SAME-Infer: *S*oftware *A*ssisted *M*emory Resilience for *E*fficient *Infer*ence at the Edge

Irina Alam
irina1@ucla.edu
University of California, Los Angeles
Los Angeles, California

Puneet Gupta
puneetg@ucla.edu
University of California, Los Angeles
Los Angeles, California

ABSTRACT

Design of edge devices is driven by the need for the lowest possible cost and energy consumption. Both of these are strongly affected by on-chip memories as they often constitute a large fraction of embedded processors. One way to reduce energy consumption is by reducing the supply voltage. However, this causes memory cell hard fault rates to rise exponentially, thus degrading yield at low voltage and increasing cost. Also the weaker memory cells often lead to worsened chip yield and mean-time-to-failure. Deep learning neural network applications constitute a significant fraction of the workloads that are run today on these low cost embedded devices. Despite the inherent resilience of most of these deep learning applications, inference accuracy degrades significantly at high fault rates. We propose SAME-Infer, a **s**oftware **a**ssisted **m**emory resilience technique for **e**fficient **infer**ence at the edge. It is a fault-aware linking methodology for software managed embedded memories to efficiently map the critical code/layers onto the non-faulty segments of the memory and the non-critical fault tolerant sections in the faulty or error-prone memory segments. This is done in a way such that memory hard faults can be tolerated and voltage be lowered without degrading the accuracy (***SAME infer***ence accuracy at lower voltage/higher error rate). Our evaluation on 10 real microcontroller class chips shows that more than 175mV reduction in voltage can be achieved without any loss in accuracy for a variety of neural networks. SAME-Infer can also be considered as an efficient fault tolerance/in-field repair technique as it tolerates on average 25x (upto 350x) increase in bit error rate with minimal impact on inference accuracy.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; **Embedded systems**; • **Hardware** → **Fault tolerance**; **Hardware reliability**; • **Software and its engineering** → *Embedded software*; • **Computing methodologies** → **Neural networks**.

KEYWORDS

voltage scaling, fault resilience, inference, embedded memory

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA

ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422774

ACM Reference Format:
Irina Alam and Puneet Gupta. 2020. SAME-Infer: *S*oftware *A*ssisted *M*emory Resilience for *E*fficient *Infer*ence at the Edge. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3422575.3422774

1 INTRODUCTION

The demand for deploying deep learning neural network (DNN) algorithms in edge and mobile devices is increasing. These applications are extremely compute intensive. Since the edge devices are often energy constrained, it is critical to enhance the energy efficiency of DNN inference on such devices. Further, these edge devices are deployed in increasingly harsh environments resulting in worsened hardware failure rates [8] but still require continued reliable operation. To make matters worse, typical fault tolerance techniques (sparing, system-level fault tolerance, error correcting codes) are usually unaffordable due to cost or energy reasons in these contexts. Furthermore, many of the faults are permanent or semi-permanent and possibly wearout related. As a result, in-field repair/replace, though needed, is very difficult in many environments.

The primary characteristics of embedded systems at the edge of the Internet-of-Things (IoT) are low cost and low energy consumption, which are both strongly affected by on-chip memories [18]. On-chip SRAM-based embedded memories occupy a large fraction of chip area and consume a significant portion of the overall system energy. To make these memories efficient, the embedded systems community has increasingly turned to software-managed on-chip memories – also known as *scratchpad memories* (SPMs) [27] – due to their 40% lower energy as well as latency and area benefits compared to hardware-managed caches [11].

One way to reduce the energy consumption of these on-chip memories is by reducing the supply voltage. However, doing so leads to an exponential rise in the memory cell hard fault rate. Also, due to manufacturing variability, some memory cells turn out to be weaker (or faulty) and often leads to bit-failures which affects yield and, in turn, cost of these edge devices. These weak memory cells also constrain the lowest voltage an SPM can be run at, and are prone to aging induced failures. Running applications on a voltage scaled device with faulty memory leads to erroneous behaviour of the application.

On the other hand, DNN algorithms are known to be approximation friendly and fault resilient [26]. Previous works have shown that if a few elements in the weight matrix or inputs are erroneous, the final inference accuracy remains unchanged. These errors often do not get propagated to the output or the perturbation these errors

cause is negligible enough such that the final classification is not affected. A recent body of work has focused on exploiting this characteristic of Deep Learning (DL) networks by reducing precision of computation through quantization as it does not affect accuracy but significantly reduces storage requirement [17, 21]. Therefore, DNN algorithms have an inherent capability of masking errors due to memory faults if the errors occur in non-critical locations.

In this work we propose SAME-Infer, a lazy link-time fault-aware memory mapping approach for Deep Learning networks. SAME-Infer extends the software construction toolchain (compiler and linker), for software managed memories, to intelligently map the critical regions of the network in non-faulty memory segments and non-critical fault tolerant regions in the faulty segments of the memory. Contributions of this work include:

- We study the impact of memory faults on inference accuracy by approximating the error tolerance of each layer's weight and activation values. We extend this methodology to approximate the error tolerance of every weight kernel (per layer, per filter). Further, we use this methodology to bin the data sections of a DNN program based on varying degrees of criticality.
- We develop SAME-Infer approach to relink compiled program based on memory fault map and the criticality measure of the data partitions.
- We develop analytical models for predicting probability of successful relinking given the program and the expected bit error rate (BER).
- We evaluate the SAME-Infer approach on ten real microcontroller class chips running 8-bit and binarized CNNs and MLPs on well-known MNIST, Google Speech command and CIFAR10 datasets. We measure the achieved energy reduction and fault tolerance. The results show opportunities of memory voltage reduction by up to 175mV and 350x improvement in BER tolerance.
- We also show approaches which try to make trained networks generally robust (e.g., [23]) do not work as well as SAME-Infer.

Thus, SAME-Infer provides a methodology to tolerate (and repair) increased hard fault rate in systems with scratchpad based memories while maintaining the **same infer**ence accuracy for Deep-Learning applications. The increased BER tolerance not only helps to lower supply voltage and save energy, it also helps to tolerate aging induced faults. When in-field memory faults happen, it provides a simple software patch solution alternative to difficult in-field repair and/or expensive hardware replacement solutions. SAME-Infer also provides significant cost saving since device manufacturers can now save chips with fault-prone or faulty cells rather than discarding them.

2 BACKGROUND

In this section we briefly present the essential background to understand our contributions.

2.1 Scratchpad Memories (SPMs)

SPMs are small, on-chip, low latency memories like caches that help to reduce memory access latency by storing frequently accessed data or instructions. However, unlike hardware managed caches, SPMs are software managed, i.e., the placement of data in these memories is orchestrated by the software (compiler/linker or programmer). Most SPMs in today's edge devices are SRAM-based. Such on-chip memories consume significant fraction of the chip area (can be as high as 70% [1]) and contribute significantly to overall power consumption.

2.2 Program Sections and Memory Segments

The Executable and Linkable Format (ELF) is ubiquitous on Unix-based systems for representing program executable images in a portable manner [7]. ELF files contain a header that specifies the Instruction Set Architecture (ISA), Application Binary Interface (ABI), a list of program sections and memory segments, and various other metadata.

- A *section* is a contiguous chunk of bytes with an assigned name: sections can contain instructions, data, or even debug information. For instance, the well-known `.text` section typically contains all executable instructions in a program, while the `.data` section contains initialized global variables.
- A *segment* represents a contiguous region of the memory address space (i.e., ROM, instruction memory, data memory, etc.). When a final output binary is produced, the linker maps sections to segments. Each section may be mapped to at most one segment; each segment can contain one or more non-overlapping sections.

Manipulating the mapping between program sections and segments is the core focus of the proposed SAME-Infer approach.

2.3 SRAM Faults

SRAM faults can be primarily characterized as either soft or hard faults. Soft faults manifest at runtime due to radiation induced high energy particle strikes, value disturbance due to cell leakage etc. Error Correcting Codes (ECC) is a typical approach to deal with soft faults. *Hard faults*, on the other hand, include all recurring and/or predictable failure modes that can be characterized via testing at fabrication time or in the field. These include: manufacturing defects, weak cells at low voltage, and in-field device/circuit aging and wearout mechanisms [13]. Using ECC for low voltage induced hard faults will require a very strong protection scheme with high overheads, making them impractical in the context of low cost platforms. A common solution to hard faults is to characterize the memory, generate a *fault map*, and then deploy it in a system-level mechanism (e.g., page retirement in systems which support virtual memory) to hide the effects of hard faults. However, most IoT devices are bare metal and do not have support for operating system and virtual memory framework due to limited memory capacity and energy budget. Simple solutions used traditionally by designers to increase reliability are including spare rows and columns [30] in the memory arrays and employing large voltage guardbands [16]. Unfortunately, as the voltage is scaled and the fault rate rises exponentially, sparing soon becomes insufficient. Also, large voltage guardbands limit the energy proportionality of memory, thus reducing battery life for duty-cycled embedded systems [34], a critical consideration for the IoT. Although there are several past works that propose approaches for reliable operation in low voltage SRAM

caches [9, 36], they cannot be used in the context of scratchpads and embedded main memory and often incur impractical overheads for low cost devices.

2.4 Fault Resilient DL networks

Most deep learning neural networks are known to be moderately fault resilient because of the abundant redundancy present in these networks. However, the resilience of a DL network depends on the type of data (such as inputs vs. weights), data values, data-types (32-bit float vs 8-bit integer), layer type/position in the network (such as input layer vs. hidden layer, convolution layer vs fully connected layer), etc [26]. Inherent resilience and redundancy in neural networks has also been leveraged to reduce precision of computation [17, 21] or for compression [20]. A recent work [23] focuses on exploiting the fault resilient characteristic of these networks for performance improvements and energy savings in DRAM. However, it requires the network to be retrained on the target approximate DRAM system. Such an approach is often infeasible for low cost, compute/memory starved edge devices. They also proposed offloading the retraining on a separate system using a random bit error rate (BER). However, hard faults in memory (especially at lower voltage in SRAMs) are often correlated and hence, modeling the bit errors as a uniform random distribution is not very accurate (as we show later in this paper). Our fault aware linking approach is similar to [14]. However, unlike [14], we exploit the approximation-tolerant nature of DNNs and use faulty regions of memory to store appropriately non-critical regions of the program, thereby delivering much higher energy reduction/fault tolerance (more than 100mV min-VDD gain) as compared to [14].

3 SAME-INFER METHODOLOGY

Software construction toolchains, by default, consider the memory address space to be contiguous and place the program code and data accordingly. However, with hard faults in the memory, the contiguous placement of data and code can result in the intersection of program sections with faults, making the system and program execution unreliable. SAME-Infer extends the default toolchain so that it is fault-aware and has the ability to incorporate the fault map while placing instructions and data into the memory with faults. Thus, at software deployment time, SAME-Infer, with the help of the modified toolchain, prepares a customized binary for each chip such that all critical sections of the program are placed in non-faulty contiguous memory segments and the non-critical sections in memory segments containing faults.

Figure 1 shows the complete SAME-Infer flow. In order to be able to place program sections into different memory segments efficiently, the monolithic program sections such as *.text* and *.data* need to be split up on a per-function and/or a per-variable basis so that each smaller section can be mapped to a particular memory segment by the fault-aware linker. In order to do that, the programmer initially needs to compile the code using special compiler flags for GCC (`-ffunction-sections` and `-fdata-sections`) so that the compiler can place each subroutine and global variable into their own named sections in the ELF object file. After compiling the code the programmer does not link the object files. In the next step, the object file is parsed using standard ELFIO C++ library [25] to record the name of each program section and its size.

Once the program sections and their sizes are recorded, the sections are then annotated with their criticality level (i.e., how many least significant bit (LSB) errors can be tolerated). The section packing algorithm (described later in Section 3.2) then iteratively maps sections to segments starting with most critical sections first (each criticality level gets its own memory fault map). If the tool provides a feasible solution, a part of the customized linker is generated based on that solution for the critical program sections. The stack and heap are placed in the largest remaining non-faulty contiguous memory segment.

3.1 Fault Impact Analysis

The non-critical sections of the network (in this work we considered weights and activation data as non-critical sections) are fault resilient, but upto a certain degree. Naively placing these non-critical sections in the memory can dramatically impact accuracy. As a result it is important to measure the effect of bit errors in each of these non-critical weights and activation data on overall accuracy. In the ideal scenario, it is required to search for the the highest tolerable bit error rate (maximum error) of each weight and activation data that would still yield an acceptable inference accuracy. However, this search space is exponential, given the total number of weights and activations in a reasonably sized DNN. In order to keep the granularity of sections reasonable, we did layer-wise sensitivity analysis of the weights and activations to understand the impact of each layer's weights and outputs on the overall accuracy.

The approach to calculate inference error sensitivity to bit errors leverages the quantization approach proposed in [28]. We essentially approximate bit errors in the k least significant bits by reducing the precision by k bits. For example, in our 8-bit 2 layer MLP network (weights and activations have 8-bit fixed point precision), the most sensitive weights (layer-1 weights) can be quantized down to 5 bits without loss in accuracy. We interpret this as layer-1 weights can have upto three bit errors in the least significant 3 bits. The fault map for layer-1 weights, thus, will contain all memory addresses where there is an error in the most significant 5 bits.

3.2 Packing Critical and Non-Critical Sections

We solve the section to segment mapping problem iteratively. In a 8-bit network, we allow 5 criticality levels: from 0 to 4 LSB errors. For every criticality level, the corresponding program sections are identified from the sensitivity analyses above. For every chip, the fault map is obtained from a software/BIST memory testing routine. The fault map is different for every criticality level depending on how many LSB errors can be tolerated. Section packing is then solved in criticality order (most critical first) i.e., the section packing algorithm is run 5 times (in the 8 bit case) sequentially each time with a different fault map and available memory segments. A sample packing solution is shown in Figure 2.

The section-packing problem itself, is a variant of the Multiple Knapsacks problem [3] which we solve using an ILP[1] with multiple criticality levels. The objective is to minimize the number of packed

[1]Number of sections/segments is small enough that the ILP runtime was always less than 20 seconds in our experiments.

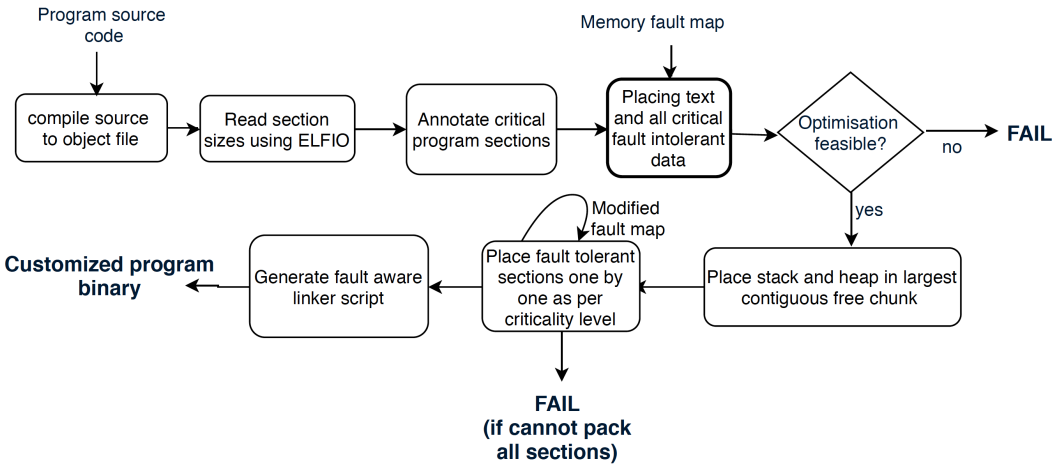


Figure 1: SAME-Infer procedure: given source code of a DL network and a memory fault map, produce a per-chip custom binary executable that will work in presence of known hard fault locations in the SPMs.

memory segments so that there are large enough chunks of memory in between the packed segments to accommodate the program sections for the remaining criticality levels. This objective also helps to naturally provide a solution that avoids memory regions with higher fault densities. The placement algorithm ensures that no weights or activation data of a particular layer intersects with faulty bytes that have errors in the intolerable more significant bit positions. The algorithm also ensures that these non-critical sections do not overlap with the already placed critical program sections. Once this mapping is done, the linker generates the customized binary ready to be deployed on that particular chip.

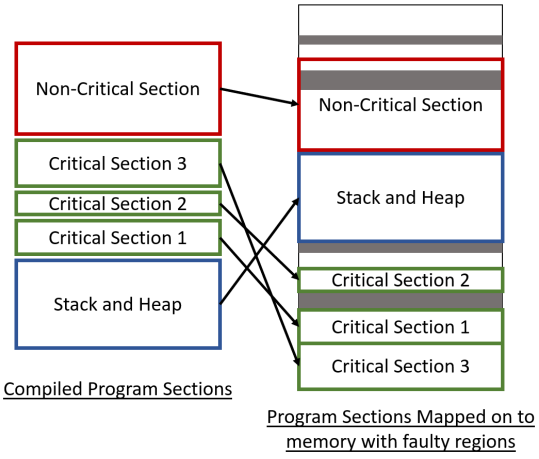


Figure 2: A sample section packing solution provided by SAME-Infer. The critical sections are placed in fault free memory segments while the non-critical sections intersect with faults (grey regions represent fault locations). The stack and heap is placed in the largest non faulty contiguous memory segment remaining after placing the critical sections.

3.3 Breaking up monolithic weight sections into smaller kernels

We observe that SAME-Infer fails when the packing of the largest section fails. A lot of times the largest section turns out to be a data section corresponding to a particular layer's weight. One way to relieve this would be to do a one-time simple modification of the source code where the weight data sections are broken down into smaller sections. A simple way to do this would be to break up the convolution layer weights on a per-filter basis. This splitting induces no code space overhead as the same functions can be used, the only additional step would be to concatenate the final output. The layer-wise sensitivity analysis of the weights in [28] can be modified to compute weight quantization noise gain on a per layer per filter basis as shown below:

$$E_{W,l,k} = \left[\sum_{\substack{i=1 \\ i \neq Y_{fl}}}^{M} \frac{\sum_{h \in W_{l,k}} \left| \frac{\partial (Z_i - Z_{Y_{fl}})}{\partial w_h} \right|^2}{24 \left| Z_i - Z_{Y_{fl}} \right|^2} \right] \quad (1)$$

Here, M is the number of classes, $\{Z_i\}_{i=1}^{M}$ are the soft outputs, Z_{Yfl} is the floating point output, and $\{\{W_{l,k}\}_{l=1}^{L}\}_{k=1}^{N_C}$ are the per layer (L is the total number of layers), per filter (N_c is the number of filters in layer l) weights.

We computed the per filter quantization noise gain for a large CNN (with 6 convolution layers and 3 fully connected layers). The network architecture is $32C3 - 32C3 - MP2 - 64C3 - 64C3 - MP2 - 128C3 - 128C3 - 256FC - 256FC - 10$ using CIFAR-10 dataset [5]. The layer 1 and 2 filter wise quantization noise gain results obtained using Equation 1 are shown in Figure 3. From the results it can be seen that the sensitivity of the weights across kernels varies significantly. Since the precision assignment matches the quantization noise gain profile on a logarithmic scale, using the same number of least significant bits for error tolerance for every weight in a layer might lead to under utilization of available network redundancy for

energy efficiency. In Section 5.2, we analyze the min-VDD benefits of weight splitting.

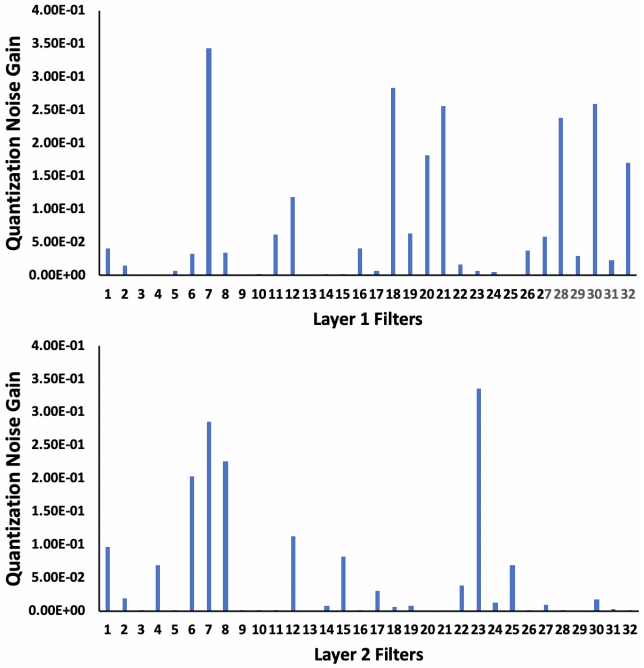


Figure 3: Weight quantization noise gain per filter - layers 1 and 2 of a nine layer CNN.

Finding the best split size: The smallest granularity at which the non critical sections need to be split in order to be able to run at a given voltage can be determined from Equation 3 in the next subsection. Note that going to finer-grained splitting than what is "natural" for neural networks (layer/filter), would require fairly intrusive code changes which we want to avoid. Furthermore, making the split granularity much smaller than section sizes in code part of memory (dictated by the code and not weights/activations) is not useful as code memory will limit the voltage scalability in that case(Figure 7). In this work, we limit splitting to a filter granularity.

Performance and Code Size Overheads. We evaluated the performance and code size overheads of splitting up data and code sections on a per-variable/per-function basis and placing them in non-contiguous memory segments. The performance impact is almost negligible (~ 0.1%). This is because of static allocation of program sections. Also, since stack and heap are not split, there are not any additional performance overheads due to increased pointer chasing. There is no impact on code size since the source code remains unchanged and hence, the size of the final executable that is loaded into the memory also remains unchanged. Even splitting to a filter granularity resulted in minimal code changes and negligible (~ 1%) code size overhead.

3.4 Analytical Critical and Non-critical Section Packing Estimation

The section packing mostly fails when the largest program section is larger than the largest non-faulty contiguous memory segment. We extend the packing failure probability model developed in [14] to account for multiple criticality levels. The analytical model is based on the probability distribution of the longest consecutive sequences of coin flips as provided in [29]. Let L_k be a random variable representing the length of the largest run of heads in k independent flips of a biased coin (with p as the probability of heads). The following equation is an approximation for the limiting behavior of L_k, i.e., the probability that longest run of heads is less than x and assuming $k(1-p) \gg 1$ [29]

$$P(L_k < x) \approx e^{-p^{(x-\log_{p^{-1}}(k(1-p)))}}. \quad (2)$$

We only consider the largest critical program section size ($m_{\max}$). Let b be the i.i.d. bit-error-rate and s be the probability of no errors occurring in a 32-bit word, i.e., $s = (1-b)^{32}$. For a total memory capacity of `size` bytes, we can approximate the probability of there *not* being a memory segment that is large enough to store the largest program section [14]:

$$P\left(L_{\texttt{size}/4} < \frac{m_{\max}}{4}\right) \approx e^{-s^{\left(\frac{m_{\max}}{4} - \log_{s^{-1}}\left(\frac{\texttt{size}}{4}(1-s)\right)\right)}}. \quad (3)$$

For multiple criticality levels, we need to iteratively perform the estimation for each section since the fault tolerance capability of different sections is different. We use the same equation 3 for approximating the packing failure probability. However, `size` and s varies between program sections. For example, if each weight of a particular layer is n bits and it can tolerate errors in upto $kbits$ from the LSB, then

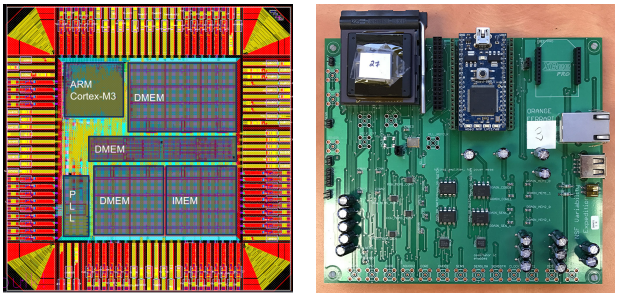
$$s = \left((1-b)^{(n-k)}\right)^{32/n} \quad (4)$$

This value of s is then substituted in equation 3. We start with the most critical weights and activations. For that layer, the size of the memory is taken as $size - size_{crit}$ bytes where $size_{crit}$ is the sum of the sizes of all program sections more critical than the current one being packed. After computing each layer, the size of the memory is reduced by the total size of weights and outputs of that layer since we will try to pack the next layer (in terms of criticality) after the previous layer has already been placed in the memory.

This analytical approach, combined with the sensitivity analysis results, can be used to estimate the achievable accuracy and packing yield at a particular VDD or BER and hence predict before deployment, fault tolerance and/or energy benefits of SAME-Infer for a specific hardware platform and neural network.

4 EXPERIMENTAL SETUP

We evaluate SAME-Infer on ten micro-controller class test chips. Each chip contains a single ARM Cortex-M3 core, 176 KB of on-chip data memory, 64 KB of instruction memory. They were fabricated in a 45nm SOI technology with dual-Vth libraries the chip floorplan and test board are shown in Figure 4. We characterized the voltage scaling-induced bitwise fault maps for these ten chips using detailed March-SS tests [19].



(a) Chip floorplan (b) Board

Figure 4: Test chip and board used to collect hard fault maps for SAME-Infer.

For most of our experiments, we used four 8-bit networks as given in Table 1. The first network is a minimally sized perceptron (MLP) with one hidden layer and is tested using MNIST dataset [6]. The second is a convolutional neural network (CNN) with two convolution layers and one fully-connected layer and is tested using Google Speech Command dataset [35]. The third and the fourth networks are bigger convolutional neural networks with three and six convolution layers and one and three fully-connected layers, respectively. Both these networks are tested using CIFAR-10 dataset [5]. All the layer weights and activation data of all networks are quantized to 8-bit precision. The first three networks are implemented using the ARM CMSIS-NN [24] library, version 5.6.0, optimized for Cortex-M processors and run on the test chip. The fourth network was too big to fit on our test chip. Hence, a fault injection framework was developed using PyTorch 0.4.1. In Table 1, 32CONV5-MP2 means 32 5x5 filters and 2x2 max pooling layer while 12FC/10FC means fully connected layer with 12/10 output neurons. The networks were trained using PyTorch 0.4.1.

Table 1: DL networks used in our experiments

Model (Precision)	Architecture	Dataset
MLP (8-bit, 1-bit)	784-128-10	MNIST
CNN-1 (8-bit)	32CONV5-MP2 32CONV5-MP2 12FC	SPEECH
CNN-2 (8-bit)	32CONV5-MP2 32CONV5-MP2 64CONV5-MP2 10FC	CIFAR-10
CNN-3 (8-bit)	32CONV3-32CONV3 MP2-64CONV3-64CONV3 MP2-128CONV3- 128CONV3 256FC-256FC-10FC	CIFAR-10

For all these networks we considered all weights and activation data to be non-critical (i.e., where sensitivity to errors would be calculated to assign them a criticality level) and all other parameters and instructions to be critical.

The toolchain, by default, packs the entire program code sequentially in the memory with no notion of faults. The supply voltage of each chip is reduced from the nominal 1V to 600mV in step size of 25mV. As the voltage is reduced, the hard fault rate in the memory increases. For each step, the binary is loaded and ran till completion to note the accuracy drop with decrease in voltage.

We then performed SAME-Infer for all 10 chips at every 25mV voltage step from 1V to 600mV and ran the customized binary on the chips to measure the final accuracy for all the voltage levels which had feasible packing solutions. The layer-wise precision results were obtained using a Theano [32] based framework. We used CPLEX [2] to solve the ILP packing problem in our experiments. All code is packed in the instruction memory while all data is packed in the data memory. If the section packing (solved iteratively for the multiple criticality levels) failed, we considered SAME-Infer not to have a feasible solution for that network on that particular chip at voltage equal and/or less than the current one. Since CNN-3 was too large for the test chip, the accuracy results were obtained using our PyTorch based fault injection framework. We created a series of randomly-generated *synthetic fault maps* for memory of size 1 MB. We synthesized 10 fault maps in 10 mV increments for a total of 10 *"synthetic test chips."* using detailed Monte Carlo simulation of SRAM bit-cell noise margins in the corresponding 45 nm technology. The fault maps were fed into our PyTorch framework and inference with faulty weights and activations were run to get the final accuracy.

We extended our analysis to binarized networks which can often be used for very resource-constrained embedded devices. As an example, we analyze binarized versions (dense and one with 85% sparsity) of the two layer MLP network. Since binarized networks, especially the sparse one, have the least amount of redundancy in them and are much smaller in size as compared to the 8-bit networks, they will be less resilient to faults, while at the same time, would be easier to fit in the memory. As binarized networks are already at the lowest precision level, error sensitivity is not calculated for these networks. Instead, while packing the weights, we try to maximize the intersection of 1's with stuck at one faults and the 0 values with stuck at zero faults.

5 RESULTS

As the supply voltage of each of the ten chips is reduced from the nominal 1V to 600mV in step size of 25mV, the hard fault rate starts increasing. The first faults start appearing around 800-850mV and the fault rate increases exponentially beyond 750mV.

5.1 Reduction in voltage with SAME-Infer

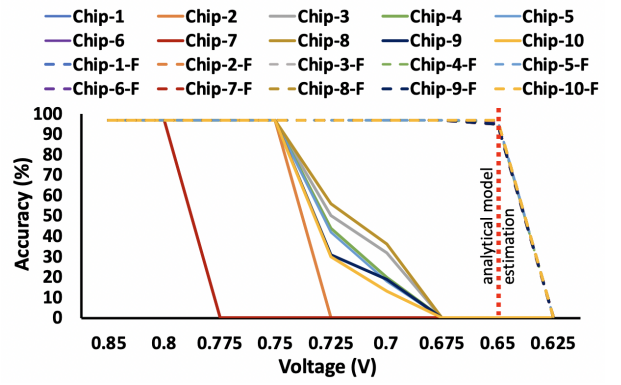
Using the default toolchain placement (without SAME-Infer), all three networks are ran for each voltage step on all ten chips and the results are shown by the solid lines in Figures 5a, 5b and 5c for MLP, CNN-1 and CNN-2, respectively. For the 2-layer MLP and the 2-layer CNN (CNN-1), network accuracy remains almost unchanged till above 750mV for 8 out of 10 chips and drops drastically at 725mV. This is because from 750mV to 725mV, the hard fault rate (bit error rate) increases by 2.7x. For the three-layer CNN (CNN-2), the network accuracy starts dropping at around 750mV. This is because of the larger size of the network resulting in a higher number of intersection with faults. Also, in CNNs, if a particular

weight or an input is erroneous, it affects multiple output values due to the high amount of reuse. Thus, the weights and activation data in CNNs have higher impact on the final accuracy as compared to MLP. Two, out of the ten test chips had intersecting faults with critical code sections at 775mV and hence, these chips start failing at higher voltage compared to the rest.

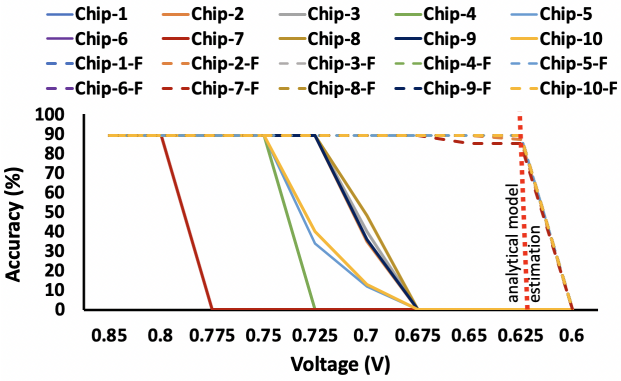
The results with SAME-Infer are shown in Figures 5a, 5b and 5c using dotted lines. The red vertical line shows the minimum voltage estimated by the analytical model that can be scaled down to while having minimal impact on accuracy. For the two layer MLP and the two layer CNN, there is minimal impact on accuracy above 650mV and 625mV respectively. At 600mV, for all the ten test chips, the critical section packing failed. Thus, with SAME-Infer, about 100mV-150mV voltage reduction was achieved for the two layer MLP and 125mV-175mV reduction was achieved for the two layer CNN network. The min-VDD estimated by the analytical model for the two layer CNN (CNN-1) is 620mV, but since we used step size of 25mV, we have not shown the exact result at 620mV. However, the model estimation falls within our obtained min-VDD range.

For the three layer CNN (CNN-2), the voltage could be scaled down to 675mV with no impact on accuracy (100mV lower than baseline). At 650mV, the non-critical section packing failed. From the sensitivity analysis results, the layer-2 weights and activation have the highest quantization noise gain or the highest minimum precision requirement. Therefore, for this layer, the number of tolerable faulty bit positions is 1 from the least significant bit (LSB) for weights and activation data. The weights in this layer also form the largest data section. As a result, packing the layer's weights in a contiguous memory segment with where each memory location can have at most one faulty LSB becomes infeasible at 650mV. The best case reduction achieved for this network was 125mV. This is similar to the minimum voltage estimated by the analytical model (670mV), thus, validating the model. Once again because we used step sizes of 25mV, we have not shown the results at 670mV, but we tested on three chips at 670mV and the network accuracy gets minimally (2%) affected at that voltage.

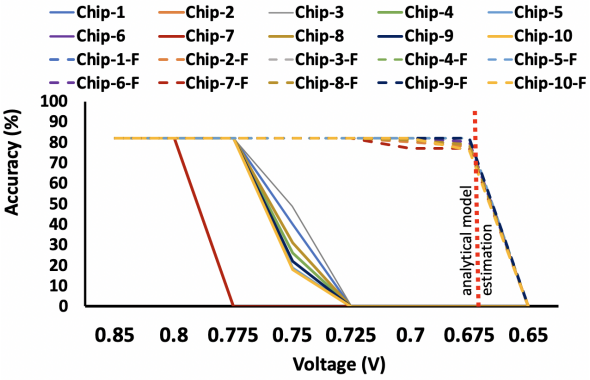
For all three networks, SAME-Infer achieved more than 100mV min-VDD reduction. Since memories consume significant fraction of the total system energy, 100mV-175mV reduction in min-VDD of the SRAM based scratchpad memories would lead to dramatic decrease in overall system energy consumption. Also, SAME-Infer can now tolerate upto 350x higher Bit Error Rate (BER). This is critical for tolerating aging induced or other in-field failures. If a built-in-self-test (BIST) engine can periodically upload fault maps to the cloud, SAME-Infer can be run remotely and the new failures can be avoided with a simple inexpensive software patch instead of an expensive faulty hardware replacement or in-field repair.



(a) MLP



(b) CNN-1



(c) CNN-2

Figure 5: Change in inference accuracy (8-bit networks) as voltage on the test chips is scaled down. Dotted lines are results with SAME-Infer while the solid lines are without SAME-Infer. The red dotted vertical line shows the analytical model estimation for the minimum voltage possible with no/minimal impact on accuracy.

5.2 Splitting up Weights to Achieve Better Packing

As seen in the three layer CNN, at 650mV, SAME-Infer fails to pack the largest and the most sensitive weight section. As mentioned in Section 3.3, one way to further reduce min-VDD and achieve better packing would be to split per layer's weight sections into smaller sections. The sensitivity analysis of the weights is extended to compute weight quantization noise gains on a per layer per filter basis. The results for the three layer CNN-2 are shown in Figure 6. An additional 50mV reduction in min-VDD was achieved for all 10 chips tested at negligible (<1%) code space overhead and no impact

on accuracy compared to SAME-Infer with layer-wise monolithic weight sections.

The weights can be further split up by granularity finer than a kernel. The best case is when each weight section is as small as a memory word. However, splitting up the data sections into such small sizes require non-negligible code modifications. Also, as can be seen in Figure 7, for the bit-error rate measured in our test chips and the three layer CNN network, it is seen that at the voltage where critical code sections fail, the smallest size of the non critical data section doesn't need to be smaller than the size of the kernel. Therefore, splitting on a per kernel basis is often good enough and results in least intrusive code changes.

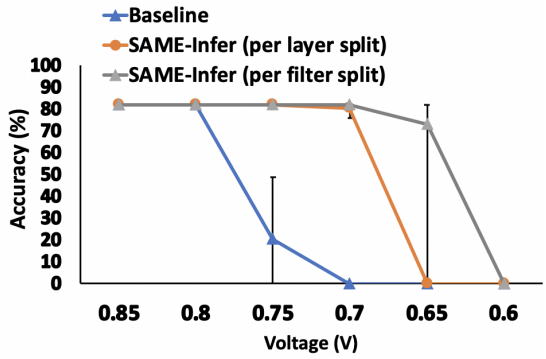


Figure 6: Change in three layer CNN-2 inference accuracy as voltage on the test chips is scaled down. The result shown here is the average accuracy across 10 test chips for each test case. The test cases are - (1) without SAME-Infer (2) with SAME-Infer and layerwise monolithic weight sections (3) when the weight sections are split up on per filter basis in every layer.

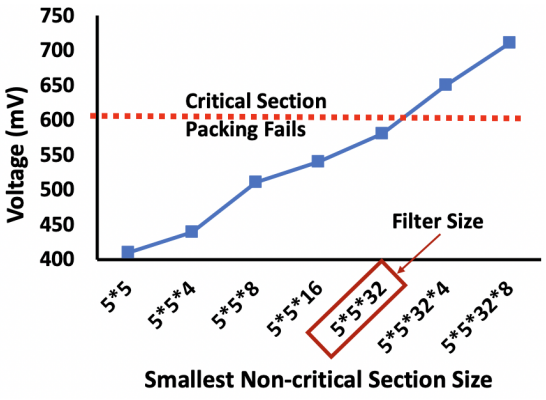


Figure 7: Achievable min-VDD as the smallest non-critical section size is reduced for the three layer CNN. The min-VDD is obtained using Equation 3 while the min-VDD for critical section is obtained from the test chip results.

We extended the analysis for a larger CNN (with 6 convolution layers and 3 fully connected layers). The network architecture is

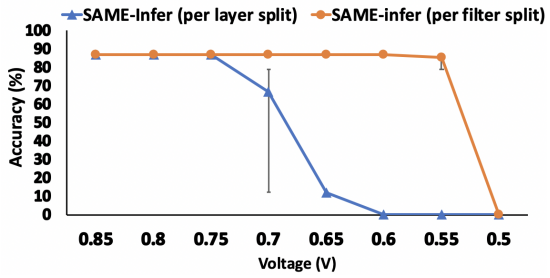


Figure 8: Change in nine layer CNN inference accuracy as voltage on the synthetic chips is scaled down. The result shown here is the average accuracy across 10 synthetic chips for each test case. The test cases are - (1) with SAME-Infer and layerwise monolithic weight sections (2) when the weight sections are split up on per filter basis in every layer.

$32C3 - 32C3 - MP2 - 64C3 - 64C3 - MP2 - 128C3 - 128C3 - 256FC - 256FC - 10$. Since the network was too large for the test chip, we created a series of randomly-generated *synthetic fault maps* for memory of size 1 MB. We synthesized 10 fault maps in 10 mV increments for a total of 10 *"synthetic test chips."* We used detailed Monte Carlo simulation of SRAM bit-cell noise margins in the corresponding 45 nm technology. The filter-wise precision results were obtained in Theano [32]. The accuracy results were obtained by running inference with faulty weights and activations using our PyTorch based fault injection framework and the synthetic fault map. The results are shown in Figure 8. With only SAME-Infer and monolithic layer weights, the desired packing could not be obtained for layers 5, 6 and 7 below 750mV. With split weight sections, in 8 out of 10 synthetic test chips, the min-VDD achieved was 550mV with desired precision, thus, having almost no impact on accuracy. The overall min VDD reduction with split weights was as much as 200mV compared to simple layerwise weight packing.

5.3 Importance of Sensitivity Analysis of Fault Tolerant sections

For comparison against naive placement strategy, we tried placing the non-critical weights and activation data sections at the first available memory region (greedy placement - first available unused memory segment) with no notion of sensitivity or bitwise intersection with faults at 700mV for the three layer CNN network on five test chips. Thus, instead of having 5 levels of criticality, we only had two levels. The first level is for the critical sections, and the second level is for all non-critical sections with no upper bounds on the number of faulty least significant bits. So, for the non-critical sections, even the most significant bit could have a fault.

The impact on accuracy was significant (shown in Figure 9) because a large number of weights and activation data were intersecting with faults in the most significant bits, thus, making the chips unusable at 700mV. With the intelligent placement of the fault tolerant sections we can run the network at 700mV with negligible impact on accuracy. Thus, doing a sensitivity analysis of the fault tolerant weights and activation data and placing these sections such

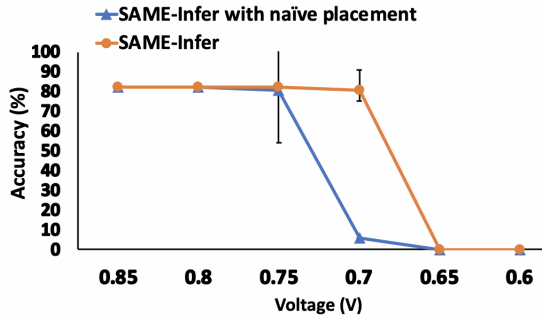


Figure 9: Change in the three layer CNN-2 inference accuracy as voltage on the test chips is scaled down. The result shown here is the average accuracy across 5 test chips for each test case. The test cases are - (1)when the fault tolerant sections are naively placed (greedy placement) in the memory while the critical text sections are placed in non-faulty memory regions (2) SAME-Infer with criticality aware placement.

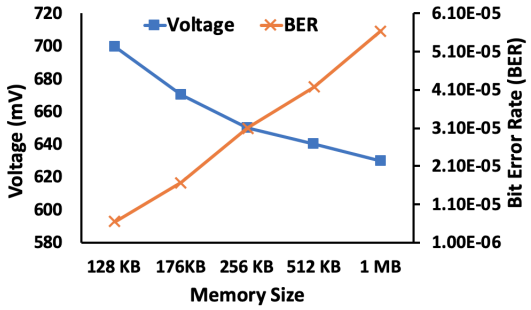


Figure 10: Voltage reduction or BER tolerance estimation by the analytical model for the three layer CNN-2 on different memory sizes.

that only the tolerable bits intersect with faults result in more than 50mV reduction in voltage.

5.4 Analytical Model to Estimate for Larger Sized Memories

Using the analytical model we estimated the minimum voltage (maximum BER) that the three layer CNN-2 network can tolerate if the size of the memory is increased. In most cases, the target inference systems are of standard sizes while the network sizes vary greatly. The results are shown in Figure 10. It can be seen that for a memory size of 512KB (instead of the 176KB in our test chip), the voltage can be scaled down to 640mV (from 670mV with 176KB) and a 2.5x higher BER can be tolerated. Once again, we used detailed Monte Carlo simulation of SRAM bit-cell noise margins in the corresponding 45 nm technology to calculate the bit error rate.

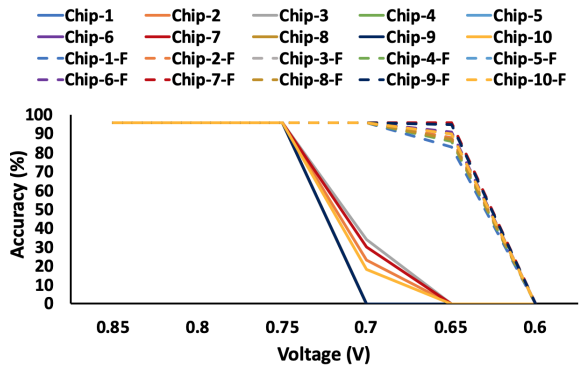


Figure 11: Change in dense binarized MLP inference accuracy as voltage on the test chips is scaled down.

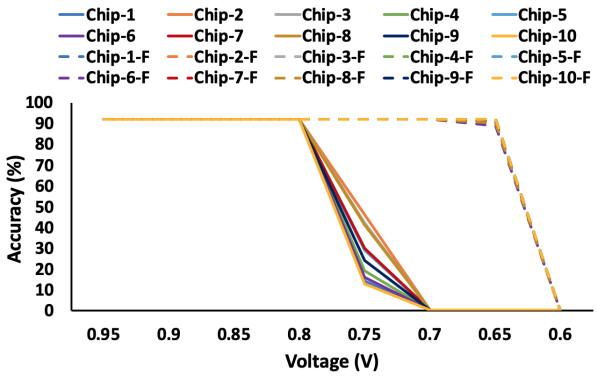


Figure 12: Change in sparse binarized MLP inference accuracy as voltage on the test chips is scaled down.

5.5 Evaluation for Binarized Dense and Sparse Networks

To evaluate the impact of SAME-Infer on networks that are expected to be less fault tolerant (quantized and/or sparse networks), we extended our analysis to binarized dense and sparse versions of the two layer perceptron network, tested using MNIST dataset (results shown in Figures 11 and 12). For the 8-bit version of the same network, at 750mV, the network accuracy almost remains unaffected for all 10 chips. The same is true for the dense binarized version. However, in the binarized sparse MLP network, we start seeing an impact on accuracy at 750mV. This is because most of the redundant weights have been removed from the network and only the critical weights are used. Therefore, any intersection with faults results in an impact on accuracy, causing the network to have very low tolerance to faults. However, all three versions of the network can be scaled down to 650mV with SAME-Infer. The non-critical sections in the sparse network are the smallest in size and hence, can be packed perfectly even at 650mV.

5.6 Comparison with Past Works

5.6.1 Treating all data sections as critical. In [14], the authors propose a software assisted methodology to place program sections

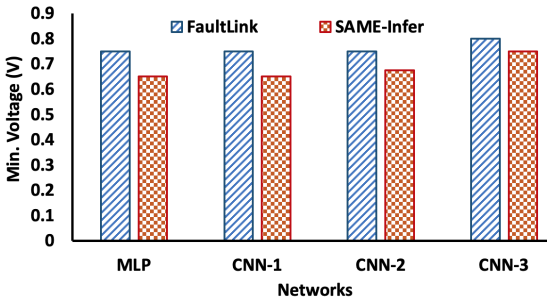


Figure 13: SAME-Infer achieves lower min voltage as compared to FaultLink [14] with negligible impact on accuracy because SAME-Infer allows intersection with faults in the less critical LSB bits of non-critical fault tolerant data sections.

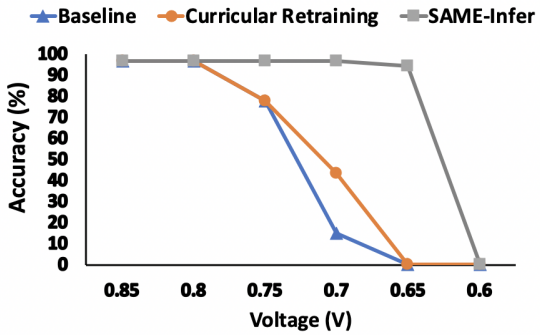


Figure 14: MLP (2 layer) with MNIST - average accuracy measured across 10 chips for each test case - (1) Baseline (2) Curricular Retraining (3) SAME-Infer

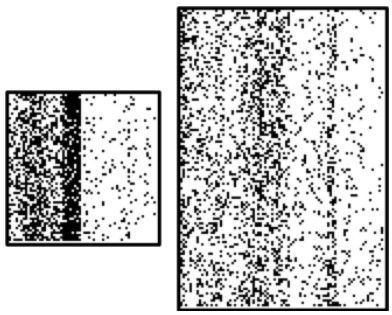


Figure 15: Hard Fault Map of the 64KB instruction memory (left) and the 176KB data memory (right) of a test chip. The black dots represent the faulty byte locations.

in non-faulty memory segments. However, they treat all program sections (data and instructions) as critical and try to fit them in fault-free segments of the memory. Doing so has two primary disadvantages. Firstly, it fails to exploit the inherent redundancy in Deep Learning Inference (or any other Approximation Tolerant) applications. As a result, the packing solution would fail at a much lower fault rate (higher voltage) than what it can actually tolerate. Secondly, since this solution does not allow any intersection with faults, the actual size of the memory needs to be much higher than the size of the binary to be able to successfully pack all sections in fault free memory segments at low voltages. We compared [14] with SAME-Infer and the results are in Figure 13. SAME-Infer allows scaling by more than 100mV (average) for most applications. The 100mV voltage scaling translates to >25x higher fault tolerance as well. Thus, SAME-Infer delivers much higher energy reduction/fault tolerance as compared to [14]. Also, for the same memory size, SAME-Infer will be able to fit a larger sized network than FaultLink when running at the same voltage. This is critical as network sizes that are being deployed on these edge devices is increasing rapidly.

5.6.2 Fault Injection During Training as an Alternative to SAME-Infer. In order to boost DNN's error tolerance, [23] proposed a curricular retraining approach. This mechanism injects errors into the DNN training procedure to boost the error tolerance of the network. A similar fault-aware training has also been proposed in [22]. We assumed a random uniform distribution of bit errors and measured the average bit error rate (BER) at every voltage across our 10 test chips. This is because training on the target faulty edge devices is impractical and hence, exact fault maps cannot be used while retraining. The authors also make a similar assumption in [23]. The results are shown in Figure 14. The baseline here refers to running the original trained network (without curricular retraining) on the chip at reduced voltage. The baseline min-VDD is compared against the min-VDD obtained with curricular retraining and with SAME-Infer. For the 10 test chips tested, curricular retraining helps to lower the voltage by at most 50mV in only 5 chips (a few of them see a non-negligible impact on accuracy), whereas, SAME-Infer allows voltage reduction in all of them.

From the results it can be seen that curricular retraining using uniform random bit error distribution provides negligible improvement in this case. This is because, at lower voltage, faults in SRAMs tend to be correlated (shown in Figure 15). Hence, unless it is retrained on the target platform, the retraining mechanism would provide very limited improvement. But retraining on target platform is likely impractical (e.g., training may have been done with proprietary datasets on high performance GPUs). Most edge platforms lack the computational power required for training these networks [37]. Large networks require tens of exaFLOPS of compute across the entire training cycle [10], making them infeasible to be run on edge devices. Moreover, faults appear in code memory as well. Therefore, only making network data (weights and activations) more resilient to faults is insufficient. In the next section, we discuss how error-injection based training may be helpful.

Overall we see that SAME-Infer not only allows energy saving through voltage scaling, it is also an efficient fault tolerance technique as it tolerates 25x average increase in byte error rate with minimal impact on inference accuracy. For some chips it tolerates upto 350x increase in BER with minimal to no impact on network accuracy. Since edge devices may have long lifetimes, aging becomes a concern for the reliability of the device. SAME-Infer can be used as an in-field repair technique where fault maps are periodically sampled using BIST and uploaded to cloud. SAME-Infer is

then run remotely for aging induced faults and the updated binary is deployed during software updates with minimal disruption to the customers. SAME-Infer also helps to increase the increase the yield of chips by allowing usage of faulty chips leading to significant cost savings.

6 DISCUSSION

In this section, we briefly discuss some of the possible extensions to SAME-Infer, which though not explored thoroughly in our current set of experimental results, can provide additional fault tolerance and/or power benefits as well as easier deployment.

6.1 Fault Injection During Training to Tolerate Soft Errors

As we saw in Section 5.6.2, curricular retraining by injecting bit errors while training does not provide much benefit against correlated hard faults at lower voltage when used on its own. However, curricular retraining or random fault injection during training can help with tackling soft errors (random memory bit flips during runtime). At lower voltage, susceptibility to radiation-induced soft faults increases because critical charge, which is the charge threshold to cause a soft error, decreases [12, 31]. Curricular retraining can be used to augment SAME-Infer to tolerate this increased soft error rate at lower voltage.

6.2 Improving Packing by Optional Reversing of Non-Critical Sections

SAME-Infer is able to tolerate faults in the least significant bit regions of the memory but largely leaves the most significant bit regions untouched. As a result, faults in roughly half the memory remain unaddressed by SAME-Infer. An interesting way to extend SAME-Infer is to optionally reverse the weights/activations (i.e., essentially reverse the order of bits in the byte).

In our framework, we try to pack our non-critical weight/activation sections in contiguous memory segments with no faults in the desired most significant bits. If the number of error tolerant least significant bits for a particular weight is 2, every address in the memory segment used to pack this section needs to have fault-free 5 most significant bits. Now if the weights have the option of being reversed before being stored in the memory, the packing algorithm has the option of storing the weights in either a memory segment where every address has fault free 5 MSBs or in a memory segment where every address has fault free 5 LSBs. If the latter is chosen, the weights need to be reversed. This doesn't require a pre-compilation modification to the source code for every chip. The way to do this is to have custom load and store procedures for weights. When storing or loading weights, a particular address location is checked. If the value stored there is 1, then the weights are not reversed, if 0, then the weights need to be read from or written to in a reversed fashion. The value to be stored in that particular address can be done during link time based on the packing solution. Therefore, this is a one-time source code modification, every-time link solution (like the rest of our methodology). The probability of there not being a memory segments that is large enough to store the largest program section decreases and the updated Equation 3 will be:

$$P\left(L_{\texttt{size}/4} < \frac{m_{\max}}{4}\right) \approx e^{-2s^{\left(\frac{m_{\max}}{4} - \log_s{-1}\left(\frac{\texttt{size}}{4}(1-s)\right)\right)}}. \quad (5)$$

We evaluated this for a limited number of synthetic test chips with the nine layer CNN. For two out of ten chips, it helped to reduce the min-VDD by 50mV as compared to the baseline SAME-Infer (with no weight splitting). The obvious drawback of this approach is the additional runtime and code size overhead of checking and reversing. The code size overhead is small though the runtime overhead can be noticeable since every load operation now translates to branch, load and rotate operations. As a result, we did not explore this option further. However, this approach is useful in very high fault rate or very low power scenarios.

6.3 Universal Packing Solution to Allow Dynamic Voltage Scaling and Tolerate Aging Induced Faults

Hard faults in SRAMs due to voltage scaling are inclusive, that is, faults that appear at a higher voltage remain as the voltage is lowered [15]. The fault map at 600mV would include all the faults that appeared at voltages higher than 600mV (along with some additional faulty bits). Therefore, if the SAME-Infer packing is done for the lowest possible VDD, the same packing solution can be used when running the chip at higher voltages. This allows having a universal packing solution for a given chip. The application is loaded into the memory based on this solution during deployment and the voltage can be dynamically scaled during runtime without having to repack every time.

We tested the solution out on three test chips where the packing solution at 650mV was used when running at 700mV and the accuracy was the same as what was achieved when specifically packed using the memory fault map and packing solution of 700mV.

Memory chips go through multiple rounds of burn-in [4] testing which involves a series of full chip read and write operations. We suggest having something similar for the memories in embedded chips. During burn-in, a March test equivalent can be run. As embedded memories are much smaller than standard DRAM chips, burn-in testing overhead should be small. Based on the stored fault map, the application can be packed for the lowest possible supply voltage. Having this not only saves the effort of repacking every time the voltage is scaled during runtime, it also provides protection against in-field aging induced failures. This is because the weaker cells are expected to fail and get captured in the low voltage fault map and the universal packing solution would take care of it.

6.4 Addressing the Code Memory Bottleneck

In several of our benchmark/chip combinations (especially for smaller networks), SAME-Infer packing failure is due to code (which is all considered critical) that is unable to get packed in code memory. There are two possible ways to address this. First, microcontroller designs can allow for separate power delivery network for code memory (so that data memory can be independently voltage scaled). Second, more intrusive changes to the machine learning code can be made to build it from smaller functions. This would have a negligibly small impact on code size and runtime but will

result in more packable code in presence of faults in code memory (as every function can be mapped to different memory segment).

6.5 Use of Error Correcting Codes (ECC)

ECC is a common approach for error detection and correction in memories. However, they are better suited for random, temporary faults and incur area, performance and energy overheads. If a t-bit error correcting code is used, i.e., errors upto t-bits can be detected and corrected by the code, all k-bit messages get encoded into (k+r)-bit codewords before they are stored in the memory. The extra r-bits of parity are added onto the original message to enable error correction. As the code becomes stronger or the requirement for t increases, the number of parity bits (r) also increases. During a read operation, the encoded message is loaded from the memory and decoded such that the original k-bit message can be recovered and errors upto t-bit can be detected and corrected. The additional parity bit storage as well as the encoding and decoding overheads are non-negligible and increase rapidly as the correction capability of the code increases. As a result, they can be an overkill (and therefore a bad approach) for permanent faults.

Our experiments revealed that the hard faults at lower voltages normally tend to be correlated. As a result, multi-bit error correction would be required. While SAME-Infer can tolerate up to 4-bit faults on 8-bit weights/activations with negligible performance overheads, an double-bit error correcting (DEC) code would require 8 additional bits of parity and have 2 cycles of encoding and decoding latency. For example, the 9-layer CNN-3 network has a total size of ~700KB. To fit this network at nominal voltage with no faults, at least 1.4MB of memory is needed if DEC code is used for protection. Moreover, around 650mV-670mV, triple bit errors start appearing and thus, lowering the voltage further will lead to loss in accuracy as the un-correctable errors can coincide with MSBs. On the other hand, with SAME-Infer, we managed to fit in the entire network within a 1MB memory and could lower our voltage to less than 600mV with no impact on accuracy (Figure 8). However, if ECC is available, it should augment SAME-Infer to address soft errors during runtime.

6.6 Extending SAME-Infer to Other Approximation Tolerant Applications

The SAME-Infer framework can be easily extended to other approximation tolerant applications. There are several applications in the field of approximate computing that can tolerate controlled relaxation of correctness for improving performance or energy efficiency. For such workloads, already existing frameworks (such as Approxilyzer [33]) can be used to identify the non-critical approximation friendly sections of the code that can intersect with faults without impacting output quality. Once the one-time analysis of the code is done, SAME-Infer can be used to correctly map the critical and non-critical sections in non-faulty and faulty memory segments respectively, leading to lower energy or higher fault tolerance.

7 CONCLUSION

Design of edge devices is driven by the need for the lowest possible cost and energy consumption, which are both strongly affected by on-chip memories. Further, many of these may be deployed in harsh environments where in-field replacement is difficult due to faults. The proposed SAME-Infer methodology addresses both these issues for embedded scratchpad memories running machine learning applications. SAME-Infer uses the linker to map the critical code/layers onto the non-faulty segments of the memory while the fault-tolerant sections of data are placed in faulty memory segments. This allows SAME-Infer to tolerate upto 350x (average 25x) higher bit error rate without degrading inference accuracy. Our evaluations on 10 real micro-controller class chips and 10 larger synthetic chips show that up-to 175mV reduction in voltage can be achieved without any loss in accuracy for a fully connected network and for two convolutional neural networks. Thus, SAME-Infer helps to tolerate higher hard fault rate by exploiting the redundancies in the DL applications and helps in cost savings by making error prone memory chips usable.

ACKNOWLEDGMENTS

The authors thank the MEMSYS'20 program committee and the anonymous reviewers for their detailed and constructive feedback. The authors thank Tianmu Li from UCLA for helpful discussions and his help with setting up the framework for training the Deep Learning networks.

REFERENCES

[1] [n.d.]. 1-Transistor SRAM Cell Scales to FinFET Technology Node. https://www.eeweb.com/profile/max-maxfield/articles/1-transistor-sram-cell-scales-to-finfet-technology-node/.
[2] [n.d.]. CPLEX Optimizer. https://www.ibm.com/analytics/cplex-optimizer.
[3] [n.d.]. Multiple Knapsack Problem. http://www.or.deis.unibo.it/kp/Chapter6.pdf.
[4] [n.d.]. Stringent Tests from ICs to Modules Ensure DRAM Reliability. https://www.atpinc.com/blog/dram-testing-module-chips-ic-burn-in-quality-characteristics.
[5] [n.d.]. THE CIFAR-10 DATASET. https://www.cs.toronto.edu/~kriz/cifar.html.
[6] [n.d.]. THE MNIST DATABASE. http://yann.lecun.com/exdb/mnist/.
[7] 1995. Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification (Version 1.2).
[8] M. Ahmad. 2014. Reliability Models for the Internet of Things: A Paradigm Shift. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. 52–59. https://doi.org/10.1109/ISSREW.2014.107
[9] Alaa R. Alameldeen, Ilya Wagner, Zeshan Chishti, Wei Wu, Chris Wilkerson, and Shih-Lien Lu. 2011. Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*.
[10] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR* abs/1512.02595 (2015). arXiv:1512.02595 http://arxiv.org/abs/1512.02595
[11] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. 2002. Scratchpad Memory: A Design Alternative for Cache On-Chip Memory in Embedded Systems. In *Proceedings of the ACM/IEEE International Symposium on Hardware/Software Codesign (CODES)*.
[12] V. Degalahal, Lin Li, V. Narayanan, M. Kandemir, and M. J. Irwin. 2005. Soft errors issues in low-power caches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, 10 (2005), 1157–1166.
[13] Nikil Dutt, Puneet Gupta, Alex Nicolau, Abbas BanaiyanMofrad, Mark Gottscho, and Majid Shoushtari. 2014. Multi-Layer Memory Resiliency. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*.
[14] Mark Gottscho, Irina Alam, Clayton Schoeny, Lara Dolecek, and Puneet Gupta. 2017. Low-Cost Memory Fault Tolerance for IoT Devices. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 128 (Sept. 2017), 25 pages. https://doi.org/10.1145/3126534
[15] Mark Gottscho, Abbas BanaiyanMofrad, Nikil Dutt, Alex Nicolau, and Puneet Gupta. 2015. DPCS: Dynamic Power/Capacity Scaling for SRAM Caches in the Nanoscale Era. *ACM Transactions on Architecture and Code Optimization (TACO)* 12, 3 (2015), 26.
[16] Puneet Gupta, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Rajesh K. Gupta, Rakesh Kumar, Subhasish Mitra, Alexandru Nicolau, Tajana Simunic Rosing, Mani B. Srivastava, Steven Swanson, and Dennis Sylvester. 2013. Underdesigned and Opportunistic Computing in Presence of Hardware Variability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32, 1 (2013), 8–23.
[17] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. *CoRR* abs/1502.02551 (2015). arXiv:1502.02551 http://arxiv.org/abs/1502.02551
[18] Said Hamdioui, Georgi Gaydadjiev, and Ad J. van de Goor. 2004. The State-of-art and Future Trends in Testing Embedded Memories. In *International Workshop on Memory Technology, Design and Testing (MTDT)*.

[19] Said Hamdioui, Ad J. van de Goor, and Mike Rodgers. 2002. March SS: A Test for All Static Simple RAM Faults. In *International Workshop on Memory Technology, Design, and Testing (MTDT)*.

[20] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *CoRR* abs/1506.02626 (2015). arXiv:1506.02626 http://arxiv.org/abs/1506.02626

[21] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783722

[22] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe. 2018. MATIC: Learning around errors for efficient low-voltage neural network accelerators. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–6.

[23] Skanda Koppula, Lois Orosa, A. Giray Yağlçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. 2019. EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. ACM, New York, NY, USA, 166–181. https://doi.org/10.1145/3352460.3358280

[24] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *CoRR* abs/1801.06601 (2018). arXiv:1801.06601 http://arxiv.org/abs/1801.06601

[25] Serge Lamikhov-Center. 2016. ELFIO: C++ Library for Reading and Generating ELF Files. http://elfio.sourceforge.net/

[26] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. ACM, New York, NY, USA, Article 8, 12 pages. https://doi.org/10.1145/3126908.3126964

[27] Preeti Ranjan Panda, Nikil Dutt, and Alexandru Nicolau. 1999. *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration.*

[28] C. Sakr and N. Shanbhag. 2018. An Analytical Method to Determine Minimum Per-Layer Precision of Deep Neural Networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1090–1094. https://doi.org/10.1109/ICASSP.2018.8461702

[29] Mark F. Schilling. 2012. The Surprising Predictability of Long Runs. *Mathematics Magazine* 85, 2 (2012), 141–149.

[30] Stanley E. Schuster. 1978. Multiple Word/Bit Line Redundancy for Semiconductor Memories. *IEEE Journal of Solid-State Circuits (JSSC)* 13, 5 (1978), 698–703.

[31] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz. 2006. Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices. In *2006 IEEE International Reliability Physics Symposium Proceedings*. 217–225.

[32] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (2016). http://arxiv.org/abs/1605.02688

[33] R. Venkatagiri, A. Mahmoud, S. K. S. Hari, and S. V. Adve. 2016. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14.

[34] Lucas Wanner, Charwak Apte, Rahul Balani, Puneet Gupta, and Mani Srivastava. 2013. Hardware Variability-Aware Duty Cycling for Embedded Sensors. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)* 21, 6 (2013), 1000–1012.

[35] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

[36] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. 2008. Trading off Cache Capacity for Reliability to Enable Low Voltage Operation. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

[37] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang. 2019. Machine Learning at Facebook: Understanding Inference at the Edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 331–344.