ReMeCo: Reliable memristor-based in-memory neuromorphic computation

Ali BanaGozar*
Sander Stuijk, Henk Corporaal
Eindhoven University of Technology, The Netherlands
{a.banagozar,s.stuijk,h.corporaal}@tue.nl

Seyed Hossein Hashemi Shadmehri*
Ali Afzali-Kusha
University of Tehran, Iran
{seyed.hashemi.ho,afzali}@ut.ac.ir

Mehdi Kamal
University of Southern California, CA, USA
mehdi.kamal@usc.edu

ABSTRACT

Memristor-based in-memory neuromorphic computing systems promise a highly efficient implementation of vector-matrix multiplications, commonly used in artificial neural networks (ANNs). However, the immature fabrication process of memristors and circuit level limitations, i.e., stuck-at-fault (SAF), IR-drop, and device-to-device (D2D) variation, degrade the reliability of these platforms and thus impede their wide deployment. In this paper, we present ReMeCo, a redundancy-based reliability improvement framework. It addresses the non-idealities while constraining the induced overhead. It achieves this by performing a sensitivity analysis on ANN. With the acquired insight, ReMeCo avoids the redundant calculation of least *sensitive* neurons and layers. ReMeCo uses a heuristic approach to find the balance between recovered accuracy and imposed overhead. ReMeCo further decreases hardware redundancy by exploiting the bit-slicing technique. In addition, the framework employs the ensemble averaging method at the output of every ANN layer to incorporate the redundant neurons. The efficacy of the ReMeCo is assessed using two well-known ANN models, i.e., LeNet, and AlexNet, running the MNIST and CIFAR10 datasets. Our results show 98.5% accuracy recovery with roughly 4% redundancy which is more than 20× lower than the state-of-the-art.

KEYWORDS

Reliability, Neural Networks, Computation In-Memory, Neuromorphic, Memristor, Redundancy, Stuck-at-fault, Process Variation

1 INTRODUCTION

Artificial Neural Networks (ANNs) are becoming more prevalent and more complex. They have been successfully adopted in various fields, e.g., computer vision, natural language processing, etc. [6]. However, conventional processing elements (PE)s execute such data-intensive workloads inefficiently due to the memory-wall [1]. Computation in-memory (CIM) alleviates this problem by reducing the movement of data required for vector-matrix multiplication (VMM), a core operation of ANNs [15]. A promising approach to realizing CIM is deploying emerging non-volatile memories, e.g., memristors, and phase-change memories (PCM), in a crossbar structure (Figure 1.(a)). Memristor-based CIM significantly reduces the energy consumption and executes VMM with $O(1)$ complexity.

Despite all the benefits offered by memristor-based ANN PEs, their output accuracy is unstable. Therefore, their reliability has been a critical point of concern [7]. Several different factors contribute to this. Stuck-at-fault (SAF) is a significant reliability issue in which a defective device is frozen in a state [11, 14]. In addition, device-to-device (D2D) variation at the device-level and IR-drop at the system-level further degrade the accuracy [11]. All these non-idealities diminish the promise of widely adopting memristor-based platforms for implementing ANNs.

To counter these non-idealities, many prior researches have proposed to re-train the neural network model taking some non-idealities into account, remap the weights to the memristor crossbar, pre-test the circuit, or perform a combination of these techniques [2, 10, 13, 19, 21]. Some works re-train the network in the post-fabrication stage by back-annotating the physical characteristics of each chip and including them in the training algorithm [3, 10]. Joksas et al. propose a redundancy-based approach inspired by static committee machine. They implement an ANN model multiple times on different crossbars to form a committee machine (CM). The CM yields better accuracy than individual implementations by leveraging ensemble averaging, commonly used in conventional ANNs [11].

In this paper, we first do a thorough design space exploration (DSE) to understand the effects of various non-idealities on the final accuracy of neuromorphic PEs. Furthermore, we introduce ReMeCo, a redundancy-based reliability improvement framework where redundancy is applied in a structured manner. In ReMeCo, we first analyze the ANN –to be implemented– to recognize its *sensitive* parts. Using this insight, we improve the CM by applying redundancy to where it recovers accuracy most. Hereby we trade off the overhead of redundancy and accuracy. Our contributions are:

- Design space exploration of the effects and importance of various sources of non-idealities, e.g., SAF, D2D variation, and IR-drop. In addition, we study the impact of the write-and-verify algorithm in alleviating the detrimental effects of mentioned reliability issues. (Section 2.2)
- ReMeCo: a novel redundancy-based framework to improve the reliability of memristor-based ANN PEs by addressing the non-idealities. In doing so, we do not need to change the training algorithm or do retraining/remapping for every fabricated PE. By identifying the sensitive neurons (S-neuron)

*Both authors contributed equally to this research.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01.
https://doi.org/10.1145/3566097.3567889

during the training process of the ANN –without any overhead, we apply redundancy in a structured manner to where it can contribute the most to recovering accuracy. (Section 3)
- Assessing our approach using two widely accepted ANN models, namely LeNet and AlexNet that are trained/tested with the MNIST and CIFAR-10 data-sets. Our evaluation results show that ReMeCo recovers accuracy by up to 98.5% while limiting the energy and area overhead. (Section 4)

ReMeCo solves the reliability problem of memristor-based ANN PEs without any software overhead and with limited hardware redundancy.

2 PRELIMINARIES AND RELATED WORK

In this section, we briefly describe memristor crossbars and how they operate. Then, we detail the non-idealities that affect their reliability the most. We finish by explaining the solutions proposed to resolve the reliability issue of memristor-based CIM PEs.

2.1 Memristor crossbars

Memristor crossbars are considered one of the promising candidates for implementing ultra low-power in-memory neuromorphic computation systems. Memristor-based CIM exploits two fundamental circuit laws, namely Kirchhoff's current summation law and Ohm's law, to execute the VMM operation in the analog domain.

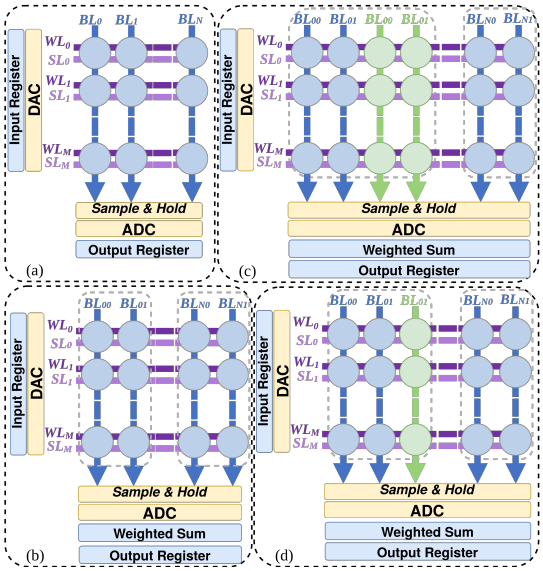


Figure 1: We consider 1T1R memristor crossbars [4]. Thus, two horizontal wires are needed. WL enables/disables the access transistor while SL is for feeding input. (a) A crossbar where the precision of ANN weights and memristors match. (b) A crossbar where the precision of ANN weights is twice that of a memristor. Hence, two columns per neuron are employed (*bit-slicing*). (c) A crossbar with sensitivity-based redundancy (green columns). The first neuron (left two columns) is duplicated whereas the last one is not. (d) A crossbar exploiting Most Significant Bits (MSB) redundancy.

This leads to orders of magnitude improvement in energy consumption and highly parallel execution of VMM, which is critical for the execution of an ANN (Figure 1.(a)) [1, 5, 16]. To perform VMM, a matrix, e.g., the weights of an ANN layer, is first mapped to a set of conductances that are implemented on the memristor devices in the crossbar. The input vector, e.g., the inputs of an ANN layer, is mapped to a vector of voltages. Then, this vector is applied to the rows of the crossbar (SL in Figure 1). The applied voltage vector induces currents in the columns of the crossbar (BL in Figure 1), which represent *output neurons* of an ANN layer. The analog currents flowing in the columns correspond to the dot-product of the input vector and the weight matrix. In the end, using an analog-digital converter (ADC), the analog currents are converted back to digital values.

Memristors, however, are capable of storing a limited number of conductance levels. Thus, they cannot store high-precision weights. A common way to deal with this is to split the ANN weights over multiple memristors residing in different (usually adjacent) columns. In this approach, called *bit-slicing* [1, 16], a weighted sum on the bit-lines yields the final result (figure 1.(b)).

2.2 Non-idealities (DSE)

Memristive devices are subject to various non-idealities. Therefore, reliability is one of the major aspects that need to be studied for a wider deployment of memristor-based platforms. Examples of such non-idealities are stuck-at-fault (SAF), and device-to-device (D2D) variation [11]. Stuck-at-fault (SAF) is a significant reliability issue in which a defective device is frozen in a state [11, 14]. This

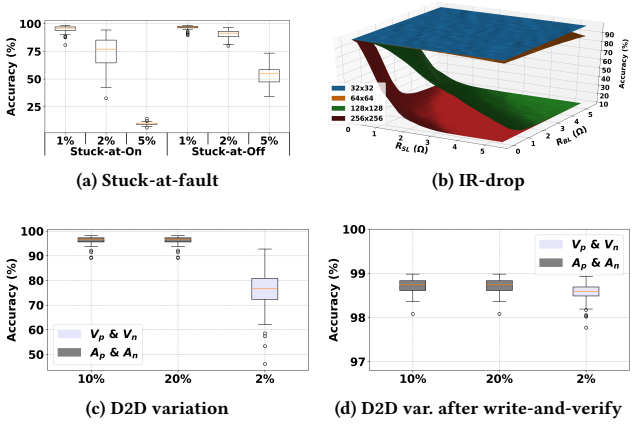


Figure 2: The effect of various non-idealities on classification accuracy of LeNet on MNIST dataset. (a) Our results show that the stuck-at-on devices affect the accuracy more significantly than stuck-at-off. (b) As the crossbar size becomes larger, the effect of line resistance (R_{BL}, R_{SL}) on the accuracy grows significantly. (c-d) Ap(An), and Vp(Vn) are the modulation factor and threshold voltage for setting(resetting) memristors, respectively. As (d) shows, the write-and-verify technique can address the D2D variation effect to a significant extent.

Table 1: Summary of the works done to address reliability issues. Except for ReMeCo and [11] all the other works propose approaches that require to be applied to each fabricated chip individually which elongates the time to market. Furthermore, all works address SAF, and some of them propose solutions for D2D process variation (PV) as well. However, only ReMeCo and [11] address the above issues and system-level IR-drop as well. ReMeCo excels CM by performing sensitivity analysis.

First author (Year)	Non-idealities	Model-aware training	Post-fabrication training	Remapping	Hardware redundancy	Sensitivity analysis	Time to market
Gaol (2021) [8]	SAF, PV*	X	X	X	-	X	High
Jin (2020) [10]	SAF	-	X	X	-	X	High
Xu (2021) [19]	SAF, PV	X	X	X	-	X	High
Liu (2017) [13]	SAF	X	X	X	X	X	High
Joksas (2020) [11]	SAF, PV, IR-drop	-	-	-	X	-	*Low*
ReMeCo	SAF, PV, IR-drop	-	-	-	X	X	*Low*

phenomenon causes the weights of an ANN to be mapped incorrectly, resulting in an inaccurate computation. D2D variation also can cause the memristors to be miss-programmed. This is due to the fact that different devices behave differently when programming pulses are applied to them. Therefore, they may be programmed to the wrong conductance level. To alleviate these problems, various techniques such as the write-and-verify programming scheme [17], or using high-precision processing units in conjunction with memristive elements [12, 18] are proposed. The limited write-endurance of memristors is another issue that becomes significant when in-situ training is used since it requires memristors to be programmed many times [11].

In addition to the device-level non-idealities, IR-drop, at the system-level, affects the final accuracy [9]. Since memristor crossbars perform multiplication with Ohm's law, wire resistance can cause the induced current and subsequently the VMM output to deviate. The problem exacerbates as the crossbar size (and consequently wire resistance) grows.

Using the simulator proposed in [15], we perform DSE to study the effect of these non-idealities on the reliability of a memristive CIM platform. Figure 1 depicts the simulation result for the execution of LeNet-MNIST on 1000 different memristor crossbar models with 10K test instances (implementation details in Section 4.1). As the figures 2 (a-c) suggest, SAF, IR-drop, and D2D variation have a detrimental effect on the accuracy of these platforms. Our simulations further demonstrate that different types of SAF impact accuracy differently. Figure 2.(a) shows that stuck-at-on, which is a more prevalent type of SAF [19], degrades accuracy more significantly than stuck-at-off does. Figure 2.(b) confirms the aforementioned claim that the bigger the crossbar is, the more impactful the IR-drop becomes. Figure 2.(d) shows that the D2D variation effect depicted in figure 2.(c) (Vp$\in N(\mu = 4, \sigma = 0.08)V, Ap\in N(\mu = 8.16e5, \sigma = 0.1, 0.2\mu)$) can be mitigated by employing the *write-and-verify* programming method. For the rest of this article, we assume that the write-and-verify method is employed. Hence, D2D variation has a negligible effect on accuracy.

2.3 Addressing non-idealities

Different approaches have been proposed to deal with the reliability issues of memristive ANN PEs. Plenty of research has been directed toward software-based approaches. Some of them enhance the initial training scheme by including physical characteristics extracted from device models and circuit simulations [2, 8, 19]. The problem with this approach is that they are still susceptible to variation and SAF since the developed model cannot predict the exact distribution of non-idealities and thus cannot mitigate their effect.

Other works re-train their ANN model in the post-fabrication test stage and include the actual physical characteristics of each chip in the re-training process [13, 19]. Another post-fabrication approach that is widely studied is remapping. The main idea behind remapping is to avoid mapping sensitive weights to faulty devices [8, 10, 13, 19]. The sensitivity of an ANN model weight is determined on the basis of the impact its variation has on the final output. Post-fabrication approaches are non-trivial, and costly since the process of extraction of physical characteristics and re-training or remapping should be applied for each fabricated chip [8].

Besides software approaches, some hardware approaches have been proposed [11]. Joksas et. al. propose a static CM where an ANN model is implemented on several memristive crossbars with different non-idealities. The ensemble averaging-based CM reports the averaged output of all implementations as the final result. Although compared to software-based ideas, the CM is easier to implement and avoids costly re-training and remapping processes, it is expensive in terms of energy consumption and area overhead. For instance, the authors of [11] have implemented a simple ANN model on *five* different crossbars to form a CM that recovers the accuracy to its original state.

Software approaches are orthogonal to hardware methods. Thus, they can be applied jointly. Lastly, note that the main non-ideality that most of the approaches above try to address is SAF; and other significant non-idealities are rarely studied and resolved. Table 1 summarizes these recent memristor-crossbar reliability studies.

3 REMECO: APPROACH AND IMPLEMENTATION

We perform DSE on non-idealities that affect the memristor crossbar to realize the significance of the impact that they have on the accuracy of such platforms. The corresponding results are depicted in Figure 2 and briefly discussed in Section 2. Having understood the problem, we propose ReMeCo, which is a framework to improve the reliability of memristor-based ANN PEs in the presence of various non-idealities by adding hardware redundancy. ReMeCo is independent of device properties, and it does not have to be adjusted for each fabricated chip. Therefore, it does not elongate the *time*

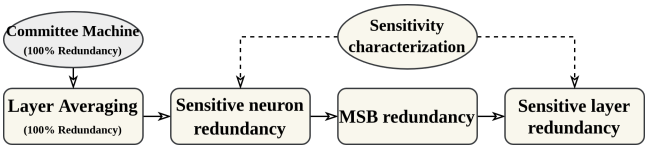


Figure 3: The flow-chart of the ReMeCo framework. The data from sensitivity characterization is used in later steps.

to market. The key idea behind ReMeCo is to *apply redundancy to where it contributes most* and hereby limit the overhead. To do so, ReMeCo, first, identifies the *sensitive* neurons and layers. ReMeCo, then, starts with the CM and uses the acquired insight to reduce CM's hardware overhead as much as possible while improving the accuracy recovery (Figure 3). We discuss these in detail in the rest of this section.

3.1 Sensitivity Characterization

It is crucial to determine the *sensitivity* of an ANN output with respect to its individual neurons and layers. This enables us to apply our accuracy recovery method in a structured and more efficient fashion, which in return reduces the incurred overhead. To quantify the sensitivity, we use the *back-propagation* algorithm. Back-propagation is a widely used algorithm for training neural networks. Basically, it is the process of fine-tuning the weights of an ANN until it reaches the desired accuracy. Back-propagation follows a feed-forward path in which instance(s) from the training data-set are fed to the ANN and their corresponding output error with respect to the defined target is calculated. Back-propagation algorithm dispenses the output error back to the ANN parameters. Then, the ANN weights are adjusted in proportion to their contribution to the final error [13, 18]. Back-propagation for training ANNs is commonly formulated as a gradient descent optimization problem, as $\Delta\omega_{i,j} = -\eta \frac{\partial E}{\partial \omega_{i,j}}$, where η denotes the learning rate, E denotes the final error, and $\frac{\partial E}{\partial \omega_{i,j}}$ shows the *sensitivity* of E to $\omega_{i,j}$. Repeating the process for every instance in the training set, (1) we compute the feed-forward path, (2) obtain the output error, and (3) calculate the average error contribution of individual neurons over all the instances in the training data-set. Therefore, neuron sensitivity is formulated as, $Sens._{neuron} = AVR_{j=0}^{N}(\frac{\partial E}{\partial \omega_{i,j}})$. In the same manner, we calculate the *sensitivity* of an ANN *layers* by averaging over the *sensitivity* of all of its output neurons. This way, by exploiting the training process and without causing any software overhead, ReMeCo apportions the error at the output of an ANN to the individual neurons and layers [13, 18]. Figure 4 shows the histogram of neuron sensitivity of LeNet ANN model. For instance, it indicates that neurons of layer 3 and 4 are less *sensitive*; hence, they can be targeted to reduce neuron and layer hardware redundancy.

3.2 Implementation

We map the weights of an ANN to the memristor crossbar so that *columns* in the memristor crossbar corresponds to *output neurons* in an ANN layer. To perform high-precision computation on memristor crossbars, we employ the *bit-slicing* technique –explained in Section 2 [1]. Therefore, after analog to digital conversion (ADC), the partial results from several columns, which together store one

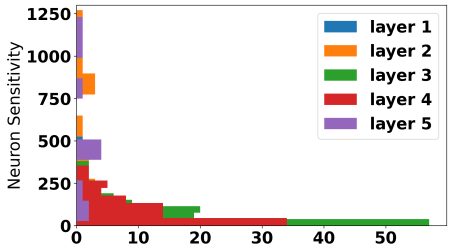


Figure 4: Absolute neuron sensitivity histogram for LeNet.

weight, are shifted and added (weighted sum) to produce the final result (Figure 1.(b)).

The ReMeCo framework follows the flow shown in Figure 3. As the figure indicates, after acquiring the sensitivity characterization, ReMeCo takes the CM [11] as its starting point. It applies the techniques shown in the figure one after another to increases the final accuracy and simultaneously decreases the area and energy overhead.

Layer averaging Joksas et al. in [11] propose to use CMs to address the non-idealities associated with memristive ANN PEs and improve their output accuracy. They follow the ensemble averaging (EA) method to combine the results of individual ANN implementations in the CM. As the first step in the ReMeCo framework, we propose to modify the CM and perform the *EA* not only for combining the final output but rather *at the end of every layer*. By combining the results after every layer, ReMeCo diminishes the error produced in them. Therefore, the error that is propagating through the network to the following layers is smaller. This improves the accuracy of every layer's output and subsequently the overall accuracy of the memristive ANN PE.

Note that, we use the CM with two ANN implementations (the minimum overhead) as the starting point. It is important since by constraining the number of copies of a neuron to a power of two, the averaging can be implemented by simple *shift-&-add* operations. Considering that the bit-slicing technique already requires the weighted-sum module, performing averaging operation after every layer imposes negligible overhead compared to the CM.

Sensitive neuron redundancy The CM requires 100% to 400% hardware redundancy to recover the accuracy to its original software level. Although this might be acceptable for small-sized ANNs, it gets unwieldy when implementing deeper and larger ANN models. Therefore, in the second step of the ReMeCo framework we propose a method to reduce required redundancy. Basically, in *S-neuron redundancy* step, we propose to apply redundancy only to where it contributes most, i.e., *sensitive* neurons. Employing a heuristic approach and using the insight from *sensitivity characterization* step, we can gradually remove the redundant calculation of the less *sensitive* neurons. In doing so, we trade-off the unmanageable hardware overhead and the recovered accuracy. Figure 1.(c) visualizes this approach. As the figure suggests the *sensitive* neurons, e.g., the 1^{st} column, are calculated redundantly, while the *less sensitive neurons*, e.g., the N^{th} column, are calculated once. By avoiding the calculations that hardly affect the accuracy, ReMeCo promises to consume less energy and area.

MSB redundancy The bit-slicing technique offers an opportunity to further reduce the redundancy overhead. After adding the

redundancy according to the description in the previous stage, we propose to remove the redundant calculation of the columns storing to the least significant bit(s) of the weight, and perform the redundant calculation solely for the columns storing the most significant bit(s). Figure 1.(d) depicts this methods. In the example shown in the figure, one ANN neuron is implemented using two columns in the memristor crossbar. We reduce the area overhead by removing the copy of the LSB column and only keeping the MSB part. Employing this method in the shown example, ReMeCo reduces the crossbar area overhead by half. In implementations where the precision of memristors are low and the precision of the weights are high this approach yields even more hardware efficiency. Obviously, this method also decreases the energy consumption by reducing the number of redundant calculations, A/D conversions, and required averaging operations.

Sensitive layer redundancy In the final step, we use the the information obtained about the sensitivity of the ANN layers to further enhance the redundancy-based reliability framework. As in *S-neuron redundancy* stage, we propose to avoid the calculation of the layers that are less *sensitive*. This heuristic approach yields a significant reduction in hardware overhead as it eliminates the computation of a whole layer. This become even more significant as our sensitivity characterization studies show that the fully connected layers, which usually are significantly larger than convolutional layers, are actually the least sensitive layers. This is not quite applicable to the *last* fully connected layer though.

4 EVALUATION

For the evaluation of ReMeCo, we use two well-known deep ANN models, namely LeNet and AlexNet. The software accuracy for quantized 8-bit LeNet and AlexNet models on MNIST and CIFAR-10 data-sets are 98.94% and 80.12%, respectively. To address the reliability issues we employ both ReMeCo and the state-of-the-art CM. Since the CM is evaluated using a small NN model that comprises two fully connected layers to make a comparison, in terms of recovered accuracy and induced overhead, we decided to apply the method proposed in CM to our test cases, i.e., LeNet-MNIST and AlexNet-CIFAR10. To make the comparison fair (in terms of the caused overhead), we assume the CM only consists of two ANN implementations and therefore, has a redundancy factor of 100%. In the rest of this section, first, we present the characteristics of the memristor and of the required peripheral circuitry. We also describe the mentioned neural networks and the benchmarks. Then, we present the accuracy results achieved by employing ReMeCo. In the end, we present an estimation of the energy and area overhead that ReMeCo imposes on the system and compare them with those of the CM approach.

Table 2: Memristor crossbar physical paameters

Crossbar Parameters	**Value**	
Memristor Technology	Device in [20]	
Cell precision	8-bit (2×(4-bit) devices)	
Compute Energy/8-bit	160 fJ (2×80 fJ/4-bit device)	
Write Energy/8-bit	160 pJ (2×80 pJ/4-bit device)	
Crossbar Area	$10^{10} memristors/cm^2$	
SL/BL Resistance	$N(\mu = 2,\ \sigma = 0.5)\Omega$	
Peripheral Circuitry	**Energy**	**Area**
Analog Components	2.1 pJ/cycle (@1.2GHz)	1252 μm^2
Digital Components	1.1 pJ/byte	865 μm^2

Table 3: ANN parameters and data-sets

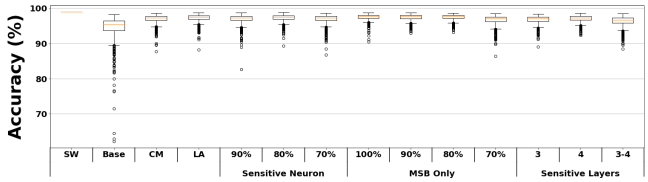
ANN	**Data-set**	**Layers**		**Parameters**	**FLOPs**
		Conv.	**Dense**		
LeNet	MNIST	2	3	866 K	28 M
AlexNet	CIFAR-10	5	3	62 M	1.5 B

4.1 Experimental Setup

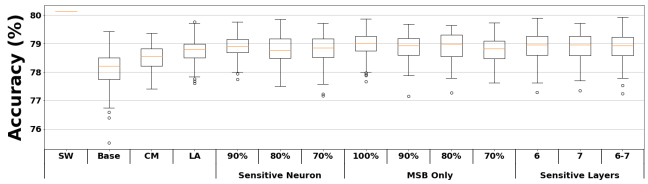
We use the simulator presented in [15] to evaluate our proposed approach. We extend the simulator to support our averaging scheme. Table 2 shows the default values for key crossbar parameters in the following experiments. To assess the effectiveness of our approach in recovering the accuracy of ANN, we employ LeNet and AlexNet. The details of the ANN models and their corresponding benchmarks are presented in Table 3. Both ANN models are quantized to eight bits. Since we use a 4-bit memristor device [20], we use the bit-slicing technique to map the weights to memristors. We perform these simulations on several different crossbar models for LeNet and AlexNet. We generate these different models by selecting the line-resistance, i.e., source-line and bit-line resistance (Figure 1), from a normal distribution, $N(\mu = 2,\ \sigma^2 = 0.5)$. In each circuit model, 1% of devices are randomly considered to be stuck, with 16.2% of all stuck devices being stuck-at-off and 83.8% being stuck-at-on [19].

4.2 Results and discussion

In Section 2, we discussed different major non-idealities associated with memristor-based ANN PEs. To comprehend their effects, we



(a) LeNet



(b) AlexNet

Figure 5: Accuracy results for different recovery techniques. Both figures show accuracy results for different stages of the ReMeCo framework. ReMeCo compared to the CM is more successful in reducing outliers and decreasing the standard deviation of the results. In the rightmost part the numbers on x-axis indicate the layer without any redundancy. For instance the label 3-4 represent the case where all layers in LeNet except layers 3 and 4 have 80% MSB redundancy. (LA: layer averaging)

Table 4: Accuracy, area, and energy evaluations of different implementations. Accuracy and Area are reported in percentage and mm^2, respectively. Energy is in mJ and J for LeNet and Alexnet for 10K test instances, respectively.

ANN	Metric	Base (non-ideal)	Committee Machine	Layer Averaging	Sensitive neuron	MSB (80%)	Sensitive layer
LeNet	**Accur.**	94.4	96.9	97.2	97.2	**97.4**	96.4
	Area	7.23e-3	13.63e-3	13.64e-3	12.35e-3	9.79e-3	**7.53e-3**
	Energy	897.1	1791.2	1791.6	1612.7	1254.9	**1211.1**
AlexNet	**Accur.**	78.11	78.5	78.7	78.8	**78.9**	**78.9**
	Area	1.41	2.78	2.78	2.51	1.96	**1.45**
	Energy	1.84	3.68	3.68	3.31	2.58	**2.54**

run a test-set –comprising 10000 samples from the MNIST dataset– on LeNet ANN model that is implemented on 1000 different circuit models. Due to limited computational resources, for AlexNet-CIFAR10 we run the test-set –with 10000 instances– on a pool of 100 different circuit models. The results are presented in Figure 2 and Table 4. Using LeNet-MNIST and AlexNet-CIFAR10, we show how effectively ReMeCo addresses the non-idealities. We also compare the area and energy overhead that the ReMeCo and the state-of-the-art CM cause [11]. Details are discussed next.

Layer averaging: Layer-by-layer averaging diminishes the error produced in each layer and prevents a large error from propagating through the ANN. Figures 5.(a,b) confirm this and show that by averaging at the end of every layer not only the mean, and median accuracy increase, but also the standard deviation and the number of outlier instances decrease. Table 4 shows that this modification does not cause extra overhead (compared to the CM) if the memristor-based ANN PE already uses the bit-slicing technique to perform high-precision processing.

Sensitive neuron: The key idea behind the second step in the ReMeCo framework is to use the information from ANNs' *sensitivity* analysis to direct the redundant computation toward the *sensitive* neurons. As the second part in Figures 5.(a,b) show, ReMeCo improves reliability, i.e. low STD and few outliers, and increases accuracy. The figures indicate that even after decreasing hardware overhead by 30% the achieved accuracy is similar to that of the state-of-the-art CM. In Table 4, we present the area and energy overhead of the best achieved accuracy using *S-neuron redundancy*.

MSB: The third part of Figures 5.(a,b) show that if we only target the most significant bits of the S-neurons, we can achieve higher accuracy not only compared to the CM but also compared to the S-neuron redundancy. The figure indicates that the STD and the number of outliers are reduced. Table 4 shows that the MSB approach imposes lower overhead compared to both CM and S-neuron redundancy whilst the achieved accuracy is higher.

Sensitive layer: In the right most part of Figures 5.(a,b) it is shown that by applying the last step of the ReMeCo framework we can still deliver a higher accuracy than the CM. However, the STD grows to some extent compared to the previous stage. Table 4 shows that the hardware redundancy is considerably lower compared to all other cases. Therefore, it is a favorable solution for implementations where the resources are constrained.

Area and Energy: Table 4 summarizes the accuracy, area, and energy consumption of a non-ideal memristive ANN PE, the CM, and the best option at every step of the ReMeCo. As the Table shows, the deeper ANN model, i.e., AlexNet, is less affected by the non-idealities. We attribute this to the error resilience nature of ANNs, which become more robust as the network gets deeper. It shows that ReMeCo recovers accuracy by roughly 98.5% for both ANN models. The imposed area overhead for LeNet and AlexNet by ReMeCo are roughly 4% and 3%, respectively which is considerably lower than that of the CM which is at least 100%.

5 CONCLUSION

In this paper, we first performed a thorough DSE to understand the effect of various reliability issues and quantified their impact. Then, we presented ReMeCo, a novel redundancy-based reliability improvement framework. ReMeCo, analyzes the ANN model to be implemented to profile its S-neurons and layers. This information enables us to effectively use redundant calculations to improve the final accuracy of the ANN PE. Our assessments of applying ReMeCo on two deep ANN models showed that we can recover the ANN accuracy by 98.5% while the HW overhead was reduced by more than 20× compared to the committee machine.

REFERENCES

[1] Aayush Ankit et al. 2019. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *ASPLOS 2019*.

[2] Mohammad Ansari et al. 2017. PHAX: Physical characteristics aware ex-situ training framework for inverter-based memristive neuromorphic circuits. *IEEE TCAD* 37, 8 (2017), 1602–1613.

[3] Ali BanaGozar et al. 2017. Robust neuromorphic computing in the presence of process variation. In *DATE 2017*. IEEE, 440–445.

[4] Ali BanaGozar et al. 2019. CIM-SIM: Computation In Memory SIMuIator *(SCOPES '19)*. Association for Computing Machinery, New York, NY, USA, 1–4.

[5] Ali BanaGozar et al. 2020. System simulation of memristor based computation in memory platforms. In *Int. Conf. on Embedded Computer Systems*. 152–168.

[6] Melvin Galicia et al. 2021. Neurovp: A system-level virtual platform for integration of neuromorphic accelerators. In *IEEE 34th ISOCC*. 236–241.

[7] Di Gao et al. 2020. Eva-cim: A system-level performance and energy evaluation framework for computing-in-memory architectures. *IEEE TCAD* 39, 12 (2020), 5011–5024.

[8] Di Gaol et al. 2021. Reliable memristor-based neuromorphic design using variation-and defect-aware training. In *ICCAD 2021*. IEEE, 1–9.

[9] YeonJoo Jeong et al. 2018. Parasitic Effect Analysis in Memristor-Array-Based Neuromorphic Systems. *IEEE Tran. Nanotech.* 17, 1 (2018), 184–193.

[10] Song Jin et al. 2020. On improving fault tolerance of memristor crossbar based neural network designs by target sparsifying. In *DATE 2020*. IEEE, 91–96.

[11] Dovydas Joksas et al. 2020. Committee machines—A universal method to deal with non-idealities in memristor-based neural networks. *Nature communications* 11, 1 (2020), 1–10.

[12] Manuel Le Gallo et al. 2018. Mixed-precision in-memory computing. *Nature Electronics* 1, 4 (2018), 246–253.

[13] Chenchen Liu et al. 2017. Rescuing memristor-based neuromorphic design with high defects. In *54th DAC 2017*. IEEE, 1–6.

[14] Mengyun Liu et al. 2019. Fault tolerance in neuromorphic computing systems. In *Proceedings of the 24th ASP-DAC*. 216–223.

[15] Seyed Hossein Hashemi Shadmehri et al. 2022. SySCIM: SystemC-AMS simulation of memristive computation in-memory. In *DATE 2022*. IEEE, 1467–1472.

[16] Ali Shafiee et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ISCA 2016*. 14–26.

[17] Wonbo Shim et al. 2020. Two-step write–verify scheme and impact of the read noise in multilevel RRAM-based inference engine. *Semiconductor Science and Technology* 35, 11 (2020), 115026.

[18] Swagath Venkataramani et al. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *ISLPED 2014*. IEEE, 27–32.

[19] Qi Xu et al. 2021. Reliability-driven neuromorphic computing systems design. In *DATE 2021*. IEEE, 1586–1591.

[20] Chris Yakopcic et al. 2014. Efficacy of memristive crossbars for neuromorphic processors. In *IJCNN 2014*. IEEE, 15–20.

[21] Baogang Zhang et al. 2019. Handling stuck-at-faults in memristor crossbar arrays using matrix transformations. In *Proceedings of the 24th ASP-DAC*. 438–443.