be described later. Multiplications and accumulations are implemented on single input-vector and matrix bits in each column as follows. Each matrix element bit is stored in two complementary bit cells of the same column. This is done by switching one bit cell to the parallel state (with resistance R_P), and the other bit cell to the anti-parallel state (with resistance R_{AP}). Input-vector element bits are also applied as complementary signals on the corresponding word lines. With the proper arrangement of complementary signals, binary multiplication (XNOR) is implemented as shown in Table I, where x_m and $w_{m,n}$ values of -1 are represented by a logic 0. The reason the complementary bit-cell structure is needed is that multiplication is a commutative operation. However, the input-vector element bit connects to an access transistor, which modulates a transistor conductance, while the matrix element bit is stored in the bit cell, which modulates the MTJ conductance. With a single bit cell, this leads to multiplication output in the form of a bit-cell conductance that is asymmetric with respect to the two operands. Using two complementary bit cells overcomes this, giving the symmetric output conductance of G_P and G_{AP}, corresponding to 1 and 0, respectively, as shown in Table I. Then, accumulation is implemented in the column simply as the sum of conductance between the bit line BL_n and source line SL_n, from the parallel complementary bit cells.

In this way, IMC benefits both energy and latency, compared to a traditional architecture that separates memory and computation. In traditional architectures, individual bits are read out and fed to a digital accelerator for computation. This incurs memory-read energy and latency, which scales with the memory size. Comparatively, IMC does not incur memory-read costs for each bit. Instead, it accesses a compute result over many bits at once. In doing so, it incurs the energy/throughput-vs.-SNR trade-off described in Section II-A, which, in turn, is used to set the row and column dimensionalities of the IMC macro. By considering the resistance levels of the MTJ devices, the current required for adequate voltage sensing

TABLE I
XNOR OPERATION FOR THE MRAM-BASED IMC ARCHITECTURE

x_m	WL_{2m}	WL_{2m+1}	$w_{m,n}$	G_{2m}	G_{2m+1}	Output conductance	Binary equivalence
0 (-1)	0	VDD	**0 (-1)**	G_{AP}	G_P	G_P	**1**
0 (-1)	0	VDD	**1**	G_P	G_{AP}	G_{AP}	**0 (-1)**
1	VDD	0	**0 (-1)**	G_{AP}	G_P	G_{AP}	**0 (-1)**
1	VDD	0	**1**	G_P	G_{AP}	G_P	**1**

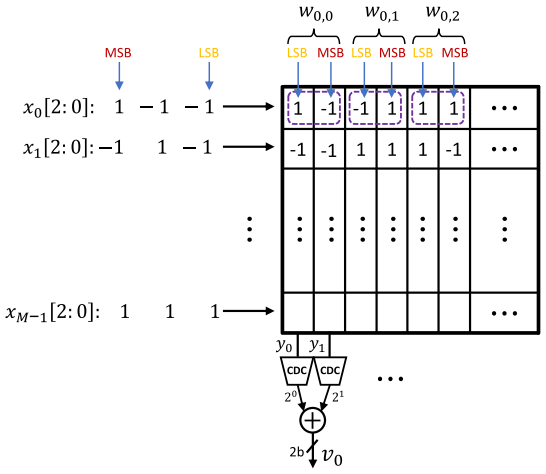


Fig. 5. Illustration of BPBS scheme.

can be determined for different column dimensionalities. For instance, 256 rows is found to require roughly 1mA, which is selected as the limit for feasible distributions. From this, the row dimensionality (number of columns) is selected to be 128, resulting in roughly 0.13A of current for the macro, which is selected to be the limit for robust power-grid current-density handling. Simulations of the 256×128 macro chosen show that a traditional architecture separating memory and computation incurs memory-read energy of 3.45pJ/bit, requiring 113nJ for reading from the entire array; on the other hand, the IMC architecture incurs compute energy of 65.4pJ/column, requiring 8.37nJ for the entire array. This corresponds to $13\times$ energy reduction.

In addition to the circuit-level considerations discussed above, the macro size and architectural design, integrating multiple macros, impacts the overall energy, throughput, and latency (e.g., as discussed in [40]) of full neural-network execution. Given the range of architectural considerations and associated computation-mapping strategies for different neural networks, we keep the primary focus of this work on the design of the IMC macro itself and its statistical abstraction/model, for enabling subsequent hardware and software design.

C. Extension to Multi-Bit Computation

To enable multi-bit input-vector and matrix elements, the bit-parallel/bit-serial (BPBS) scheme proposed in [7] is employed. Fig. 5 illustrates this, for 2-bit matrix elements and 3-bit input-vector elements. The multiple bits of each matrix element are stored in parallel columns of the MRAM memory array, such that each element spans two adjacent columns for the 2-bit case. On the other hand, the multiple bits of each input-vector element are input serially. This reduces each column computation to that of input vectors and matrices with single-bit elements, as described above, with readout achieved by a conductance-to-digital converter (CDC). Extension to multi-bit elements is then achieved by combining the digitized column computations in space and time. With each digitized column computation thus corresponding to a specific input-vector and matrix element bit position, the column computations are binary weighted by a corresponding amount (via digital bit shifting) and summed together. In this work, CDCs are assumed to have enough bit-precision to accommodate the full dynamic range of the column computation (i.e., $log_2(M+1)$). The impact of the quantization error introduced by the finite precision A/D conversion has been analyzed in detail in [7], where it is demonstrated that such error can be readily modeled for incorporation into quantized neural network training algorithms.

For a given layer, the matrix elements of input activations and weights are first normalized and then quantized to the corresponding dynamic range. The quantized elements are then binarized to feed into the IMC block for MVM computations as described above. Since complementary bit cells implement XNOR logic, individual bits represent $-1/+1$ (rather than $0/1$), allowing bit-wise multiplications to be mapped to XNOR operations. B-bit numbers are thus represented as follows:

$$x = \sum_{i=1}^{B-1} b_i \times 2^{i-1} + (b_{0+} + b_{0-}) \times 2^{-1} \tag{7}$$

where two equally-weighted bits (b_{0+} and b_{0-}) are required for the least-significant bit, so that a value of zero can be properly represented. As described in [7], the XNOR-logic can be modified to implement AND-logic, enabling the use of standard two's complement number representation.

Such multi-bit computation has both energy and throughput overheads. In terms of energy, the IMC compute cycles scale with the number of input-element bits, and the IMC columns required scale with the number of matrix-element bits. Thus, the compute energy scales linearly with both the input- and matrix-element precision. Similarly, throughput scales inversely with the number of IMC compute cycles, and thus the input-element bits, while the area, and corresponding area-normalized throughput, scales inversely with the number of IMC columns, and thus the matrix-element bits. In addition to the IMC energy and throughput scaling, BPBS requires digital hardware following IMC for shift-add reconstruction. This overhead is negligible compared to the IMC scaling, since it applies after the significant reduction performed by the highly-parallel IMC operation.

V. S-DDHR System Design

A. Statistical IMC Model

The statistical distribution of MRAM-based IMC hardware is developed based on circuit-level SPICE simulations, employing MTJ and transistor device models from a foundry technology [36]. Monte Carlo simulations are performed using the foundry-provided statistical models to characterize the variation of MTJ resistance states, which are well-modeled by Gaussian random variables with the form

$$R_P \sim N(\mu_{RP}, \sigma_{RP}^2)$$
$$R_{AP} \sim N(\mu_{RAP}, \sigma_{RAP}^2). \tag{8}$$

From this, it can be shown, using Talyor expansion techniques, that MTJ conductance states can also be modeled using Gaussian random variables, with the form

$$G_P \sim N(\frac{1}{\mu_{RP}}, \frac{\sigma_{RP}^2}{\mu_{RP}^4})$$
$$G_{AP} \sim N(\frac{1}{\mu_{RAP}}, \frac{\sigma_{RAP}^2}{\mu_{RAP}^4}), \tag{9}$$

based on the assumptions $\mu_{RP}, \mu_{RAP} \gg 0$, $\mu_{RP} \gg \sigma_{RP}$, and $\mu_{RAP} \gg \sigma_{RAP}$. For the underlying M-dimensional column computation in the IMC architecture, this enables the accumulated conductance on the bit-line of the MRAM array to be formulated as a new Gaussian random variable, weighted by $+1$ or -1 depending on the element-wise product performed by each set of complementary bit cells. This can be expressed as

$$G_n = \sum_{m=1}^{M} k_{P,m} G_{P,m} + k_{AP,m} G_{AP,m}, \tag{10}$$

where both $G_{P,m}$ and $G_{AP,m}$ are independent and identically distributed (i.i.d.) Gaussian random variables representing bit-cell conductance, $k_{P,m}$ is 1 if the bit-wise, element-wise product corresponds to a conductance of G_P from the complementary bit cells (and 0 otherwise), $k_{AP,m}$ is 1 if the bit-wise, element-wise product corresponds to a conductance of G_{AP} from the complementary bit cells (and 0 otherwise). For an M-dimensional architecture with complementary $k_{P,m}$ and $k_{AP,m}$ values, G_n can take $M+1$ different distributions.

With a statistical model in hand for the underlying MRAM-based IMC architecture, two approaches to S-DDHR are considered:

1) **Parametric S-DDHR (PS-DDHR)**, which appropriately makes the statistical distribution Z of a hardware architecture a function of the computation inputs, e.g., for Gaussian distributed neural network hardware $Z \sim N(\mu(x,\theta), \sigma^2(x,\theta))$.
2) **Approximate S-DDHR (AS-DDHR)**, which makes the approximation that the statistical distribution Z of hardware architecture is fixed, independent of the computation inputs, e.g., for Gaussian-distributed neural network hardware $Z \sim N(\mu_A, \sigma_A^2)$.

AS-DDHR pursues a significant relaxation of the statistical abstractions and the methodologies for their use within computing systems. This is expected to come at the cost of fidelity in statistical modeling and, ultimately, computation accuracy; but it serves as an initial investigation of these trade-offs.

Thus, for PS-DDHR applied to the column computation in the MRAM-based IMC architecture, the resultant G_n is one of the $M+1$ possible random variables set by the input-vector and matrix element bits. G_n is modeled as a Gaussian random variable with mean and variance set by the element bits of W and element bits of x, i.e. $G_n \sim N(\mu(x,W), \sigma^2(x,W))$. Fig. 6 shows the example of a case where $M=4$, leading to 5 possible distributions for G_n, where μ_{GP} and μ_{GAP} represent the means of the P and AP conductance states, respectively. For instance, for the WL data shown, the data stored in the first column results in an average total conductance of $4\mu_{GAP}$, while the second column results in an average total conductance of $3\mu_{GAP} + \mu_{GP}$. On the other hand, for AS-DDHR, the dependency on the matrix W and the input x is removed, yielding the simplification $G_n \sim N(\mu, \sigma^2)$. In this work, σ^2 is selected to be the largest variance among all possible distributions of G_n (i.e., right-most distribution in Fig. 6).

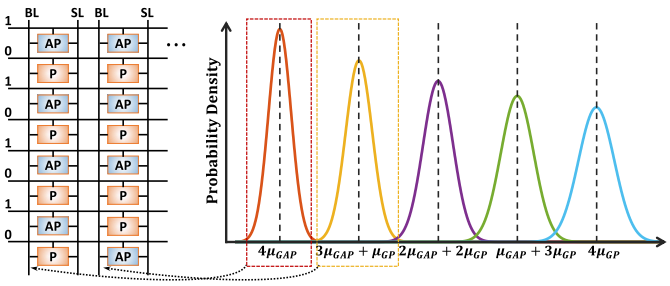


Fig. 6. Statistical model for one column computation with $M=4$.

B. Software Design

We employ PyTorch for neural-network system design. This involves developing custom convolutional (CONV) layers and fully-connected (FC) layers, modeling the IMC computations at the column level. In order to combine the neural network with the statistical conductance-based model of computation from above, operations at each layer need to be mapped to MVMs. Therefore, General Matrix Multiplication (GEMM), as part of Basic Linear Algebra Subprograms (BLAS) specification, is introduced at all layers.

In FC layers, each output is obtained by a linear combination of the inputs and the corresponding weights, which can be regarded as the matrix multiplication illustrated in Fig. 7 (where L is the batch size, M is the number of input features, and N is the number of output features). A tile of the weight matrix can be regarded as mapping to the MRAM IMC with M rows and N columns, and a segment of each row of the input matrix can be regarded as mapping to an input vector, fed to the IMC architecture one at a time. For software acceleration, the sequential input feeding is simply implemented by constructing a matrix out of the input vectors (thereby implementing sequential MVMs as a matrix-matrix multiplication). In general, the IMC architecture is designed to have specific dimensionality and may not fit the entire weight matrix. Hence, the weight matrix, segmented into tiles, can be mapped one tile at a time to the IMC hardware and for proper modeling of the computation.

In CONV layers, the output is computed from a convolution of the input and the filters, where each patch of the input is multiplied by the corresponding filter coefficients and then summed together. The mapping of convolution to an MVM can be achieved by flattening each patch of the input tensor.

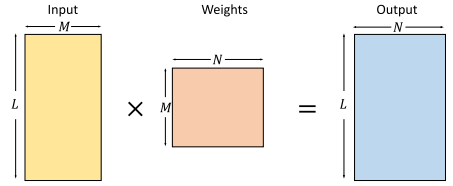


Fig. 7. Illustration of GEMM for FC layers.

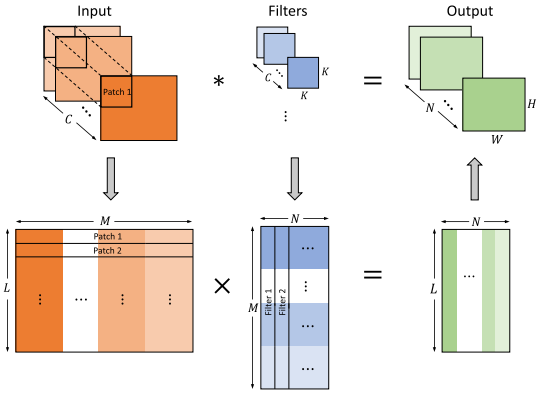


Fig. 8. Illustration of GEMM for CONV layers.

This is shown in Fig. 8 for each input tensor [where $*$ denotes the convolution operation, C is the number of input channels, K is the filter size, N is the number of output channels, W and H are the width and height of the output feature map, M is the length of each flattened patch (determined by the number of input channels and the filter size, i.e., $M = CK^2$), L is the product of W and H (determined by several parameters such as the size of the input image, the size of the filter, stride, etc.)].

Similar to FC layers, the filter matrix can be mapped to the IMC hardware, via segmentation into matrix tiles. Convolution requires edge padding, which is performed by extending the edge pixels. This is found to perform equivalent to zero-padding for the neural networks employed.

For multi-bit implementation, the input matrix grows vertically (in the outer dimension) by the corresponding input-element bit-precision (e.g., $4 \times L$ for 4-bit operation), and the weight/filter matrix will grow horizontally by the corresponding weight bit-precision (e.g, $4 \times N$ for 4-bit operation). The inner dimension is kept the same, which preserves the statistics for each column independent of the matrix growth. A drawback of this is that the required training memory increases with the bit-precision. If necessary, such matrix multiplications can be divided into separate blocks (increasing the training time, due to reduced parallelism across GPU resources).

VI. Results and Analysis

To evaluate S-DDHR, we develop neural networks for MNIST, CIFAR-10, and SVHN classifications. To explore the ability for S-DDHR to restore inference performance relative to a level limited by the neural-network model itself, the network weights and activations are scaled to different bit-precision, with their executions via the multi-bit MRAM-based IMC architecture described above. Further, to explore the ability for S-DDHR to restore inference performance relative to the level of statistical variation of the computation hardware, the variation of MRAM bit-cell is scaled from $1\times$ to $10\times$ the nominal level of the foundry model.

TABLE II

Neural Network Architectures for MNIST, CIFAR-10, and SVHN Datasets

	MNIST	CIFAR-10	SVHN
Layer type	Input: 28×28 grayscale image	Input: 32×32 color image	Input: 32×32 color image
	6-layer	9-layer	9-layer
CONV	$[3 \times 3, 64]$	$[3 \times 3, 128]$	$[3 \times 3, 128]$
	Binarization and Batchnorm	Binarization and Batchnorm	Binarization and Batchnorm
CONV	$[3 \times 3, 64]$	$[3 \times 3, 128]$	$[3 \times 3, 128]$
	2×2 Max-pooling	2×2 Max-pooling	2×2 Max-pooling
	Binarization and Batchnorm	Binarization and Batchnorm	Binarization and Batchnorm
CONV	$[3 \times 3, 128]$	$[3 \times 3, 256]$	$[3 \times 3, 256]$
	Binarization and Batchnorm	Binarization and Batchnorm	Binarization and Batchnorm
CONV	$[3 \times 3, 128]$	$[3 \times 3, 256]$	$[3 \times 3, 256]$
	2×2 Max-pooling	2×2 Max-pooling	2×2 Max-pooling
	Binarization and Batchnorm	Binarization and Batchnorm	Binarization and Batchnorm
CONV	–	$[3 \times 3, 512]$	$[3 \times 3, 256]$
		Binarization and Batchnorm	Binarization and Batchnorm
CONV	–	$[3 \times 3, 512]$	$[3 \times 3, 256]$
		2×2 Max-pooling	2×2 Max-pooling
		Binarization and Batchnorm	Binarization and Batchnorm
FC	1024	1024	1024
	Binarization and Batchnorm	Binarization and Batchnorm	Binarization and Batchnorm
FC	–	1024	1024
		Binarization and Batchnorm	Binarization and Batchnorm
FC	10	10	10
	Batchnorm	Batchnorm	Batchnorm
Softmax	Classifier		

A. MNIST Classification

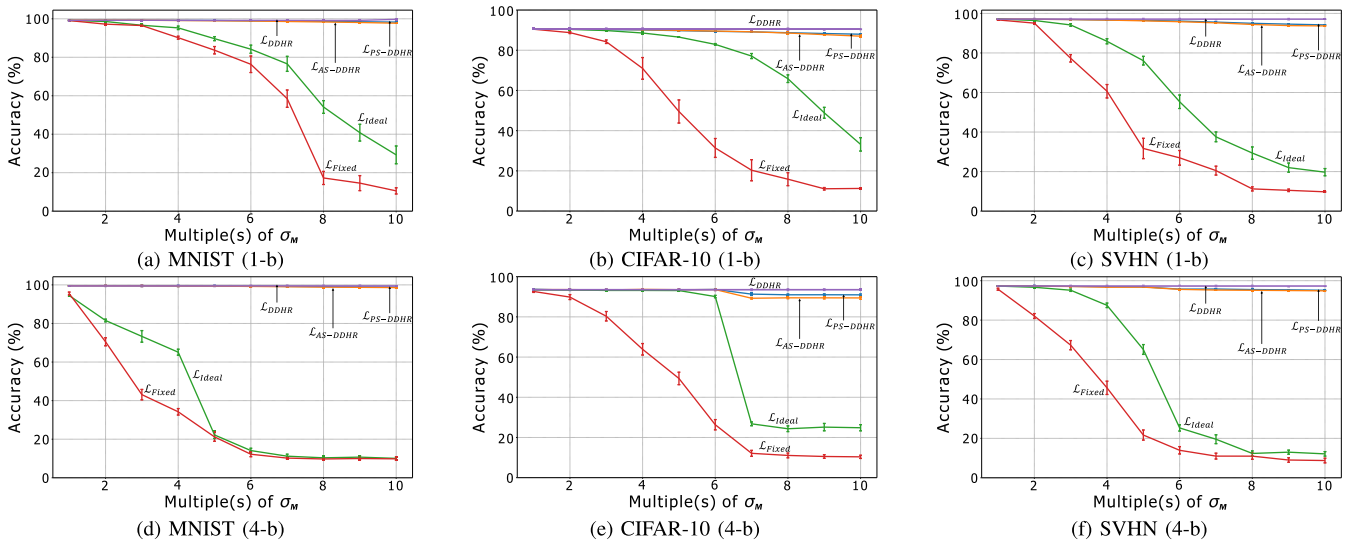
MNIST is an image-classification dataset, consisting of 70,000 images of grayscale handwritten digits with the size of 28×28. 60,000 images are used for training, while the remaining 10,000 are for testing [41]. The neural network architecture is shown in Table II. The input is normalized before feeding into the network. The final layer uses softmax for classification. We use ADAM [42] optimizer for training. The learning rate is set to decay exponentially, where the decaying rate is determined by the provided initial learning rate, the final learning rate, and the number of epochs. We trained the network for 40 epochs with a batch size of 50.

B. CIFAR-10 Classification

CIFAR-10 is an image-classification dataset, consisting of 60,000 32×32 color images, with a training set of 50,000 images and a testing set of 10,000 images [43]. The network architecture used for CIFAR-10 is the same as that used in BinaryNet [24], illustrated in Table II. The input images are randomly cropped, flipped, and normalized before feeding into the network. As before, softmax is used for classification, and ADAM is used as the optimizer for training with a globally decaying learning rate. We trained the network for 200 epochs with a batch size of 128.

C. SVHN Classification

SVHN is an image-classification dataset, consisting of 73,257 images for training, 26,032 images for testing, and additional 531,131 images, where each image has a size of 32×32 [44]. In this work, we combine the training-set images with the additional images to form our training set (e.g., 604,388 images). The network architecture is similar to that used for CIFAR-10 classification, except that the last two convolutional layers are reduced to 256 channels as



(a) MNIST (1-b) (b) CIFAR-10 (1-b) (c) SVHN (1-b)

(d) MNIST (4-b) (e) CIFAR-10 (4-b) (f) SVHN (4-b)

Fig. 9. Performance of S-DDHR with respect to different variation levels for five models. (a)-(c) show the testing results with binarized CNN on MNIST, CIFAR-10 and SVHN datasets, (d)-(f) are the testing results with 4-bit CNN on MNIST, CIFAR-10 and SVHN datasets. Each model is labeled associated with the corresponding curve.

shown in Table II. Preprocessing techniques such as random cropping, flipping, and normalization are applied to the input images before feeding into the network. The optimizer and the classifier are the same as those used for CIFAR-10 dataset. We trained the network for 200 epochs with a batch size of 128.

D. Testing Results

Fig. 9 summarizes the testing results at different levels of MRAM variation σ_M, for the cases of 1-bit and 4-bit weights and activations, respectively. The error bars denote the standard deviation over different samples of the stochastic hardware model. Each curve in the plots shows results for the different loss functions defined, with $\mathcal{L}_{PS\text{-}DDHR}$ and $\mathcal{L}_{AS\text{-}DDHR}$ corresponding to the PS-DDHR and AS-DDHR variants of S-DDHR, respectively. As the statistical variation of the hardware is increased, the baseline approaches, $\mathcal{L}_{Ideal}$ and $\mathcal{L}_{Fixed}$, show degraded accuracy. Specifically, The performance of the model $\mathcal{L}_{Ideal}$, assuming ideal hardware and incorporating no statistics of hardware variations during training, degrades rapidly. The performance of the fixed model $\mathcal{L}_{Fixed}$, which employs one instance of hardware during training and then applies the learned model parameters to a different instance of hardware, shows even more rapid degradation. This can be seen as a case of overfitting, with the instance employed for training leading to erroneous parameter updates away from the instance for testing. On the other hand, DDHR and S-DDHR restore performance. In particular, the DDHR model $\mathcal{L}_{DDHR}$ maintains high performance even at high levels of variation. The variants of S-DDHR restore performance to a level near DDHR, which was previously shown to be limited by the fundamental level of information preservation within the system [8]. However, S-DDHR does this without the complexity of instance-by-instance re-training required with DDHR.

Table III summarizes the results at variation levels of $1 \times \sigma_M$ and $10 \times \sigma_M$, and neural-network weights and activations of 1-bit and 4-bit, for the different S-DDHR loss functions defined. As seen, the case of 4-bit weights and activations follows a similar trend to the binarized (1-b) networks. Again, the performance of the baseline approaches, $\mathcal{L}_{Ideal}$ and $\mathcal{L}_{Fixed}$, show significant degradation at the high levels of variation. In fact, $\mathcal{L}_{Fixed}$ cannot achieve comparable performance even at the nominal variation level in some cases, especially with the higher bit-precision. The DDHR and variants of S-DDHR maintain high performance as expected. Particularly, PS-DDHR and AS-DDHR restore accuracy close to the level of DDHR, with an accuracy drop of less than 3% for PS-DDHR and 3.5% for AS-DDHR at the $10 \times \sigma_M$ variation level. Moreover, we observe that both DDHR and S-DDHR achieve higher accuracy in the 4-bit case than the 1-bit case. This suggests that these approaches not only restore performance, but also preserve the strength of the model executed using statistical hardware. Finally, we see that variations result in more severe accuracy degradation in the CIFAR-10 and SVHN datasets, compared to the MNIST dataset, due to the increased complexity of the task and required neural-network model. Nonetheless, S-DDHR is able to restore accuracy performance in these settings.

To complete the analysis, the weight and activation precision is scaled from 1-8 bits. CIFAR-10 dataset is taken as an example and its result is summarized in Fig. 10, for both $1\times$ and $10\times$ the nominal variation level compared with the variation-free case. S-DDHR is observed to restore performance to a level near that of variation-free hardware, despite large levels of hardware variation. Furthermore, once again, S-DDHR is observed to preserve the strength of the model itself, with higher precision weights and activations leading to increased performance at the level of the variation-free hardware.

VII. Conclusion

In this paper, we extend an approach referred to as Stochastic Data-Driven Hardware Resilience (S-DDHR), which exploits the statistical nature of machine learning models and algorithms to enable the use of statistical hardware, whereby architectural options and circuit trade-offs open up for enhancing energy efficiency and throughput. A critical

TABLE III

COMPARISON OF DIFFERENT MODELS WITH 1-BIT AND 4-BIT PRECISION ON MNIST, CIFAR-10, AND SVHN DATASETS AT DIFFERENT LEVEL OF HARDWARE VARIATIONS

Models	MNIST				CIFAR-10				SVHN			
	1-b ($1x\sigma_M$)	1-b ($10x\sigma_M$)	4-b ($1x\sigma_M$)	4-b ($10x\sigma_M$)	1-b ($1x\sigma_M$)	1-b ($10x\sigma_M$)	4-b ($1x\sigma_M$)	4-b ($10x\sigma_M$)	1-b ($1x\sigma_M$)	1-b ($10x\sigma_M$)	4-b ($1x\sigma_M$)	4-b ($10x\sigma_M$)
$\mathcal{L}_{Ideal}$	99.13%	29.26%	99.35%	10.06%	90.60%	33.18%	93.38%	24.84%	97.14%	19.70%	97.26%	12.14%
$\mathcal{L}_{Fixed}$	99.07%	10.54%	94.46%	9.82%	90.45%	11.25%	92.64%	10.36%	96.70%	9.76%	95.69%	8.76%
$\mathcal{L}_{DDHR}$	99.21%	99.17%	99.50%	99.41%	90.62%	90.40%	93.49%	93.48%	97.13%	97.07%	97.32%	97.25%
$\mathcal{L}_{AS\text{-}DDHR}$	99.27%	97.68%	99.56%	98.59%	90.67%	87.42%	93.45%	90.02%	97.15%	93.67%	97.35%	94.86%
$\mathcal{L}_{PS\text{-}DDHR}$	99.25%	98.27%	99.55%	98.84%	90.68%	88.11%	93.54%	90.96%	97.14%	94.19%	97.39%	95.04%

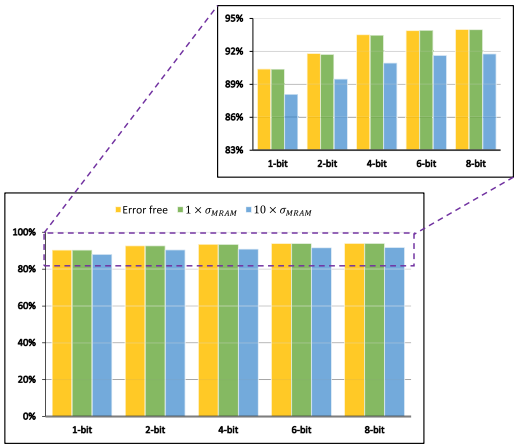


Fig. 10. Comparison of different precision levels.

aspect of S-DDHR is constructing statistical models that represent the distribution corresponding to hardware computations, rather than just a sample from the distribution. This enables a more generalized design methodology where applications can be robustly deployment on different instances of hardware. S-DDHR also involves forming abstractions of the statistical hardware computations that can be propagated to the application-design level in software, and which can be appropriately composed to represent the statistics of complex computations mapped to the hardware. S-DDHR is demonstrated for an MRAM-based IMC architecture, which explicitly leverages SNR trade-offs for energy-aggressive and throughput-aggressive operations. Implementations on other architectures, which require exploring the corresponding statistical properties, are being pursued as on-going work. Formation of a statistical model of the fundamental computation is demonstrated, with abstractions to a domain-specific design language (PyTorch), where bit-scalable matrix-vector multiplication operations are composed. Using circuit-level statistical data for a foundry technology, the ability for S-DDHR to overcome hardware variations and retain inference performance in neural networks for MNIST, CIFAR-10 and SVHN datasets are demonstrated.

ACKNOWLEDGMENT

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1605.07678

[2] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5687–5695.

[3] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our glass: An energy and heat characterization of Google glass," in *Proc. 5th Asia–Pacific Workshop Syst.*, 2014, p. 10.

[4] N. R. Shanbhag, N. Verma, Y. Kim, A. D. Patil, and L. R. Varshney, "Shannon-inspired statistical computing for the nanoscale era," *Proc. IEEE*, vol. 107, no. 1, pp. 90–107, Jan. 2019.

[5] T. Moy, W. Rieutort-Louis, S. Wagner, J. C. Sturm, and N. Verma, "A thin-film, large-area sensing and compression system for image detection," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 11, pp. 1833–1844, Nov. 2016.

[6] 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.

[7] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

[8] N. Verma, K. H. Lee, K. J. Jang, and A. Shoeb, "Enabling system-level platform resilience through embedded data-driven inference capabilities in electronic devices," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 5285–5288.

[9] Z. Wang, R. E. Schapire, and N. Verma, "Error adaptive classifier boosting (EACB): Leveraging data-driven training towards hardware resilience for signal inference," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1136–1145, Apr. 2015.

[10] 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.

[11] B. Zhang, L.-Y. Chen, and N. Verma, "Stochastic data-driven hardware resilience to efficiently train inference models for stochastic hardware implementations," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 1388–1392.

[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[13] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.

[14] K.-W. Chang and T.-S. Chang, "VWA: Hardware efficient vectorwise accelerator for convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 145–154, Jan. 2020.

[15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[16] N. Verma *et al.*, "In-memory computing: Advances and prospects," *IEEE Solid State Circuits Mag.*, vol. 11, no. 3, pp. 43–55, Aug. 2019.

[17] M. Kang, Y. Kim, A. D. Patil, and N. R. Shanbhag, "Deep in-memory architectures for machine learning–accuracy versus efficiency trade-offs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 5, pp. 1627–1639, May 2020.

[18] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *Proc. IEEE Symp. VLSI Circuits (VLSI-Circuits)*, Jun. 2016, pp. 1–2.

[19] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 490–492.

[20] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Design Automat. Conf.*, Jun. 2003, pp. 338–342.

[21] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, NY, USA: McGraw-Hill, 2002.

[22] M. Haselman and S. Hauck, "The future of integrated circuits: A survey of nanoelectronics," *Proc. IEEE*, vol. 98, no. 1, pp. 11–38, Jan. 2010.

[23] J. Lee, S. Kang, J. Lee, D. Shin, D. Han, and H.-J. Yoo, "The hardware and algorithm co-design for energy-efficient DNN processor on edge/mobile devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3458–3470, Oct. 2020.

[24] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016, *arXiv:1602.02830*. [Online]. Available: http://arxiv.org/abs/1602.02830

[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[26] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: http://arxiv.org/abs/1612.01064

[27] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. Cambridge, MA, USA: MIT Press, 2012.

[28] Y. Tang, J. Zhang, and N. Verma, "Scaling up in-memory-computing classifiers via boosted feature subsets in banked architectures," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 3, pp. 477–481, Mar. 2019.

[29] J. Zhang and N. Verma, "An in-memory-computing DNN achieving 700 TOPS/W and 6 TOPS/mm^2 in 130-nm CMOS," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 358–366, Jun. 2019.

[30] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 490–492.

[31] Y. Li and F. Liu, "Whiteout: Gaussian adaptive noise regularization in deep neural networks," 2016, *arXiv:1612.01490*. [Online]. Available: http://arxiv.org/abs/1612.01490

[32] G. Kang, J. Li, and D. Tao, "Shakeout: A new regularized deep neural network training scheme," in *Proc. AAAI*, 2016, pp. 1751–1757.

[33] C. Wang and J. C. Principe, "Training neural networks with additive noise in the desired signal," *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1511–1517, Nov. 1999.

[34] H. Noh, T. You, J. Mun, and B. Han, "Regularizing deep neural networks by noise: Its interpretation and optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5109–5118.

[35] M. Qin and D. Vucinic, "Training recurrent neural networks against noisy computations during inference," 2018, *arXiv:1807.06555*. [Online]. Available: http://arxiv.org/abs/1807.06555

[36] D. Shum *et al.*, "CMOS-embedded STT-MRAM arrays in 2x nm nodes for GP-MCU applications," in *Proc. Symp. VLSI Technol.*, Jun. 2017, pp. T208–T209.

[37] Z. He, S. Angizi, and D. Fan, "Exploring STT-MRAM based in-memory computing paradigm with application of image edge extraction," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 439–446.

[38] A. V. Khvalkovskiy *et al.*, "Basic principles of STT-MRAM cell operation in memory arrays," *J. Phys. D, Appl. Phys.*, vol. 46, no. 7, Feb. 2013, Art. no. 074001.

[39] L. Wu, M. Taouil, S. Rao, E. J. Marinissen, and S. Hamdioui, "Survey on STT-MRAM testing: Failure mechanisms, fault models, and tests," 2020, *arXiv:2001.05463*. [Online]. Available: http://arxiv.org/abs/2001.05463

[40] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled RRAM architecture," in *IEDM Tech. Dig.*, Dec. 2019, pp. 14.4.1–14.4.4.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[43] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[44] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS*, Jan. 2011, pp. 1–9.

Bonan Zhang (Student Member, IEEE) received the B.Eng. degree in electrical and computer engineering from McGill University, Montreal, QC, Canada, in 2017, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2019, where he is currently pursuing the Ph.D. degree.

His current research interests include in-memory computing systems with emerging technologies for signal processing and machine learning applications, at both algorithm and hardware levels. He received the Analog Devices Outstanding Student Designer Award in 2019.

Lung-Yen Chen (Student Member, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, in 2011, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Electrical Engineering.

His research interests include digital architectures and circuit design for in-memory computing and multimedia applications, with emerging technologies, such as magnetic random-access memory and 3D integration.

Naveen Verma (Member, IEEE) received the B.A.Sc. degree in electrical and computer engineering from UBC, Vancouver, BC, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from MIT in 2005 and 2009, respectively.

Since July 2009, he has been with Princeton University, where he is currently the Director of the Keller Center for Education in Innovation and Entrepreneurship and a Professor of electrical engineering. His research interests include advanced sensing systems, exploring how systems for learning, inference, and action planning can be enhanced by algorithms that exploit new sensing and computing technologies. This includes research on large-area, flexible sensors, energy-efficient statistical-computing architectures and circuits, and machine learning and statistical-signal-processing algorithms. He has served as a Distinguished Lecturer for the IEEE Solid-State Circuits Society, and on the technical program committees for ISSCC, VLSI Symposium, DATE, and IEEE Signal-Processing Society (DISPS). He was a recipient or co-recipient of the 2006 DAC/ISSCC Student Design Contest Award, the 2008 ISSCC Jack Kilby Paper Award, the 2012 Alfred Rheinstein Junior Faculty Award, the 2013 NSF CAREER Award, the 2013 Intel Early Career Award, the 2013 Walter C. Johnson Prize for Teaching Excellence, the 2013 VLSI Symposium Best Student Paper Award, the 2014 AFOSR Young Investigator Award, the 2015 Princeton Engineering Council Excellence in Teaching Award, and the 2015 IEEE TRANSACTIONS ON COMPONENTS, PACKAGING AND MANUFACTURING TECHNOLOGY (CPMT) Best Paper Award.