STOCHASTIC DATA-DRIVEN HARDWARE RESILIENCE TO EFFICIENTLY TRAIN INFERENCE MODELS FOR STOCHASTIC HARDWARE IMPLEMENTATIONS

Bonan Zhang, Lung-Yen Chen, Naveen Verma

Department of Electrical Engineering, Princeton University, Princeton, NJ 08544

ABSTRACT

Machine-learning algorithms are being employed in an increasing range of applications, spanning high-performance and energy-constrained platforms. It has been noted that the statistical nature of the algorithms can open up new opportunities for throughput and energy efficiency, by moving hardware into design regimes not limited to deterministic models of computation. This work aims to enable high accuracy in machine-learning inference systems, where computations are substantially affected by hardware variability. Previous work has overcome this by training inference model parameters for a particular instance of variation-affected hardware. Here, training is instead performed for the distribution of variation-affected hardware, eliminating the need for instance-by-instance training. The approach is referred to as Stochastic Data-Driven Hardware Resilience (S-DDHR), and it is demonstrated for an in-memory-computing architecture based on magnetoresistive random-access memory (MRAM). S-DDHR successfully address different samples of stochastic hardware, which would otherwise suffer degraded performance due to hardware variability.

Index Terms— Fault tolerance, In-memory Computing, Machine Learning, Statistical Computing.

1. INTRODUCTION

Traditional system design has been based on deterministic models of hardware operation. But, hardware operation is more naturally stochastic, both due to random process variations and fundamental noise sources in electronic devices. The focus on deterministic models has required introducing design margin at every layer of the system stack, to form component abstractions where the true stochastic behaviors can be hidden. Such margining imposes resource overheads (energy, throughput, area), which are becoming increasingly severe as inherent stochastic behaviors become more prominent with technology scaling [1].

In fact, emerging device technologies and emerging compute architectures, which raise the potential for substantial gains in resource efficiency, are often more significantly affected by stochastic behaviors. For instance, emerging nonvolatile memory devices such as RRAM and MRAM offer high levels of density and low levels of leakage power; however, they exhibit degraded separation between storage states and high variability of the individual states [2][3][4]. Similarly, emerging architectures for in-memory computing have the potential to overcome memory bottlenecks, by accessing a computational result over many stored bits, rather than individual bits themselves; however, such computation typically requires increasing dynamic range, degrading computational SNR under fixed signal-swing constraints [5].

Both the increasing overheads due to margining and the new opportunities opened by emerging device technologies and architectures have motivated a departure from deterministic models of hardware operation to stochastic models of hardware operation [6][7]. This paper refers to stochastic hardware as that where inherent random variations and noise are not hidden in the abstraction models. Of course, stochastic hardware has important implications on the computations. Fortunately, emerging workloads from the domains of machine learning and statistical signal processing can readily incorporate the effects of stochastic hardware within the associated algorithms, enabling minimal overheads and/or limitations on the algorithms themselves.

As an example, previous work resulted in an approach referred to as Data-Driven Hardware Resilience (DDHR) [8], where training of inference-model parameters in a machine-learning system makes use of a sample of stochastic hardware, so that optimal parameter fitting is achieved not only to the statistics of application data but also to that sample of stochastic hardware. This work extends DDHR to stochastic DDHR (S-DDHR), where a stochastic model of hardware is used so that optimal parameter fitting is achieved not for a sample of stochastic hardware but rather for the distribution of stochastic hardware. As a result, the inference-model parameters can be trained once for all samples of stochastic hardware. S-DDHR is a general approach, applicable to a range of stochastic hardware architectures, but it is demonstrated here for a binarized convolutional neural network (BNN) applied to an image-classification task on the CIFAR-10 dataset [9], where it is presumed that the BNN is implemented using an MRAM-based in-memory-computing architecture. We show how stochastic hardware can be modeled and employed in the training process, based on the underlying hardware variability sources. We also show how parametric and approximate parametric models of the stochastic hardware can be formed for neural-network systems, simplifying both the understand-

ing and computations involved in training.

2. BACKGROUND

2.1. Data-Driven Hardware Resilience (DDHR)

Several approaches have been proposed for handling stochastic hardware [10][11]. The presented approach is most closely related to DDHR [8], which utilizes the outputs from an instance of non-ideal, variation-affected hardware as training data for a machine-learning classifier. This results in an *error-aware model* for inference, where parameter fitting to the training data is optimally adapted to the hardware. In [12], DDHR is extended to permit non-ideal, variation-affected hardware for the classifier itself, via an approach referred to as Error-Adaptive Classifier Boosting (EACB), by exploiting iterative training in Adaptive Boosting [13]. The primary drawback of these approaches is that a particular instance of non-ideal, variation-affected hardware is employed for training, invoking training complexity for each new instance. Here, S-DDHR is proposed, employing a stochastic distribution to represent the non-ideal, variation-affected hardware, such that training can be performed once for all instances of the hardware (i.e., samples of the stochastic distribution).

2.2. Stochastic Training

S-DDHR is developed for neural networks, employing the back-propagation algorithm and its extension to stochastic training. Stochastic training involves injecting noise in the training process. Previous work regards it as a regularization technique. For instance, [14] develops a *whiteout* method to inject additive Gaussian noise to neural network training, while [15] proposes a multiplicative Bernoulli noise injection approach applied to input and hidden nodes during training. Recent work also begins to explore stochastic training to address compute noise in a long short-term memory (LSTM) network [16]. While motivated by neuromorphic architectures, [16] introduces a generic noise model rather than attempting to model the practical stochastic noise and variability sources from a hardware implementation. Here, we exploit stochastic training together with a noise model derived from the stochastic hardware. We further form parametric and approximate models of the hardware, aiming to enhance designer understanding and training efficiency for stochastic hardware.

3. OVERVIEW OF S-DDHR

Figure 1 illustrates the proposed approach of S-DDHR. Hardware parameters exhibiting variability are represented by a random variable Z. A particular instance of hardware is thus represented by a sample z_i. We point out that in general, even a given instance of hardware can exhibit stochastic noise (i.e., random fluctuations in the compute outputs). In this case, the model of an instance of hardware could be extended to include an additional random variable. However, in the emerging technologies and architectures of immediate interest, variation sources between instances of hardware, and not random fluctuations within a given instance, are dominant [17][18]. In this case, S-DDHR, applied to the back-propagation algorithm for neural network training, utilizes a model of stochastic hardware employing Z for forward computation of the loss function $\mathcal{L}$. The predicted output then has the form

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

Model parameters θ (e.g., weights) are then updated through back-propagation gradient computation, employing the expected value $E[Z]$. This is illustrated in algorithm 1 for one epoch. An output from the instance of hardware for a given input x_{test} during the testing process is then represented as

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

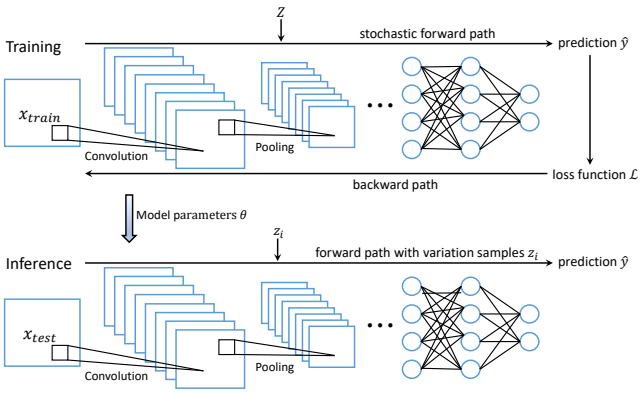

Fig. 1. System modeling employed for S-DDHR.

Algorithm 1 Stochastic Training for Neural Network

Require: model parameters θ, learning rate η, input to j^{th} layer a_j ($x_{train} = a_0$), training labels y, no. of layers J, random variable Z representing stochastic hardware, loss function $\mathcal{L}$, forward function $f(\cdot)$, and $f_j(\cdot)$ for j^{th} layer

Ensure:

▷ **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

▷ **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**

$\theta_j \leftarrow \theta_j - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}$

end for

With this framing, S-DDHR can be described and compared with previous training approaches, via the loss functions employed. To proceed with analysis of S-DDHR, the following loss functions are defined, where $\mathcal{L}^t$ represents a loss function for the t^{th} training sample (the loss function employed as an example is the cross-entropy loss $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^t$, with T being the number of training samples, and C being the number of classes):

- Standard training assumes an ideal, variability-free

model of the hardware, thus the cross-entropy loss function, independent of Z, is

$$\mathcal{L}^t_{Ideal} = -\sum_{i=1}^{C} y^t_i \log f(x^t, \theta). \quad (3)$$

- DDHR assumes training on each instance of hardware, thus the loss function, employing a sample z_i corresponding to the variability of that hardware instance is

$$\mathcal{L}^t_{DDHR} = -\sum_{i=1}^{C} y^t_i \log f(x^t, \theta, z_i). \quad (4)$$

- While DDHR is expected to give high accuracy by accounting for the specific variation of the hardware instance employed, it has the cost of instance-by-instance training. As an alternative, a fixed instance of hardware could be used, represented by $z_\emptyset$, different from that used for testing. The loss function is

$$\mathcal{L}^t_{Fixed} = -\sum_{i=1}^{C} y^t_i \log f(x^t, \theta, z_\emptyset). \quad (5)$$

- S-DDHR employs a stochastic model of the hardware variation for training. Thus, the loss function, employing the random variable Z, is

$$\mathcal{L}^t_{S\text{-}DDHR} = -\sum_{i=1}^{C} y^t_i \log f(x^t, \theta, Z). \quad (6)$$

As described in the following section, Z is derived from practical variability sources in a hardware implementation. In general, this can lead to complex models of the stochastic hardware, thus complicating training. Therefore, we also consider two variants of S-DDHR:

- Parametric S-DDHR (PS-DDHR), employs an analytical distribution for Z, derived from the structure of the hardware and the distributions of the underlying variability sources. For the example considered, this corresponds to a Gaussian distribution whose mean and variance depend on the input and model parameters, and the loss function $\mathcal{L}_{PS\text{-}DDHR}$ has the same form with equation 6, with $Z \sim N(\mu(x,\theta), \sigma^2(x,\theta))$.
- While PS-DDHR can simplify understanding of the stochastic hardware and also computations during the training process by employing an analytical distribution, the distribution parameters (mean, variance) must be computed for each layer. Approximate S-DDHR (AS-DDHR), employs fixed distribution parameters to further simplify computations. The loss function $\mathcal{L}_{AS\text{-}DDHR}$, with the form of Equation 6, now has $Z \sim N(\mu_A, \sigma^2_A)$.

4. EXPERIMENTS AND ANALYSIS

4.1. Experiment Setup

To evaluate S-DDHR, we focus on an image classification task based on the CIFAR-10 dataset [9]. For the hardware implementation, both an emerging device technology and an emerging architecture are considered, namely MRAM-based in-memory computing. The system implements a VGG-style [19] binarized CNN (BNN) based on [20], consisting of 9 layers (i.e. 6 conv. layers and 3 fully-conn. layers), with training and testing performed using PyTorch [21].

4.2. MRAM-based In-Memory Computing

Figure 2 shows an in-memory-computing architecture, particularly for MRAM, consisting of M rows and N columns. While traditional memory accesses involve one row at a time, in-memory computing involves accessing a computational result on the bit lines BL_n for all rows at once, thus amortizing the access time and energy. MRAM stores data as the resistive/conductive state of a two-terminal device in the bit cell, corresponding to parallel (P) R_P/G_P and anit-parallel (AP) R_{AP}/G_{AP} resistance/conductance, set by the spin-polarization current [3]. The in-memory-computing architecture considered performs multiplication between weight matrices with binary elements ($W_{m,n} \in \{-1, 1\}$) and input-activation vectors with binary elements ($a_m \in \{-1, 1\}$), as required for BNNs. Multiplication is performed by applying vector elements on two complementary word lines (WL_m/$\overline{WL}_m$) and storing matrix elements in the two complementary bit cells, to implement an XNOR operation. The output from the two bit cells is thus a net resistance/conductance of R_P/G_P or R_{AP}/G_{AP}, and a computed pre-activation element u_n is thus the total conductance across a bit-line/source-line pair (BL_n/SL_n). This is then followed by a comparator, to apply an activation function, obtaining a binary output activation $v_n \in \{-1, 1\}$, as required in a BNN.

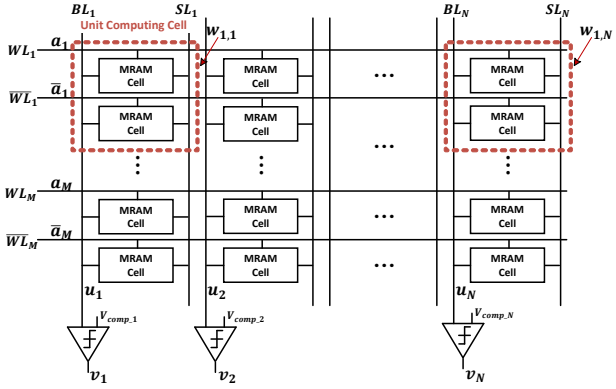

Fig. 2. MRAM-based in-memory computing architecture.

4.3. Stochastic Model

MRAM resistances exhibit variability, expressed through transistor-level (SPICE/SPECTRE) simulations based on a foundry-provided device model (GlobalFoundries 22nm FD-SOI process), where MRAM resistances are modeled as a Gaussian random variables, i.e. $R_P \sim N(\mu_P, \sigma^2_P)$, $R_{AP} \sim N(\mu_{AP}, \sigma^2_{AP})$. Assuming $\mu_P, \mu_{AP} >> 0$, the conductances are thus: $G_P \sim N(\frac{1}{\mu_P}, \frac{\sigma^2_P}{\mu^4_P})$, $G_{AP} \sim N(\frac{1}{\mu_{AP}}, \frac{\sigma^2_{AP}}{\mu^4_{AP}})$. For a BNN layer, the required matrix-vector multiplication (MVM)

$u = Wa$, can then be modeled by the stochastic conductances, where each output element u_n is represented by the total column conductance $G_n = p_{1n} \times G_P + p_{-1n} \times G_{AP}$, with p_{1n}/p_{-1n} being the number of element-wise product terms equal to 1/-1 in the n^{th} column.

For PS-DDHR, a parametric stochastic model of the computation can now be formed. With M rows, p_{1n}/p_{-1n} can take M+1 different values, determined by the input-activation vector a and the weight matrix W. Hence, we see that G_n is itself a Gaussian random variable, but whose mean and variance are set to one of M+1 different values, as shown in Figure 3 for the simple case of $M = 4$. Thus, the computation of each output-vector element u_n in the stochastic MVM computation can simply be modeled using a Gaussian random variable $G_n \sim N(\mu(a, W), \sigma^2(a, W))$, as a function of the available a and W for forward computation during training.

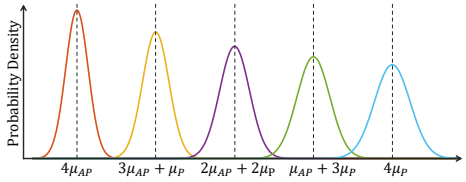

Fig. 3. Distributions of column conductance G_n for $M = 4$.

For AS-DDHR, we simply model each output element u_n by a Gaussian random variable $G_n \sim N(\mu_A, \sigma_A^2)$, where the fixed σ_A^2 is taken to be the largest variance among the $M+1$ actual distributions. While this has only modest impact on easing training complexity, it can have substantial impact on easing design methodology for an MRAM in-memory-computing architecture, in terms of specifying device-level variability requirements.

4.4. Results and Analysis

To analyze the proposed S-DDHR approach, involving a stochastic model of the hardware, the variability of the MRAM R_P/R_{AP} states is scaled and the classification testing accuracy of the different training approaches is observed. Figure 4 shows the accuracy achieved for CIFAR-10 image classification over 200 training epochs, for different multiples of a normalized level of MRAM variability σ_{MRAM} (actual foundry data used for MRAM variability has been normalized to protect confidentiality).

Figure 5 summarizes the converged classification accuracy versus MRAM variability, with error bars showing the standard deviation over five random samples of the stochastic hardaware model. As the variation σ increases, we see that training with $\mathcal{L}_{DDHR}$ maintains high performance as expected, achieving an accuracy greater than 90%, but, unfortunately, it incurs instance-by-instance training cost. On the other hand, training without accounting for stochastic hardware $\mathcal{L}_{Ideal}$ or using a fixed sample of stochastic hardware $\mathcal{L}_{Fixed}$, avoids instance-by-instance training but exhibits substantially degraded accuracy, since training is done for a model that is effectively different than used for testing.

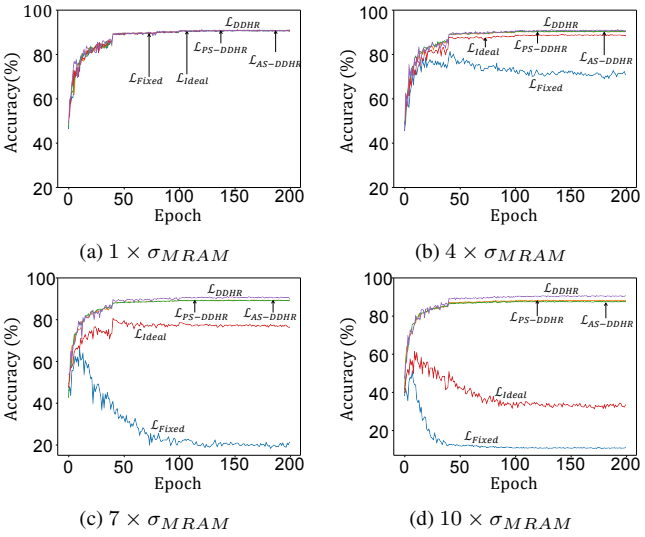

Fig. 4. Convergence under different variation levels.

Finally, training with both $\mathcal{L}_{PS\text{-}DDHR}$ and $\mathcal{L}_{AS\text{-}DDHR}$ avoid instance-by-instance training and yet also maintain high accuracy, with a slight drop as the variation level increases. These results are quantified in table 1, showing the accuracy at the $10 \times \sigma_{MRAM}$ level. As seen S-DDHR exhibits significant ability to tolerate hardware variability, with less than 3% degradation in accuracy from variability-free hardware.

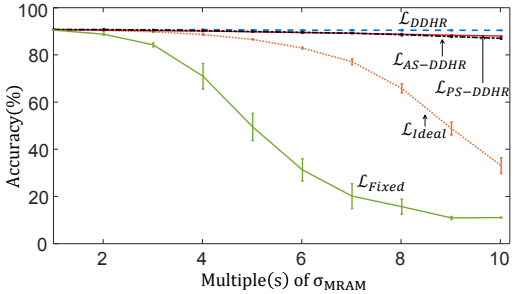

Fig. 5. Accuracy of different models vs. variation levels.

Table 1. Accuracy at $10 \times \sigma_{MRAM}$

Model	$\mathcal{L}_{DDHR}$	$\mathcal{L}_{Ideal}$	$\mathcal{L}_{Fixed}$	$\mathcal{L}_{PS\text{-}DDHR}$	$\mathcal{L}_{AS\text{-}DDHR}$
Accuracy	90.40%	33.18%	11.25%	88.11%	87.42%

5. CONCLUSION

This paper presents the approach of S-DDHR, and demonstrates it for neural-network training. S-DDHR involves forming a stochastic model to represent hardware variability, and employing this for stochastic training of inference-model parameters. Presuming neural-network implementation based on MRAM in-memory computing, the approach to forming a stochastic hardware model based on underlying variability sources is illustrated. Using this, the efficacy of S-DDHR is shown through simulations, demonstrating performance near the level of a system trained for a particular instance of variation-affected hardware, yet without requiring instance-by-instance training cost.

6. REFERENCES

[1] M. Horowitz, E. Alon, D. Patil, S. Naffziger, Rajesh Kumar, and K. Bernstein, "Scaling, power, and the future of CMOS," in *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.*, Dec 2005, pp. 7 pp.–15.

[2] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec 2010.

[3] K. C. Chun, H. Zhao, J. D. Harms, T. Kim, J. Wang, and C. H. Kim, "A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 2, pp. 598–610, Feb 2013.

[4] A V Khvalkovskiy, D Apalkov, S Watts, R Chepulskii, R S Beach, A Ong, X Tang, A Driskill-Smith, W H Butler, P B Visscher, D Lottis, E Chen, V Nikitin, and M Krounbi, "Basic principles of STT-MRAM cell operation in memory arrays," *Journal of Physics D: Applied Physics*, vol. 46, no. 7, pp. 074001, 2013.

[5] J. Zhang, Z. Wang, and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, April 2017.

[6] Armin Alaghi and John P Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 92, 2013.

[7] N. R. Shanbhag, N. Verma, Y. Kim, A. D. Patil, and L. R. Varshney, "Shannon-Inspired Statistical Computing for the Nanoscale Era," *Proceedings of the IEEE*, pp. 1–18, 2018.

[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 *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2012, pp. 5285–5288.

[9] Alex Krizhevsky and Geoffrey Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.

[10] S. K. Gonugondla, B. Shim, and N. R. Shanbhag, "Perfect error compensation via algorithmic error cancellation," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 966–970.

[11] F. N. Taher, J. Callenes-Sloan, and B. C. Schafer, "A machine learning based hard fault recuperation model for approximate hardware accelerators," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018, pp. 1–6.

[12] Z. Wang, R. Schapire, and N. Verma, "Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 3884–3888.

[13] Robert E Schapire and Yoav Freund, *Boosting: Foundations and algorithms*, MIT press, 2012.

[14] Yinan Li and Fang Liu, "Whiteout: Gaussian Adaptive Noise Regularization in FeedForward Neural Networks," *arXiv preprint arXiv:1612.01490*, 2016.

[15] Guoliang Kang, Jun Li, and Dacheng Tao, "Shakeout: A New Regularized Deep Neural Network Training Scheme.," in *AAAI*, 2016, pp. 1751–1757.

[16] Minghai Qin and Dejan Vucinic, "Training Recurrent Neural Networks against Noisy Computations during Inference," *arXiv preprint arXiv:1807.06555*, 2018.

[17] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, June 2003, pp. 338–342.

[18] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 2–13.

[19] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[21] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan, "PyTorch," `https://github.com/pytorch`, 2017.