On-Chip Deep Neural Network Storage with Multi-Level eNVM

Marco Donato, Brandon Reagen, Lillian Pentecost, Udit Gupta
David Brooks, Gu-Yeon Wei
Harvard University
Cambridge, MA

ABSTRACT

One of the biggest performance bottlenecks of today's neural network (NN) accelerators is off-chip memory accesses [11]. In this paper, we propose a method to use multi-level, embedded non-volatile memory (eNVM) to eliminate all off-chip weight accesses. The use of multi-level memory cells increases the probability of faults. Therefore, we co-design the weights and memories such that their properties complement each other and the faults result in no noticeable NN accuracy loss. In the extreme case, the weights in fully connected layers *can be stored using a single transistor*. With weight pruning and clustering, we show our technique reduces the memory area by over an order of magnitude compared to an SRAM baseline. In the case of VGG16 (130M weights), we are able to store all the weights in 4.9 mm^2, well within the area allocated to SRAM in modern NN accelerators [6].

ACM Reference Format:
Marco Donato, Brandon Reagen, Lillian Pentecost, Udit Gupta and David Brooks, Gu-Yeon Wei. 2018. On-Chip Deep Neural Network Storage with Multi-Level eNVM. In *DAC '18: DAC '18:The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196083

1 INTRODUCTION

Recent advancements in deep learning have enabled neural networks (NNs) to achieve state-of-the-art results in various classification and regression applications. However, there remain many challenges in using NNs on modern system-on-chips (SoCs) as the models are large and often require special hardware for timely execution. One major bottleneck in NN performance is off-chip memory accesses. NNs can require hundreds of MBs to store model weights, which easily exceeds the on-chip SRAM capacities of most reasonable designs. This forces designers to rely on costly accesses to off-chip DRAM. In this paper, we propose a method to eliminate all off-chip weight accesses by leveraging the fault tolerance of NN weights to store them in dense, multi-level memory cells. By co-designing the weight and memory properties, we show that even large models such as VGG16 can be stored entirely on chip without negatively impacting model accuracy.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196083

To increase the effective on-chip memory capacity, researchers have proposed various eNVMs. eNVMs are more dense and energy efficient than the SRAM alternative. Examples of existing eNVMs include resistive random access memories (ReRAM), phase change memories (PCM), and spin-transfer torque magnetic RAM (STT-RAM) [4]. However, these implementations require significant changes to manufacturing processes, which makes it challenging to integrate them into modern SoCs. Charge-trap transistors (CTTs) do not require any additional manufacturing cost, providing a promising alternative [13, 15]. CTTs are based on standard high-k devices, making them easy to integrate into SoCs with standard digital logic manufacturing processes.

To further improve eNVM density, multiple bits can be stored in a single memory cell using multi-level cell (MLC) programming. While achieving MLC storage is possible, packing more bits per cell increases the probability a cell experiences a fault. At high fault rates, the benefits of denser memory may be negated by expensive fault detection and correction hardware.

In this study, we show how CTT-MLCs can be leveraged to eliminate off-chip NN weight accesses. To address the faults incurred from storing multiple bits per device, we experiment with how many cells and levels per cell to use for fixed-point weight representations. To optimize our implementation, we co-design the weights and CTT-MLC device properties by: (i) clustering the weights to require fewer cells, (ii) using non-sequential level encodings to mitigate the effects of faults, and (iii) pruning the network to skew the distribution of weights and leverage the non-uniformity of fault probability in CTT-MLCs. While we focus on CTT-based eNVMs, this approach is generalizable to any eNVM with similar properties.

This paper makes the following contributions:

- We show that CTTs can be used as multi-level cells with measurements from a fabricated test chip in a 16nm FinFET process, which we use to build a fault model for various levels per cell. We extend NVSim [7] to model the memory cell and architecture.
- We use our fault model to demonstrate the feasibility of using CTT-MLCs to store NN weights, and we find that even a naive implementation uses as few as 3 transistors per weight.
- Finally, we provide co-design optimizations to reduce area and limit the number and effects of MLC faults. These provide up to 14.4× area savings compared to SRAM and only require a single transistor per weight value in fully connected layers. With optimized CTT-MLCs, all of VGG16's 130M weights fit in 4.9mm^2, which is smaller than the area modern NN chips allocate to SRAM.

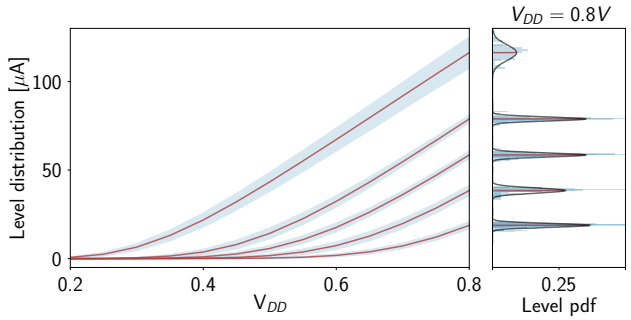

Figure 1: Measurements from a 16nm FinFET test chip we fabricated. The left plot shows the CTTs mean current and 2σ confidence intervals for different programmed levels as a function of V_{DD}. The right plot shows the distribution of the cell currents measured at V_{DD} = 0.8 V.

2 eNVM

Embedded non-volatile memories are of particular interest for increasing the storage density in resource-constrained SoCs; the basic eNVM cell tends to have a smaller area compared to standard 6T-SRAM cells. Moreover, the ability to be powered off and retain the stored values significantly reduces leakage power consumption. These improvements come with a penalty in terms of write latency, which makes eNVMs amenable to NN inference applications where weights are written only once. This section presents an overview of CTT devices, measurements from a fabricated test chip, SPICE simulation results used to build a CTT model, and a comparison of CTT memories with alternate eNVM and volatile memories using NVSim.

2.1 CTT overview

Similar to floating gate devices, CTTs work by altering the threshold voltage, V_{th}, of transistors. The cell, a single transistor, is programmed by trapping charge in the high-k gate oxide via hot carrier injection and self-heating [12]. The difference between the current of an unprogrammed and a programmed device can then be used to discriminate between binary values. This behavior has been demonstrated in bulk, FD-SOI, and FinFET processes [13].

Prior work has used CTTs as single-level memory cells [12] and as crossbar arrays for neuromorphic applications [8]. To enable denser on-chip storage, we propose using CTTs as *multi-level* memories. By changing the programming voltage and pulse width, experiments show that the threshold voltage can be adjusted to a specific target. This can then be used to change a transistor's saturation current; the current of each programed CTT can be interpreted as a discrete level, hence enabling MLCs.

2.2 Measurements and simulation setup

To demonstrate the feasibility of the CTT-MLC approach, we provide measurements from a NOR CTT-MLC memory test structure fabricated using a 16 nm FinFET process. Figure 1 shows the results from programming devices on the test arrays with five distinct levels. Within the figure, the plot on the left shows the measured results with respect to the mean current and 2σ confidence intervals across the devices in the same column. On the right, the statistical distribution of the saturation currents at V_{DD} = 0.8V are shown.

Table 1: Comparison of several eNVMs and volatile memories for 2MB capacity generated using NVSim. (*) NAND Flash and eDRAM assume a 40nm technology node, whereas all other memory technologies are based on a 22nm technology node. CTT-MLC assumes 16-levels per cell; all other memories are based on single level encoding. The read latency and bandwidth is also provided for all memories.

	Cell size $[F^2]$	Area $[mm^2]$	Latency $[ns]$	BW $[GB/s]$	CMOS compatible
SRAM	146	1.30	1.99	15.5	yes
eDRAM*	33	1.82	0.39	21.6	no
NAND Flash*	4	0.60	89.8	22.9	no
ReRAM	4	0.31	0.75	13.7	no
CTT-SLC	8	0.81	0.86	11.4	yes
CTT-MLC	8	0.38	0.57	61.0	yes

The measurements provide insights on how to best model CTT structures. First, the saturation current of the transistor can be precisely tuned to an arbitrary value, enabling MLC programming. Second, the histogram of the saturation current of each level can be modeled as a Gaussian distribution, which coincides with previous findings in MLC NAND flash memories [3]. The measurements also highlight the distinction between the distribution of the initial state and the remaining programmed states; this will later be an important aspect of co-designing NN and CTT-MLCs. Finally, these measurements provide a realistic current range for the cell's programmable levels.

The maximum and minimum saturation current derived from the measurements set the programming range, as shown by the blue and orange curves in Figure 2. Then, the intermediate programming levels are simulated in SPICE by introducing a threshold voltage shift in a modified version of the transistor's BSIM model. The mean and variance of the threshold voltage are set to match the values of the initial and programmed states from the measurements in Figure 1 *i.e.*, a wider distribution is used for the initial state and a narrower one for the programmed states. Based on these V_{th} distributions, a set of Monte Carlo simulations is used to characterize the current level distributions. We found it was reasonable to simulate between 2 and 16 levels per CTT cell.

2.3 Memory technology comparison

We used NVSim to model a CTT-based NOR memory and compare it to alternative memory technologies in terms of area, read latency, and read bandwidth.

The CTT cell size and aspect ratio are based on data from previously fabricated CTT memories [15]. Table 1 shows a comparison of the results across different technologies for 2MB memories, as this capacity is representative of on-chip storage capabilities in modern SoCs using SRAM. Though the fixed 2MB capacity is not optimal for all implementations, some generic conclusions can still be drawn about the different memory technologies. NAND flash memories and ReRAMs have the smallest cell size and commensurately the best density. However, NAND Flash are penalized by higher read latency. Although ReRAMs offer competitive area and latency to CTTs, they require a separate fabrication process. eDRAMs have

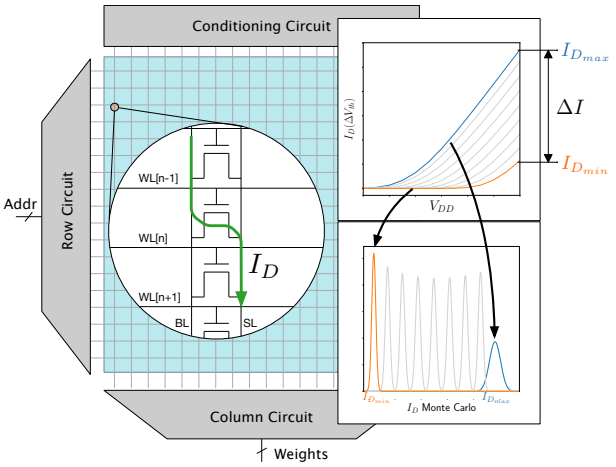

Figure 2: SPICE simulation for extracting the current distributions. The I_{Dmax} and I_{Dmin} values are based on measurement results, while the intermediate programmed levels are obtained from Monte Carlo simulations.

reasonable density, latency, and bandwidth, but are not CMOS compatible. Moreover, eDRAMs require refreshes, making them particularly unsuitable for energy-constrained applications. Thus, the CMOS compatibility of CTTs makes them an attractive solution for integrated, on-chip, dense eNVM.

Though NAND Flash and ReRAM memories, along with other eNVMs, have MLC capabilities, the results in Table 1 are based on SLCs. Thus, the benefits of MLC encoding are seen by comparing the area and read bandwidth of CTT-MLCs and CTT-SLCs. The area overhead for parallel sensing in CTT-MLCs is accounted for by setting the number of sense amplifiers equal to the maximum number of reference levels per cell. We conclude this section by considering another currently available eNVM technology, STT-RAM. Although CMOS-compatible, this technology suffers several shortcomings when applied to our case study. STT-RAM cells can also be designed for multilevel storage, but this requires stacking multiple MTJs in order to achieve more than two resistive states [2], limiting memory density compared to other MLC-capable technologies. Moreover, the additional process steps required to manufacture MTJs can significantly impact overall fabrication cost [1]. Hence, we decided not to include this memory in our analysis.

3 MODELING FRAMEWORK

In this section, we generalize the measurements of Section 2 into a fault and area model for CTT memories. This modeling framework allows us to vary the number of levels per cell and understand the effects of MLC faults on NN applications. We further propose two methods for storing weights on MLC cells based on fixed-point quantization and k-means clustering.

3.1 CTT-MLC error model

A fault in a CTT-MLC cell is defined as a device being incorrectly read as a level adjacent to the intended one. The probability of a fault occurring for a given level, L_n, is determined by two reference thresholds: $I_{\text{ref n}}$ and $I_{\text{ref n-1}}$; reference thresholds discretize current ranges of the CTT cell. Figure 3 shows the current level

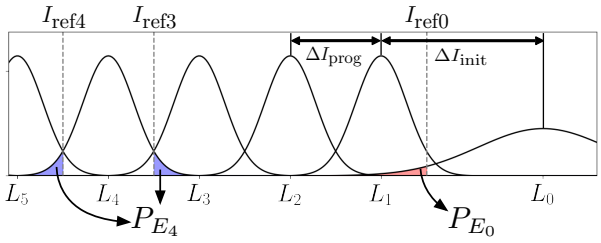

Figure 3: Example of the level current distributions showing fault probabilities for the initial level, L_0, and one programmed level, L_4.

distributions for a generic MLC cell. ΔI_{init} represents the distance between the mean value of the initial state, L_0, and the mean value of the first programmed state, L_1. The remaining programmed levels are equally spaced by ΔI_{prog}. These deltas are tunable and determine the fault rate between levels, which introduces co-design opportunities.

Figure 3 highlights the probabilities of two possible cell faults: P_{E_0} and P_{E_4}. We first consider the case of a fault in a cell set to the initial state, L_0, to be incorrectly read as a cell in the first programmed state, L_1. The probability of this fault (P_{E_0}) occurring is given by the total probability of the initial state's distribution beyond the reference current I_{ref0}.

$$P_{E_0} = P(I_{\text{cell}} < I_{\text{ref0}}) = \int_{-\infty}^{I_{\text{ref0}}} P_{L_0}(x)dx \tag{1}$$

In the case of a cell programmed to a specific level (i.e., not in the initial state), the cell could fail by being read as either of the two adjacent levels. In Figure 3, this is shown as P_{E_4}. Thus, the probability of a fault is given by the sum of the probabilities of having L_4 fall in either of the erroneous ranges:

$$P_{E_4} = [1 - P(I_{\text{cell}} < I_{\text{ref3}})] + P(I_{\text{cell}} < I_{\text{ref4}}) \tag{2}$$

Given ΔI_{init} and the number of levels per cell, a detailed fault model for CTT-MLC is constructed. ΔI_{prog} results from partitioning the remaining ΔI after ΔI_{init} is allocated. This allows us to make P_{E_0} arbitrarily small in order to protect the initial state at the cost of increased fault rates in the programmed levels. Faults are assumed to only result in one level shift and the probability of a fault resulting in a multi level value shift is negligible for the cell configurations considered here. The effects of current to voltage conversion on the distribution are taken into account.

3.2 Encoding methodology

This paper experiments with two types of weight quantization: fixed-point and clustering. The details for storing both flavors of quantized weights in CTT-MLC are presented here.

3.2.1 Fixed-point non-uniform encoding. Fixed-point data type quantization is a well-known and effective hardware optimization technique; reducing the width of data types can substantially reduce area and power dissipation while improving performance. In NNs, fixed-point quantization can be aggressively applied to weights. The 32-bit floating point types used for training weights can be reduced to use only 8-12 bits for inference without compromising accuracy.

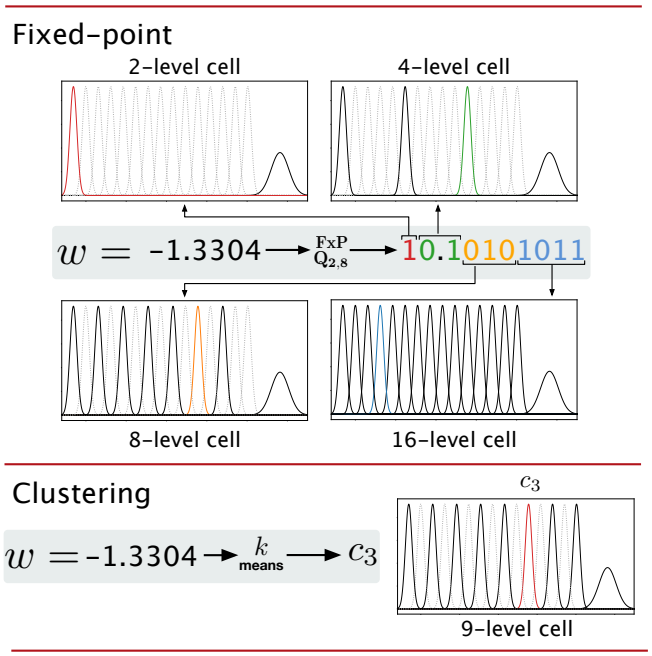

Figure 4: Example of fixed-point non-uniform encoding (top) and cluster-based encoding (bottom) using MLCs.

As a example, consider the conversion of the value $(-1.3304)_{10}$ and its fixed-point representation $(10.10101011)_2$ using two integer bits and eight fractional bits (see Figure 4). If this value is stored using binary encoding with CTT-SLC, then 10 cells are needed (one per bit). The same value can be stored using only 3 CTT-MLCs if the most dense, 16 level (4 bits), cell configuration is used. This uniform configuration reduces the area cost per bit at the expense of a higher fault rate. Since faults on higher order bits have a disproportionate impact on the stored value, we anticipate that it is advantageous to selectively mitigate these faults.

Figure 4 shows our proposed alternative: a non-uniform encoding where 4 cells are used with 2, 4, 8, and 16 levels per cell, from MSB to LSB (labeled as 248F using hexadecimal notation). The example shows that the sign and integer portion of the weight can be protected by using a CTT-MLC with fewer levels. Compared to uniform encoding, this non-uniform encoding requires only one additional cell and substantially reduces the fault rates in weight MSBs.

3.2.2 Clustering. The second weight quantization approach we consider is k-means clustering [9], in which NN weights are mapped onto a set of k values on a per-layer basis. Clustering is advantageous as only the indexes need to be stored per weight and $\lceil \log k \rceil$ is typically significantly less than the number of bits required with fixed-point quantization. The overhead for storing the look-up table of the actual k weight values is negligible.

Storing cluster indexes enables two optimization opportunities. The first technique is an intra-cell optimization to protect the initial state from faults by increasing ΔI_{init}. This technique is particularly effective when the majority of the parameters are assigned to a single cluster, a property we actively enforce with weight pruning. We further consider three ways to map clusters to levels to mitigate the magnitude and fault rates, see Table 3. The second technique is an inter-cell, multi-cell per weight optimization. If more than one cell is required to store the cluster index, then some clusters can be more protected via the asymmetrical fault rates of MLC and non-uniform encoding, *i.e.*, a non-uniform allocation of cluster index bits across cells.

Table 2: For each model, we report the quantization (Q) in the form integer.fractional bits, the number of clusters (K), and the configurations for each encoding scheme: fixed-point (FxP), clustered (C), and pruned & clustered (P+C). For VGG16, the two values of K, config, and level map (Table 3) given represent the values for CNN layers and FC layers, respectively.

Model	Encoding	Weights #	Error [%]	Q int.frac	K #	Config	Level map	Area [mm^2]
LeNetFC	FxP	270K	1.91%	2.6	-	488	S	0.033
	C				16	F	S	0.008
	P+C				12	C	Zero	0.008
LeNetCNN	FxP	600K	0.85%	2.8	-	4488	S	0.076
	C				8	8	S	0.021
	P+C				11	B	MDI	0.021
VGG16	FxP	135M	37.8%	2.10	-	4444FF	S	40.53
	C				64/8	2244/8	S/MDI	5.7
	P+C				64/9	444/9	S/MDI	4.9

4 RESULTS

To quantify the benefits of using CTT-MLC eNVMs, we use two zero added cost baselines: SRAM and CTT-SLC. We evaluate the effectiveness of CTT devices to store the weights of three prototypical NNs listed in Table 2: LeNetFC is a three-layer fully-connected (FC) network, LeNetCNN is a five-layer convolutional neural network (CNN), and VGG16 is a much larger CNN [18]. LeNetFC and LeNetCNN use the well-known MNIST dataset for handwritten digit classification, and VGG16 uses the popular ImageNet dataset of colored images to be classified into 1000 possible classes [14, 17].

The fault injection simulations are implemented using the Keras framework [5]. For both fixed-point and cluster representations, a corresponding encoding transform function is defined. Starting from a trained model, the encoding transform is applied to the original parameters on a per-layer basis. Next, the faulty cells are randomly chosen using the error probability based on the number of levels per cell and the specific level value. The transformed parameters are used to evaluate the accuracy of the model for a specific encoding configuration.

Section 4.1 presents the benefits of using CTT-MLC memories when storing NN parameters that are quantized with fixed-point representation. Section 4.2 explores k-means clustering to reduce the area footprint of storing NNs. Finally, Section 4.3 improves on the preliminary MLC approach by co-designing the NN storage scheme with characteristics of faults in CTT-MLC memories.

4.1 Storing fixed-point values in CTTs

As discussed in Section 3.2.1, non-uniform MLC encoding is used to shift the fault rate from the MSBs to the LSBs while reducing the total number of cells compared to binary representation. To find the configuration that minimizes the area footprint while maintaining model accuracy, all possible configurations of levels per cell and number of cells are tested for each NN. The results of this exploration for LeNetCNN are shown in Figure 5. The discrete steps in area correspond to sweeping the number of cells to encode each

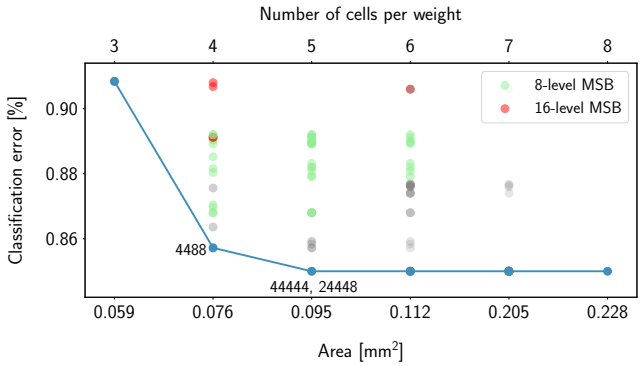

Figure 5: Model error as a function of memory area (bottom) which is determined by the number of cells per weight (top) for LeNetCNN with fixed-point quantization. Pareto configurations are labeled for configurations with 4 and 5 cells. The red and green dots show configurations using 8 and 16 levels for the MSBs.

parameter in LeNetCNN from 3 to 8. Configurations above 8 cells are not shown because they all have a classification error of 0.85%. Each of these configurations is averaged over 25 experiments with different fault patterns for statistical significance.

To incur no loss in accuracy, 5 cells can be used in either of two configurations of levels per cell: 24448 or 44444, which correspond to an area of $0.095mm^2$ in Figure 5. Alternatively, a configuration with 4 cells (4488), which corresponds to an area of $0.076mm^2$, results in an error of 0.857%. This error is within the statistical variance of the model error due to training noise [16]. All experiments using 8 or 16 levels to encode the MSBs (green, red) lie above the pareto frontier. Other configurations using 16 levels for a cell holding MSBs result in model error up to 6.3%. This is intuitive, as using more levels to encode the MSBs incurs higher magnitude errors. Repeating this procedure for the other NNs, we find that fixed-point CTT-MLC encoding requires 3 transistors per parameter for LeNetFC and 6 for VGG16, as listed in Table 2. Compared to an SRAM baseline, this results in a total area reduction of up to 7.6×, as shown in Figure 6.

4.2 Reducing area with parameter clustering

As discussed in Section 3.2.2, another way to quantize NNs is with k-means clustering. Clustering is often preferable to fixed-point quantization because only cluster index pointers are stored. All three NN models require between 8 and 64 clusters to preserve model accuracy, which requires just 3 to 6 bits to represent each parameter. When encoding clustered parameters in CTT-MLC, the benefits are immediate. For LeNetFC and LeNetCNN, *each parameter can be stored in a single transistor* because only 16 and 8 clusters are needed to preserve accuracy for each model. Compared to fixed-point quantization, k-means clustering saves 3.8× and 3.6× area when storing LeNetFC and LeNetCNN in CTT-MLCs. Compared to SRAM, storing the clusters in CTT-MLCs saves 14.4× and 13.2× for LeNetFC and LeNetCNN, respectively.

Up to this point, we assumed that all layers in a network use the same MLC configuration to store NN parameters. This assumption results in pessimistic storage estimates for larger models. For

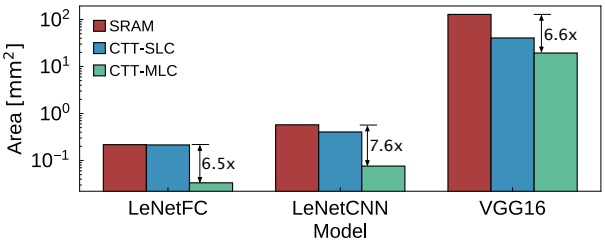

Figure 6: Area comparison for SRAM, CTT-SLC, CTT-MLC for fixed-point encodings.

Table 3: Cluster to level mapping alternatives where L_0 corresponds to the initial level for an example cell with 9 total levels. The clusters are labeled by value c_4^- through c_4^+, with the most populous cluster designated as c_0.

	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
S	c_4^-	c_3^-	c_2^-	c_1^-	$\mathbf{c_0}$	c_1^+	c_2^+	c_3^+	c_4^+
Zero	$\mathbf{c_0}$	c_4^-	c_3^-	c_2^-	c_1^-	c_1^+	c_2^+	c_3^+	c_4^+
MD-I	$\mathbf{c_0}$	c_2^-	c_4^-	c_3^-	c_1^-	c_1^+	c_2^+	c_3^+	c_4^+
MD-II	$\mathbf{c_0}$	c_2^+	c_4^+	c_3^+	c_1^+	c_1^-	c_2^-	c_3^-	c_4^-

instance, in VGG16, 89.5% of the parameters are in the FC layers. These FC layers need only 8 or 9 clusters, while the CNN layers need up to 64 to maintain model accuracy. Using the same number of cells for parameters in the FC and CNN layers is wasteful as, once again, we *need a single transistor per parameter* to encode the parameters in the FC layers. For VGG16, 1 cell is used for each FC parameter and 4 cells are used for each CNN parameter, and this use of heterogeneous configurations saves us 3× area. For VGG16, CTT-MLCs provide 9.8× total area savings compared to SRAM.

4.3 Additional optimizations for clustering

To further optimize the proposed technique, we co-design NN and CTT-MLC device properties.

4.3.1 Skewing weights to leverage CTT initial state. In CTT-MLC devices, the initial state (see Figure 3) provides a unique opportunity for optimization: because it can be deliberately separated from the programmed states, the fault probability can be skewed to protect the most frequent cluster. To further leverage this protected state, we propose pruning small parameter values by setting them to zero. Previous work demonstrated that pruning is a highly effective optimization that can set upwards of 90% of weight values to zero [10]. Therefore, mapping the zero-valued cluster of the pruned networks to the initial state has the overall effect of reducing the number of faults. As an example, pruning LeNetFC reduces the raw number of faults across all layers by 89%.

In VGG16, we prune 97% of all parameters in the first FC layer, which has over 100M parameters. This effectively protects a much larger proportion of the parameters compared to the unpruned network. In the CNN layers, pruning reduces the number of cells needed to store parameters from 4 to 3, leading to a further area reduction of 1.17× over CTT-MLC with clustering.

4.3.2 Cluster encoding ordering. Another optimization considers how the ordering of clusters assigned to levels within a cell affects model accuracy. Four different cluster-to-level mappings are enumerated in Table 3. The most trivial is sequential (S), where the level mapping follows the order of the clusters. The rest of the mappings assign the most frequent cluster to the initial state. The zero mapping assigns the remaining clusters sequentially, while the minimum distance (MD) mappings minimize the fault magnitude between adjacent levels. For LeNetFC and LeNetCNN, the level mapping corresponding to negligible accuracy loss changed after pruning and differed between the two models, as indicated in Table 2. For VGG16, the level mapping did not significantly affect CNN results, and the MD-I scheme was preferable for the FC layers.

Figure 7 summarizes results for storing NNs using the optimized encoding scheme including clustering, pruning, and level mapping. Using CTT-MLCs allows us to store LeNetFC and LeNetCNN in 0.009 mm^2 and 0.021 mm^2, which corresponds to a 14.4× and 13.2× savings in area compared to SRAM. VGG16 can be stored in 4.9 mm^2, which is a 10.6× reduction over SRAM.

Based on the memory footprint of a recently published NN SoC fabricated using a 28 nm process, our proposed techniques for co-designing NNs with CTT-MLCs allow us to store a large model like VGG16 entirely on chip [6].

5 CONCLUSION

We present a co-design methodology based on the concurrent optimization of CTT-MLC eNVM and NNs. The storage of fixed-point parameters can be optimized using a non-uniform encoding that protects the sign and integer bits using fewer levels per cell, and this solution gives up to 7.6× area savings. As a further optimization, using k-means clustering together with MLC storage requires *just a single transistor for each parameter* in FC layers. Additionally, pruning NN parameters and fine-tuning the cell level distributions protects the most populous cluster and allows for more aggressive encoding configurations for CNN layers as well. The concurrent adoption of these optimizations reduces the memory footprint of VGG16 to a total area 4.9 mm^2, which can be reasonably integrated in a modern SoC. While this work focused on a specific eNVM implementation, our co-design methodology can be extended to any eNVM technology capable of MLC storage. This aspect makes our approach generalizable to other popular eNVM implementations such as ReRAM and PCM.

6 ACKNOWLEDGEMENTS

This work was supported in part by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA. The work was also partially supported by the U.S. Government, under the DARPA CRAFT and DARPA PERFECT programs. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. Reagen was supported by a Siebel Scholarship.

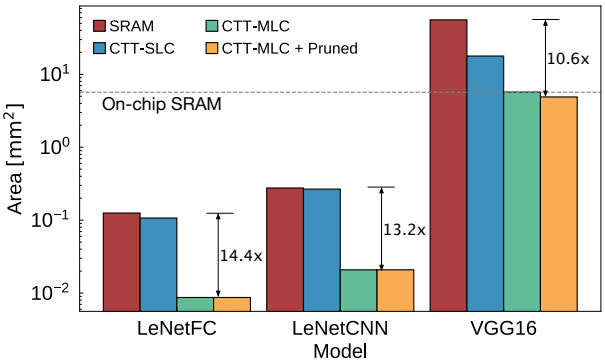

Figure 7: Area comparison for SRAM, CTT-SLC, and CTT-MLC for clustered encodings. Area for CTT-MLC with clustered and pruned encodings is also given (orange). The horizontal gray line indicates the memory area footprint from a recently published NN SoC [6], showing that our most optimized encoding enables VGG16 to fit on chip.

REFERENCES

[1] I. Bayram, E. Eken, D. Kline, N. Parshook, Y. Chen, and A. K. Jones. Modeling STT-RAM fabrication cost and impacts in NVSim. In *IGSC*, 2016.
[2] X. Bi, M. Mao, D. Wang, and H. Li. Unleashing the potential of MLC STT-RAM caches. In *ICCAD*, 2013.
[3] Y. Cai et al. Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling. *DATE*, 2013.
[4] A. Chen. A review of emerging non-volatile memory (NVM) technologies and applications. *Solid. State. Electron.*, 2016.
[5] F. Chollet et al. Keras, 2015.
[6] G. Desoli et al. A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. *ISSCC*, 2017.
[7] X. Dong et al. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2012.
[8] Y. Du et al. A Memristive Neural Network Computing Engine using CMOS-Compatible Charge-Trap-Transistor (CTT). *CoRR*, 2017.
[9] Y. Gong et al. Compressing Deep Convolutional Networks using Vector Quantization. *CoRR*, 2014.
[10] S. Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ICLR*, 2016.
[11] N. P. Jouppi et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. *ISCA*, 2017.
[12] F. Khan et al. The Impact of Self-Heating on Charge Trapping in High-k-Metal-Gate nFETs. *IEEE Electron Device Lett.*, 2016.
[13] F. Khan et al. Charge Trap Transistor (CTT): An Embedded Fully Logic-Compatible Multiple-Time Programmable Non-Volatile Memory Element for high-k-metal-gate CMOS technologies. *IEEE Electron Device Letters*, 2017.
[14] Y. LeCun and C. Cortes. The MNIST database of handwritten digits.
[15] K. Miyaji et al. Zero Additional Process, Local Charge Trap, Embedded Flash Memory with Drain-Side Assisted Erase Scheme Using Minimum Channel LengthWidth Standard Complemental Metal-Oxide-Semiconductor Single Transistor Cell. *Jpn. J. Appl. Phys.*, 2012.
[16] B. Reagen et al. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. *ISCA*, 2016.
[17] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.
[18] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, 2014.