Towards Resilient Analog In-Memory Deep Learning via Data Layout Re-Organization

Muhammad Rashedul Haq Rashed*, Amro Awad†, Sumit Kumar Jha‡, Rickard Ewetz*
*Department of ECE, University of Central Florida, Orlando, FL, USA
†Department of ECE, North Carolina State University, Raleigh, NC, USA
‡CS Department, University of Texas at San Antonio, San Antonio, TX, USA
rashed09@knights.ucf.edu,ajawad@ncsu.edu,sumit.jha@utsa.edu,rickard.ewetz@ucf.edu

ABSTRACT

Processing in-memory paves the way for neural network inference engines. An arising challenge is to develop the software/hardware interface to automatically compile deep learning models onto in-memory computing platforms. In this paper, we observe that the data layout organization of a deep neural network (DNN) model directly impacts the model's classification accuracy. This stems from that the *resistive parasitics* within a crossbar introduces a dependency between the *matrix data* and the *precision* of the analog computation. To minimize the impact of the parasitics, we first perform a case study to understand the underlying matrix properties that result in computation with low and high precision, respectively. Next, we propose the XORG framework that performs data layout organization for DNNs deployed on in-memory computing platforms. The data layout organization improves precision by optimizing the weight matrix to crossbar assignments at compile time. The experimental results show that the XORG framework improves precision with up to 3.2X and 31% on the average. When accelerating DNNs using XORG, the write bit-accuracy requirements are relaxed with 1-bit and the robustness to random telegraph noise (RTN) is improved.

ACM Reference Format:
Muhammad Rashedul Haq Rashed*, Amro Awad†, Sumit Kumar Jha‡, Rickard Ewetz*. 2022. Towards Resilient Analog In-Memory Deep Learning via Data Layout Re-Organization. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530532

1 INTRODUCTION

Deep neural network (DNN) models have surpassed human-level capabilities for several cognitive tasks such as image classification and object detection [10]. However, it is a daunting task to execute DNNs on von-Neumann based computing systems. Mainly, due to the power-hungry and bandwidth-limited data movement between the computing and memory units [17]. In-memory computing is a promising contender for conventional computing in that computations occur in memory, and thus minimizes data movements. Many recent studies have shown that deep learning workloads can benefit significantly from in-memory computing, and how in-memory computing makes deep learning at edge devices more plausible [6].

While in-memory computing systems are promising, they face several challenges that can limit their wide adoption. Perhaps one of the most pressing challenges is the reliability and resilience against analog errors [1, 5]. Due to non-linear device characteristics of memristor cells, in addition to noise effects, in-memory computing operations can suffer from unpredictable accuracy losses due to voltage IR-drops across crossbar metal wires. To overcome this issue for deep learning applications, several prior work explored techniques to improve reliability by increasing the model size [13] or injecting noise in the training process [3]. Recent work also explored algorithms to convert matrices into appropriate memristor conductance values while accounting for the resistive parasitics [4, 11, 15, 16]. Such techniques compensate for the voltage IR-drop over the parasitics by tuning the memristors to be more conductive.

While the techniques for tuning conductance can significantly improve reliability, we observe that the matrices' values play a major role in the accuracy of computations, even when such techniques are applied. In particular, we observe that the precision for different matrices could vary significantly (up to 5.9X). Such accuracy difference stems from ignoring the dependency between the *matrix data* and the *precision* of the analog computation. For instance, we observe that a very poor precision consistently appears when matrix elements with large magnitudes are mapped to the top-right corner of a crossbar. One reason behinds this is that it becomes *impossible* to tune the corresponding memristor devices to be sufficiently conductive for compensating the voltage IR-drop over the resistive paratisitcs. Therefore, in this paper, we investigate the root causes for such errors, and propose novel matrix mapping techniques that can improve accuracy.

In this paper, we propose XORG, a framework that performs resilient data layout organization for DNN models deployed on in-memory computing platforms. The data layout organization aims to improve precision by modifying the matrix data to crossbar assignments at compile time. Specifically, the data is organized to map matrix elements with large (small) magnitude to the top-right (bottom-left) crossbar corner. The experimental results demonstrate that XORG improves the analog precision with up to 3.2X and with 31% on the average. When accelerating DNNs using XORG, the device write-bit accuracy requirements are relaxed with 1-bit and the robustness to RTN is improved. The improvements come at the modest expense of 6% longer compile time.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530532

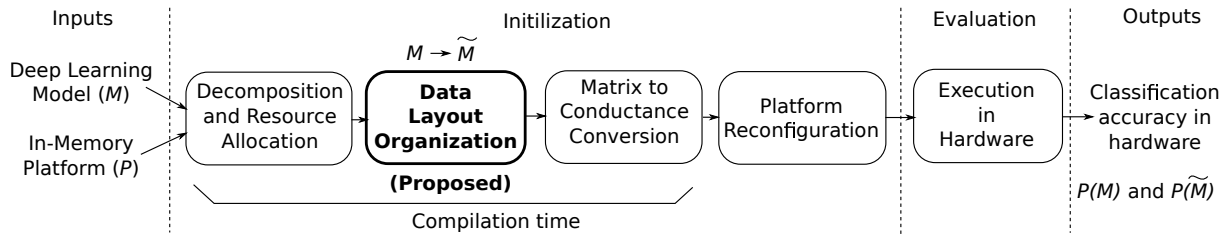

Figure 1: Flow for compiling a DNN model M to an in-memory computing platform P.

2 PRELIMINARIES

In this section, we first introduce the flow for compiling DNN models to in-memory computing platforms. Next, we provide motivation and problem statement.

2.1 Flow for compiling DNN models to in-memory computing platforms

The flow for compiling a DNN model to in-memory computing platform for inference is shown in Figure 1. The input to the framework is a deep learning model M and an in-memory computing platform P. The output is the classification accuracy in hardware $P(M)$.

The first step is decomposition and resource allocation [7]. This involves decomposing the deep learning model M into matrix-vector multiplication operations [14, 20]. Next, the matrices are further partitioned and assigned to specific crossbars within the platform [8]. The assignment is performed to minimize data movement between the crossbar accelerators while considering the platform interconnect topology.

The second step is the proposed data layout organization. This involves modifying the layout of the data within the deep learning model M to obtain an model $\widetilde{M}$ that has exact same classification accuracy in software. However, the data layout organization modifies the assignment of the weight matrices to the crossbars within the platform. This can for example be performed by swapping the location of two neurons in the same layer of a neural network. The details of the data layout organization are provided in Section 4.

The third step is to convert each matrix into memristor conductance values while accounting for non-ideal crossbar effects [4, 11, 15, 16]. This mainly involves specifying the conductance of the memristors in the top-right corner of each crossbar to be more conductive, which compensates for the voltage IR-drop over the array parasitics. As conductance values cannot be negative, we use a differential pair configuration to represent each matrix. Consequently, each matrix is split into a positive and negative component.

The fourth step is to program the memristor devices in hardware to the specified conductance values. Programming techniques based on accurate closed-loop tuning have demonstrated a write bit-accuracy of 5-6 bits within array structures [5].

The fifth step is to perform inference by streaming input data to the configured in-memory computing platform. The classification accuracy in hardware is evaluated as the percentage of inputs that are correctly classified. The classification accuracy of a model M compiled to the platform P is denoted $P(M)$. Similarly, the classification accuracy of the model $\widetilde{M}$ is denoted $P(\widetilde{M})$. The difference between the classification in software and hardware stem from errors introduced by the analog computation within each crossbar and the peripheral circuitry used to convert signals between the analog and digital domain, i.e., digital-to-analog converters (DACs) and analog-to-digital converters (ADCs).

2.2 Motivation and problem definition

In this section, we provide the high-level motivation for studying data layout organization for DNN models deployed on in-memory platforms. Given a neural network model M, we apply data layout organization to obtain two models M_1 and M_2. The classification accuracy in software is 92.5% for both the models M_1 and M_2. We show the classification accuracy in hardware of the two models $P(M_1)$ and $P(M_2)$ in Figure 2. The figure shows that model M_1 achieves an accuracy close to the software level while the model M_2 achieves a classification accuracy of only 60%. This highlights the importance of data layout organization. The difference between M_1 and M_2 is only how the data is organized within the models, which results in that different matrices are mapped to the crossbars within P. Therefore, it can be concluded that *precision* of the analog computation greatly depends on how the *matrix data* is assigned to the crossbars.

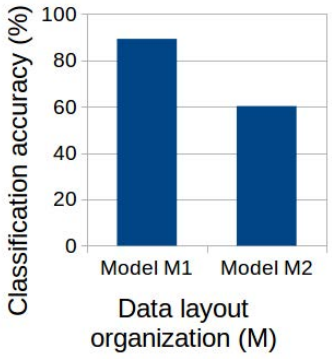

Figure 2: Hardware classification accuracy of a VGG style network [12] deployed on a platform P with data layout organizations M_1 and M_2. The platform is based on 256x256 crossbars and 5-bit memristors.

Problem statement: The objective of this paper is to determine the data layout transformation L that converts a DNN model M into a DNN model $\widetilde{M} = L(M)$ with the maximum classification accuracy in hardware $P(\widetilde{M})$.

We approach this problem by first performing a case study in Section 3 to understand what matrix properties that result in computation with low and high precision, respectively. Next, we propose a principled framework called XORG, which optimizes the matrix data to crossbar assignments using data layout organization in Section 4.

3 CASE STUDY: ANALOG ERRORS

In this section, we perform a case study to analyze the dependency between the matrix data stored in a crossbar and the precision of the analog computation. The goal is to understand the underlying matrix properties that result in computation with low and high precision, respectively.

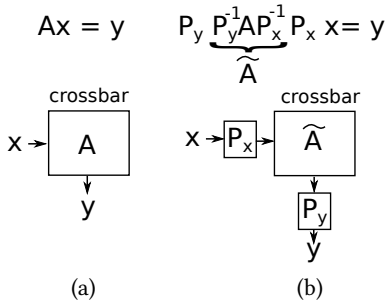

Figure 3: (a) Matrix-vector multiplication $Ax = y$. (b) Matrix-vector multiplication with rows and columns permuted using P_x and P_y, respectively.

The case study is performed by constructing a matrix A by sampling elements from a Gaussian distribution. Next, we observe that performing the matrix-vector multiplication $Ax = y$ in software is equivalent to $P_y\widetilde{A}P_x x = y$, where $\widetilde{A} = P_y^{-1}AP_x^{-1}$ and P_x and P_y are permutation matrices. However, the reformulation results in different computational accuracy in hardware because different matrices A and $\widetilde{A}$ are mapped to the crossbars, which is shown in Figure 3. To analyze the impact of the data to hardware assignment, we generate 100 different permutation matrices P_x and P_y and map the 100 different matrices $\widetilde{A}$ to a crossbar using the algorithms in [4, 11, 15, 16]. This involves converting the matrices into a set of memristor conductance values g within the programmable conductance range $[g_{min}, g_{max}]$. The conversion is performed by linearly mapping A (or $\widetilde{A}$) into the conductance range and tuning the memristors more conductive to compensate for the voltage IR-drop over the resistive parasitics. We show the normalized output errors for the different matrices with respect to the utilization of the programmable memristor conductance range in Figure 4(a). A two-dimensional histogram of the locations where memristors have been programmed to g_{max} is illustrated in Figure 4(b).

Source of Analog Errors: We observe that normalized maximum output error varies from 1X to 1.5X when only a portion of the programmable conductance range $[g_{min}, g_{max}]$ is used. In contrast, the normalized maximum output errors vary from 1.0X to 5.9X when the full conductance range is used. This is a result of that large errors are introduced when the conductance of a memristor is attempted to be tuned above g_{max}. As expected, the memristors that are attempted to be tuned larger than g_{max} are located in the top-right crossbar corner, where the voltage IR-drop is more severe.

Proposed Guideline for Data Layout Organization: Based on these observations, we propose the following guide line for data layout organization.

Matrix elements in A of large magnitude should be placed close to the bottom-left corner within a crossbar. Similarly, matrix elements of small magnitude should be placed close to the top-right corner.

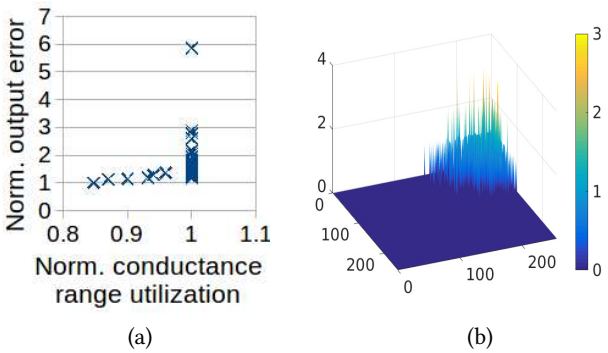

Figure 4: (a) Normalized maximum output error and norm. conductance range utilization for 100 different permutation matrices P_x and P_y. (b) A two-dimensional histogram of locations where the memristor conductance is set to g_{max} for the same 100 permutation matrices.

The placement of large elements in the bottom-left corner serves two purposes. First, it is less likely that the corresponding memristor will be attempted to be tuned above g_{max}, which we observed was the main source of errors in Figure 4(a). Second, the overall voltage IR-drop in the crossbar will be reduced because smaller currents are flowing long distances along the wordlines and bitlines. Consequently, other memristors are likely to require less tuning. The purpose of placing matrix elements of small magnitude in the top-right crossbar corner is that those elements are linearly mapped into small conductance values. Consequently, there will be a significant amount of conductance margin available for voltage IR-drop compensation.

It is important to note that matrix to conductance conversion algorithms already attempt to counter the issue of tuning memristors above g_{max} by mapping A in to a less conductive conductance matrix $G = \alpha A$ [4, 11, 15, 16], by specifying a smaller scaling factor α. The scaling factor α is realized by the peripheral circuitry. G is the conductance matrix that relates the input voltages (v_{in}) to the output currents $i_{out} = Gv_{in}$. However, using a too small α also introduces errors because fewer conductance states are utilized in the bottom-left crossbar corner. The recent mapping algorithms do already optimize α to balance these two types of errors [4, 11, 15, 16]. We view the proposed data layout organization to be orthogonal to the mapping techniques, which can be used in synergy. In the next sections, we present the XORG framework that provides a principled approach to data layout organization for DNN models deployed to in-memory computing platforms.

4 THE XORG FRAMEWORK

A cost metric that measures the quality of a data layout assignment is presented in Section 4.1. A family of data layout transformations $\mathcal{L}$ are presented in Section 4.2. The flow of the XORG framework is explained in Section 4.3.

4.1 Quality metric for data layout organization

In this section, we present the quality metric that measures how well the data layout organization of a DNN model follows the proposed guideline. The metric is based on multiplying the absolute value

of every weight in a neural network with a crossbar dependent location cost C, as follows:

$$cost(M) = \sum_{l\,=\,1\text{ to }(L-1)} \sum_{(i,j)\in W_l} C(i,j) \cdot |W(i,j)_l|, \qquad (1)$$

where W_l is the weight matrix of layer l. C is a cost matrix of the same dimensions. $C(i,j)$ and $W(i,j)_l$ are the elements on row i and column j of C and W_l, respectively. $|.|$ is the absolute value operator.

The absolute value operator is used because each weight matrix W_l is decomposed into a positive and negative component W_l^+ and W_l^-, which are mapped to two separate crossbars arranged in a differential pair configuration. Small scores correspond to high quality data layout organizations and high scores correspond to low quality data layout organizations.

4	8	12	16
3	6	9	12
2	4	6	8
1	2	3	4

Figure 5: Location based cost for a 4x4 matrix.

The costs in C are set to be high in the top-right corner of the crossbar and small in the bottom-left corner. The cost matrix C for a 4x4 crossbar is shown in Figure 5. We note that many weight matrices within a neural network are larger than the crossbar size. Therefore, we define the cost for an element C_{ij}, as follows:

$$C_{ij} = (mod(i,M)+1) \cdot (mod(j,N)+1), \qquad (2)$$

where i and j refer to the row and columns, respectively. M and N refer to the crossbar dimensions, respectively.

4.2 Data layout transformations $\mathcal{L}$

The concept of modifying the data to hardware assignment within an in-memory computing platform has been explored to handle defective devices [18, 19]. In this paper, we instead optimize the data to hardware assignment with the objective of minimizing the impact of the resistive parasitics.

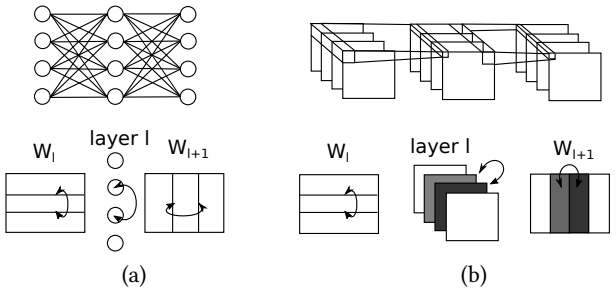

Figure 6: Data layout transformations in the form of (a) neuron and (b) channel transformation.

We define a family of transformations $\mathcal{L}$ consisting of a *neuron transformation* and a *channel transformation*. The two transformations are illustrated with an example in Figure 6. A CNN consists of L layers of neurons. The layers are in the form of convolutional layers or fully-connected layers. The neuron transformation is applied to neurons between fully-connected layers. The channel transformation is applied to feature maps between convolutional layers. The neuron transformation involves reordering the neurons within a layer l. The reordering of the neurons naturally modifies the ordering of the rows in the weight matrix W_l connecting layer $(l-1)$ to l and the ordering of the columns in the weight matrix W_{l+1} connecting layer l to layer $(l+1)$, which is shown in Figure 6(a). The channel transformation involves reordering the channels within the feature map of layer l. The reordering modifies the groups of rows in the weight matrix W_l connecting layer $(l-1)$ to layer l and the ordering of columns in the weight matrix W_{l+1} that connects l to $l+1$, which is shown in Figure 6(b). The group size is $9 \cdot C_l$, where C_l is the number of channels in layer l.

4.3 Flow of XORG framework

In this section, we describe the flow of the XORG framework. The framework is applied to perform data layout organization after the decomposition and resource allocation step and before the platform reconfiguration step in Figure 1. The input to the XORG framework is a DNN model M and the in-memory computing platform P. The output is a more resilient deep learning model $\widetilde{M}$ that has a smaller cost defined by the metric in Eq (1). The model M is compiled into the resilient model $\widetilde{M}$ by applying data layout transformations $L \in \mathcal{L}$ using Algorithm 1. Consequently, the classification accuracy in hardware $P(\widetilde{M})$ is expected to be higher than $P(M)$. Note that both M and $\widetilde{M}$ have the exact same accuracy when evaluated in software.

Algorithm 1: XORG: Data Layout Organization.

Input: DNN model M with L layers.
Output: Resilient DNN model $\widetilde{M}$.
for *l = 2 to (L-1)* **do**
  // Apply transformation to layer l
  Compute cost matrix
  Solve assignment problem
  Reorganize data within W_l and W_{l+1}
end
return $\widetilde{M}$;

The XORG framework casts the data layout organization problem as an optimization problem focused on minimizing the metric in Eq (1). The metric is minimized by iterating over the internal layers of a the deep learning model M and applying the neuron transformation or the channel transformation to each layer. The feature maps in the first layer and the neurons in the last layer are not transformed to ensure that the optimization is seamless within the flow in Figure 1. The neuron or channel transformation for a layer can be viewed as assigning a neuron/channel to a location, which can be cast as the well known assignment problem. The assignment problem can be solved both optimally using the Hungarian algorithm [2, 19]. Next, the model M is updated with the new neuron order in layer l based on the assignment solution.

5 EXPERIMENTAL EVALUATION

5.1 Simulation setup

The experimental results are obtained using a quad core 3.4 GHz Linux machine with 32GB of memory. The XORG framework is

implemented using a cross-layer framework involving, C++, Tensorflow, MATLAB, and HSPICE. The crossbar parameters used in the evaluation are provided in Table 1. The default crossbar size is set to 256x256. The sensitivity to the crossbar dimensions and noise is evaluated at the end of the experimental results section. We first evaluate XORG on the crossbar level in Section 5.2. Next, we evaluate XORG using neural network applications in Section 5.3.

Table 1: Crossbar parameters in evaluation.

Property	Value
Array block resistance	1 Ω
Input resistance	100 Ω
Output resistance	100 Ω
Programmable resistance range	$[2k, 300k]$ Ω
Max input voltage	$0.25V$
Memristor bit-accuracy	6 bits
DAC/ADC bit-accuracy	8 bits
Transistor model	JETMOS v1

5.2 Crossbar level evaluation

To demonstrate that the XORG framework is capable of improving the precision of analog matrix-vector multiplication, we repeat the experiment in the case study and compare the baseline (random organization) with XORG+ and XORG−. XORG+ is the XORG framework proposed in Section 4. XORG− is the XORG framework while maximizing the cost in Eq (1), which makes the data layout organization worse instead of better. We evaluate the three techniques in terms of cost in Eq (1) and the normalized output errors in Figure 7. The analog computation is performed using circuit simulation with SPICE level accuracy.

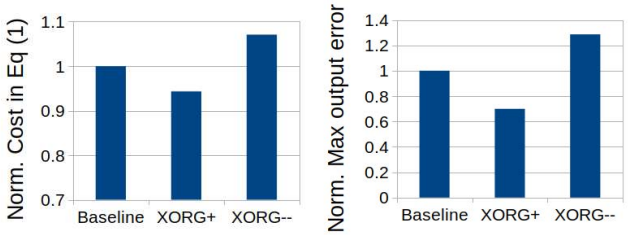

Figure 7: (a) Norm. cost in Eq (1). (b) Norm. max output error for 100 matrices with different data layout organizations.

The cost in Eq (1) is evaluated in Figure 7(a). The figure shows that the XORG framework is capable of reducing and increasing the cost in Eq (1) with 6% and 7%, respectively. This translates into that XORG+ reduces the maximum output errors with 31% and XORG− increases the normalized output errors with 28%, which is shown in Figure 7(b). The strong correlation between the cost in Eq (1) and the normalized output errors confirms our conclusions drawn in the case study. The 1.8X difference in output errors highlights the need for resilient data layout organization and the effectiveness of the XORG framework.

5.3 DNN level evaluation

In this section, we evaluate the effectiveness of XORG on the application level using the flow in Figure 1. The input to the flow is a set

Table 2: Evaluated neural networks.

Name	Dataset	Software accuracy (%)	#Conv. layers	#FC layers	#Max pooling layers	#Norm layers	#Train. params (M)
VGG-7	CIFAR-10	82.8	4	2	2	6	2.2
VGG-10	CIFAR-10	88.8	7	2	3	9	1.5
VGG-13	CIFAR-10	92.5	10	2	5	12	9.7
VGG-16	CIFAR-10	94.0	13	2	5	14	14.9

of DNN models M and an in-memory computing platform P. For the DNN models, four convolutional neural networks (CNNs) are trained on the CIFAR-10 dataset [9] using TensorFlow on a NVIDIA Tesla K80 GPU. The details of the neural networks are provided in Table 2. We use a platform P with crossbars of dimension up to 256x256. Consequently, weight matrices that are larger than 256x256 will be partitioned to multiple crossbars. Weight matrices smaller than 256x256 are mapped to smaller crossbars if power can be saved. Within the flow in Figure 1, we use the XORG framework described in Section 4 to perform the data layout organization step. Again, we let XORG+ and XORG− denote the case when XORG is used to improve or degrade precision and classification accuracy, respectively. Here, baseline is the DNN model obtained from Tensor Flow.

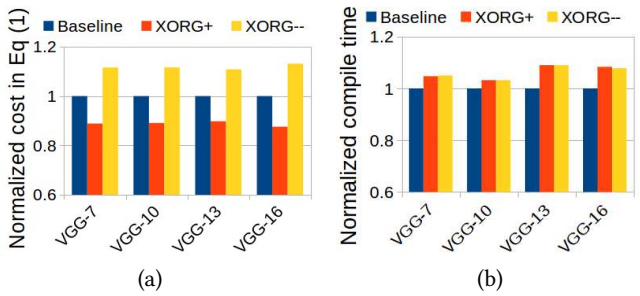

Figure 8: (a) Cost in Eq (1). (b) Compile time overhead.

The cost in Eq (1) and the overhead in compile time is evaluated in Figure 8. Compared with the baseline, XORG+ reduces the cost with 11% and XORG− increases the cost with 11%, respectively. The improvements come at the expense of a modest 6% overhead in terms of compile time. The data layout organization takes between 2 and 26 min depending on the network size, which is significantly smaller than the run-time of 0.8 to 5.4 hours for the voltage IR-drop compensation in [4, 11, 15, 16].

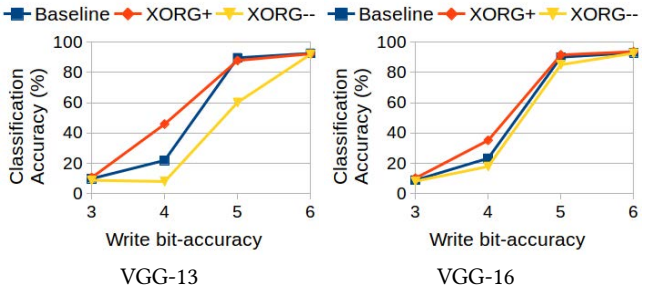

Figure 9: Evaluation of mapping cost and classification accuracy with respect to memristor bit-accuracy.

Now we turn our attention to evaluating the capability of the XORG framework at reducing the memristor write bit-accuracy requirements in Figure 9. We show the classification accuracy for VGG-13 and VGG-16 in (a) and (b) of Figure 9. While only using a write bit-accuracy of 5 bits, it can be observed that the Baseline and XORG+ is capable of achieving an classification accuracy close to the software level. In contrast, XORG– requires a write bit-accuracy of 6-bits to achieve similar classification accuracy. While the initial data layout organization happens to result in high accuracy, XORG+ ensures that a poor data layout organization is not selected.

Next, we evaluate the sensitivity of the classification accuracy to the crossbar dimensions in Figure 10. The classification accuracy of VGG-7 and VGG-10 with respect to the crossbar dimensions is shown in (a) and (b) of Figure 10. The memristor bit-accuracy is set to 6-bits. While all methods have similar classification accuracy for smaller crossbars, it is clear that XORG+ achieves the highest classification accuracy when the crossbar dimensions are scaled to 512x512. The explanation is that it is naturally more important to mitigate the impact of parasitics when larger crossbars are used. The figure shows that Baseline is significantly worse than XORG– gor VGG-7. Given the results on the crossbar level, we speculate that this stems from application level properties.

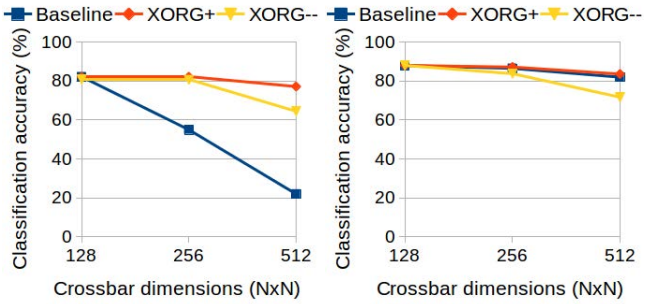

Figure 10: Classification accuracy w.r.t. crossbar dimensions.

Now we focus on evaluating the effectiveness of XORG at improving the robustness to RTN using 256x256 crossbars. We introduced RTN, which we model using a uniform distribution with $[-x\%, +x\%]$. It can be observed that XORG+ produces data layout organizations that are more resilient to the RTN than both the Baseline and XORG–. The higher resilience to errors stems from that XORG+ results in mapping solutions with larger α and the errors in the analog domain are scaled with $1/\alpha$. To the best of the authors knowledge, this observation provides a new pathway to improving the resilience to RTN.

6 SUMMARY AND FUTURE WORK

In this paper, we proposed the XORG framework that performs resilient data layout organization for DNN models deployed on in-memory computing platforms. The framework is based on the observation that the resistive parasitics within a crossbar introduces a dependency between the matrix data and the precision of the analog computation. This opens the door to reorganizing the data within a DNN model to improve classification accuracy by optimizing the matrix to crossbar assignments. The experimental results show that the XORG framework relaxes the hardware requirements

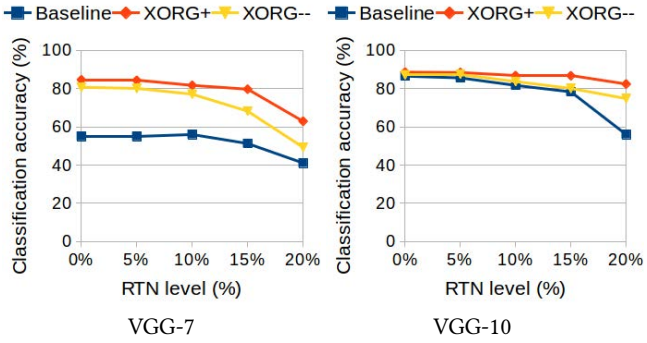

Figure 11: Evaluation of resilience to RTN.

and improves the robustness to noise. We believe that this paper will establish data layout organization as a standard design step within compilers for in-memory computing platforms.

ACKNOWLEDGMENTS

This work was in part supported by NSF awards #1755825, #1908471, #2008339, #2113307, DARPA cooperative agreement #HR00112020002, ONR grant #N000142112332, and DOE/NNSA.

REFERENCES

[1] S. Choi, Y. Yang, and W. Lu. Random telegraph noise and resistance switching analysis of oxide based resistive memory. *Nanoscale*, 6, 11 2013.
[2] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
[3] Z. He et al. Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping. DAC'19, pages 57:1–57:6, 2019.
[4] M. Hu et al. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. DAC'16, pages 1–6, 2016.
[5] M. Hu et al. Memristor-based analog computation and neural network classification with a DPE. *Adv. Materials*, 30, 2018.
[6] A. James, O. Krestinskaya, and L. Chua. Neuromemristive circuits for edge computing: A review. *IEEE Transactions on Neural Networks and Learning Systems*, PP, 02 2019.
[7] Y. Ji et al. Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler. ASPLOS '18, page 448–460, 2018.
[8] K. Kourtis et al. Compiling Neural Networks for a Computational Memory Accelerator. *arXiv e-prints*, page arXiv:2003.04293, 2020.
[9] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
[10] Y. LeCun et al. Deep learning. In *Nature*, pages 436–444, 2015.
[11] B. Liu et al. Reduction and IR-drop compensations techniques for reliable neuromorphic computing. ICCAD'14, pages 63–70, 2014.
[12] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, Nov 2015.
[13] C. Münch, R. Bishnoi, and M. B. Tahoori. Reliable in-memory neuromorphic computing using spintronics. ASPDAC '19, page 230–236, 2019.
[14] L. Song et al. Pipelayer: A pipelined reram-based accelerator for deep learning. HPCA'17, pages 541–552, 2017.
[15] N. Uysal et al. Dp-map: Towards resistive dot-product engines with improved precision. In *ICCAD'20*, pages 1–9. IEEE, 2020.
[16] N. Uysal et al. Xmap: Programming memristor crossbars for analog matrix-vector multiplication: Towards high precision using representable matrices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.
[17] W. A. Wulf and S. A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, 1995.
[18] L. Xia et al. Fault-tolerant training with on-line fault detection for rram-based neural computing systems. DAC'17, pages 1–6, 2017.
[19] B. Zhang et al. Handling stuck-at-fault defects using matrix transformation for robust inference of dnns. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1(56):1–14, 2019.
[20] Z. Zhu et al. Mixed size crossbar based RRAM CNN accelerator with overlapped mapping. ICCAD'18, pages 69:1–69:8, 2018.