EDEN: Enabling Approximate DRAM for Efficient Neural Inference with Error-Resilient Neural Networks

ABSTRACT

The effectiveness of deep neural networks in vision, speech, and language tasks has prompted a growth in systems specifically designed to accelerate network inference workloads. A well-known energy and performance bottleneck of these accelerators is off-chip memory access, which can dominate the system's power and stall time.

In this work, we aim to improve the energy efficiency and latency of neural network inference through the use of approximate DRAM. We exploit two key observations: (1) modern neural networks exhibit an intrinsic capacity to absorb input and weight bit errors and (2) violation of DRAM standards allows greater performance at the cost of lower bit reliability. This work explores the use of aggressive DRAM voltage-scaling and read/write timing parameter scaling as two conduits for improved timing and energy efficiency.

We introduce EDEN, a cross-platform system that optimizes memory power and performance within a given accuracy budget for the neural network. EDEN characterizes and boosts the error resiliency of the neural network using empirically-derived error models. It schedules inference execution on a refined DRAM design that permits fine-grained adjustment of read/write timing and supply voltage. EDEN improves on prior works by performing extensive characterization of network accuracy using different error models. Using this characterization, we demonstrate increased neural network resiliency to permit execution on approximate DRAM.

We evaluate EDEN on multi-core CPU, GPU, and cycle-accurate CNN accelerator architectures. We find that EDEN enables a DRAM power reduction of >29% across all three platforms, and total cycle reductions of up to 15% when evaluating memory latency-bottlenecked neural networks, while maintaining a model accuracy within 2% of the original.

1. INTRODUCTION

Deep neural networks (DNNs [1]) have become a mainstay solution to challenges in computer vision, speech recognition, language translation, and a number of other domains [2, 3, 4, 5, 6, 7]. DNNs and their various flavors (convolutional neural networks (CNNs), fully-connected networks, and recurrent networks) are now commonly used in data centers and consumer devices that demand high performance and low-power [8, 9]. Unfortunately, these performance demands are often at odds with the high computational and memory demands of neural network evaluation. As such, neural networks have been the subject of much interest because of their small core set of operations that makes these models suitable for new architectures and efficient acceleration. Significant recent work (Eyeriss, Cnvlutin, EIE, Cambricon-X, and others) has focused on building specialized spatial-array based architectures for efficient computation scheduling and on-chip dataflow [10, 11, 12, 13, 14, 15, 16].

One of the core challenges in efficient computation of neural networks is off-chip memory performance. Memory performance contributes significantly to both the energy consumption and throughput of optimized inference accelerators and many CPU setups [17]. Prior work on inference accelerators report that between 30 to 50% of system power is consumed by DRAM [10, 18]. Other work has shown that reducing DRAM usage via compression can yield a >120x energy improvement in inference accelerators [13]. For modern systems, a common rule of thumb based on empirical data is that single off-chip memory accesses to DDR4 DRAM consume up to 100-1000x more energy than a single floating point addition on 45nm CMOS; similarly, a load or store that misses through the cache hierarchy takes up to 100x longer time to service compared to an L1 hit [19, 20]. On the neural network side, memory demands are growing. The winning model of the 2017 ILSVRC image recognition challenge, ResNeXt, contains 837M FP32 parameters; more recent models have broken the one billion FP32 parameter mark many times over [21, 22]. As the model development community trends towards larger, more expressive neural networks and as systems designers seek higher levels of parallelism for network inference, we expect the off-chip memory problem to bottleneck neural network performance without memory-cognizant systems design.

The growing memory bottleneck has motivated recent explorations on ways to reduce the memory intensity of neural network workloads or improve the underlying memory technology. In particular, this includes (1) algorithmic approaches that choose to reduce memory usage (e.g. by reducing numeric bitwidth, re-using model parameters, increasing sparsity) [23, 24, 25, 26, 27, 28] or (2) change the underlying memory device to enable either use of processing-in-memory or emerging memory technologies [9, 29, 30, 31, 32, 33, 34].

To achieve improved energy efficiency and reduced inference latency, we propose improving the use of existing DRAM technology to better suit the intrinsic characteristics of neural networks. This work leverages two key insights:

1. Neural networks demonstrate remarkable generalizability. This allows highly accurate recognition in spite of

random errors in the neural network input (e.g., random perturbations in pixel space).

2. Prior work analyzing real DRAM devices show that manufacturers trade performance for complete storage reliability [35, 36, 37]. Reducing main memory supply voltage and latencies can improve the power consumption and timing, respectively, but at the cost of bit reliability.

This paper introduces EDEN: a cross-platform framework that exploits these two insights to achieve improved energy efficiency and reduced latency operation for neural network inference by using unreliable memory. EDEN proposes supply voltage (V_{DD}) scaling as a practical mechanism for reducing power consumption, and DRAM read latency reduction as an option for reducing execution time when the neural network inference is bounded by memory latency.

EDEN performs three key steps: (1) it executes a profiling stage to obtain accurate, device specific, and empirically informed error models of the target DRAM device, (2) it performs an automatic characterization of the target neural network using these error models, in order to understand the network's error tolerance and improve its bit error resilience, and (3) it generates mappings between the available DRAM partitions, network components, and DRAM device settings (supply voltage and read latencies). After these three stages, users of EDEN are able to run neural network inference on unreliable DRAM using EDEN's recommended main memory and neural network settings.

EDEN's network characterization step is guided by extensive empirical analysis that we perform on the effect of neural network pruning, numeric bit-width, network type, and memory error model on final model accuracy and error resilience. This analysis underlines the importance of having an empirically-driven memory error model when improving error tolerance in neural networks, an observation we find missing in prior works. This analysis also provides us with an estimate of the tolerable bit error rates (BERs) for state-of-art computer vision networks.

Finally, we supplement EDEN with proposed changes to the DRAM memory controller that would allow EDEN to operate with even higher efficiency. In particular, we propose row-group adjustment of timing parameters, approximate fault mitigation, and chip-level adjustment of voltage in the DRAM memory controller.

The potential of EDEN stems from its capacity to be run on many of the hardware platforms in use today for neural network inference. Almost all processors (e.g., an embedded CPU or a TPU-based ASIC) rely on DRAM. We find that the principles of EDEN can be applied across different processing models and memory technologies.

This paper makes the following contributions:

- We provide the first systematic study of memory-error sensitivity of state-of-art CNN architectures [38, 28, 27] used in computer vision. We demonstrate the importance of an accurate memory error model, and study error resiliency across different numeric precisions, pruning levels, and network components.
- With the correct error model, we find that we can improve the tolerable bit error resiliency by one to two orders of magnitude, depending on the network, through EDEN's iterative re-training procedure.
- We introduce EDEN, a generalizable framework to run neural inference with higher energy efficiency and lower latency through the use of approximate DRAM. In particular, EDEN provides a systematic way to scale main memory parameters such as supply voltage and read latencies, while still achieving neural network accuracy targets.
- EDEN demonstrates a network characterization procedure, that given a target neural network accuracy, learns the tolerable error rates of each individual neural network kernel under a specific memory device error model. This information is used by a scheduler that maps network kernels and intermediate features to memory partitions/memory parameters based on each kernel's maximum tolerable error rates.
- We evaluate EDEN on top of multi-core CPU, GPU, and cycle-accurate CNN accelerator simulation platforms. EDEN provides execution time reductions of up to 15% on latency-bound neural networks and power reductions of >29% across all platforms on 8 neural network inference benchmarks, while maintaining a model accuracy within 2% of original.

The code for EDEN, simulation configuration, and final experiment logs can be found at `<link to be provided after review to preserve double-blind anonymity>`.

2. PRELIMINARIES

2.1 Deep Neural Networks

Deep neural networks (DNNs) are witnessing wide adoption, and are used across many domains, including vision, language, gaming, and medicine [39, 5, 4, 8]. The various flavors of DNNs (convolutional network [CNNs], fully connected networks [FCNs], etc.) have demonstrated state-of-art performance on many common image and speech tasks, with sizable improvements over competing approaches. Over the past decade, both the network architectures and datasets have grown larger, and correspondingly, accuracy on various standard image classification and speech recognition benchmarks has improved [2].

There are three classes of neural network components that require loads and stores from main memory: (1) the initial inputs (e.g. an image), (2) the intermediate activations resulting from each affine+nonlinear transformation of each intermediate layer (also called the 'intermediate feature maps' [IFMs]), and (3) the model parameters. In this work, we explore the introduction of bit errors to all three components.

Neural networks used in production systems today have on the upwards of 100 layers and 10-100+ million parameters [40, 41]. More recent work has demonstrated state-of-art results with models with the billions of FP32 parameters [22].

The effectiveness of neural networks stems, in part, from this over-parameterization [42]. Over-parameterization allows sufficient model learning capacity so that the network

Model	Dataset	Model Size	IFM Size
ResNet101	CIFAR10	163MB	100MB
MobileNetV2	CIFAR10	22.7MB	68.5MB
VGG-16	ILSVRC2012	528MB	218MB
DenseNet201	ILSVRC2012	76MB	439MB
SqueezeNet1.1	ILSVRC2012	4.8MB	53.8MB

Table 1: Models used in the evaluation of EDEN. The listed total model and summed IFM sizes are for the nominal FP32 weights, and are less in our low bit-width models. Estimated IFM sizes are calculated for batch size 1.

can approximate any complex input-output functions, and adequately capture high-level semantics (e.g. the characteristics of a cat) [43]. Importantly, this over-parameterization also allows the network to generalize across different inputs, and be robust to insignificant changes to the input (e.g. background pixels behind the cat) [44].

This over-parameterization also lends itself well to redundancy: common training-time techniques such as *activation dropout* and *adding white noise* to IFMs try to force the network to not rely on any single hidden unit and be more robust to input variance [45]. This is known as model regularization. This work shows that we can use device-level characteristics to achieve the same regularizing effect.

This work is also motivated by prior work that demonstrates the accuracy retention achieved by neural networks that have their parameters modified through *pruning* and *stochastic quantization* [46, 47, 48].

In this work, we test the hypothesis whether this intrinsic generalizability implies that neural network inference would be robust to errors introduced at the device level.

We study the performance of five networks listed in Table 1. Our models are targeted towards small and large-scale image classification, and are modern, commonly-used base architectures [38, 40, 41]. The first, third, and fourth architectures were prior top-five winners of the ImageNet ILSVRC competition [49, 50]. We use the Google MobileNetV2 architecture in our study to test smaller, mobile-optimized networks that are widely used on mobile platforms [27]. SqueezeNet is another commonly-used architecture used in embedded, real-time applications [28]. For reference, we list the summed bytesizes of all IFMs and model weights for each network; this is the amount of data shuffling in and out of the memory hierarchy when the model processes one input.

2.2 DRAM Structure and Operation

Figure 1 describes the high-level architecture of modern DRAM systems. The processor is connected to DRAM via a bus that is used to send data and store and load commands. A *DRAM module* (also called a *dual in-line memory module* [DIMM]) consists of roughly eight DRAM chips. Each chip consists of a set of parallel *banks* and peripheral control circuits. Each bank is an array of bit storage capacitors with a cross-bar: bit lines for indexing into the correct row and word lines to access and read the capacitor values. We refer the readers to [51] for more details on DRAM structure and design.

Accessing data stored in each bank row follows the se-

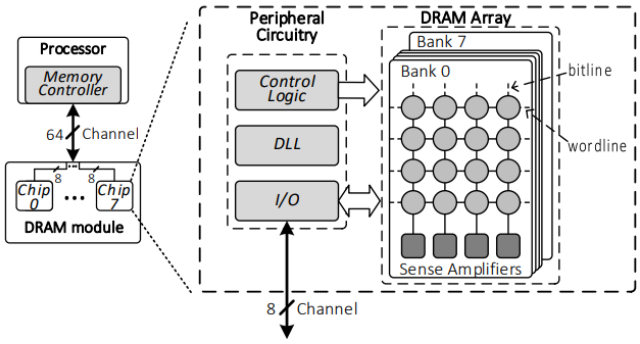

Figure 1: DRAM Structure [35]

quence of memory controller commands detailed in Figure 2. In particular, reading a cache line from DRAM requires: (1) activating the row by pulling up the bitline and waiting t_{RCD} nanoseconds for the charge to latch into a *row buffer*, (2) reading or pulling the data from the row buffer with a read/write command, and after $t_{RAS} - t_{RCD}$ nanoseconds, (3) preparing to re-start the cycle with a pre-charge command that resets the wordlines to $V_{dd}/2$. The activate of the next load/store is allowed to occur after r_{RP} nanoseconds. The nominal values of t_{RCD}, t_{RP}, and t_{RAS} are 13ns, 35ns, and 13ns, respectively.

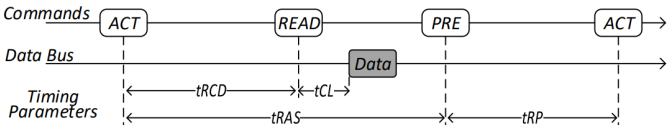

Figure 2: DRAM Read Timing [35]. We explore reductions of t_{RCD}, t_{RP}, and t_{RAS}. t_{CL} is a characteristic of the device, and not adjustable in the memory controller.

This work builds on the large body of work characterizing DRAM behavior in sub-reliable operation regimes: refresh frequency, V_{dd} scaling, and read/write latency scaling (e.g. $t_{RCD}/t_{RP}/t_{RAS}$) have all been identified as sources of DRAM optimization. Prior work has explored eDRAM refresh reduction for approximate computer vision. In this work, we generalize this to a variety of other DRAM configurations: voltage scaling, read latency scaling, and write latency scaling. This paper demonstrates that we can achieve reduced inference latencies and lower power consumption by including a more complete picture of DRAM optimizations.

DRAM power consumption is split between dynamic and static power. Dynamic power consumption is driven by execution of read, write, precharge, and activate commands. Static power consumption is largely the result of access transistor leakage. The total power consumed by a DRAM module is proportional to $V_{dd} \times \texttt{frequency}$. The nominal DDR4 standard specifies a V_{dd} around 1.3V, while the low-power DDR4 standard suggests a V_{dd} around 1.05V.

Lowering the frequency has been attempted in prior works [52, 53], but this comes with the downside of reducing memory bandwidth. Dynamic frequency-scaling/voltage-scaling (DVFS) in DRAM suffers from performance degradation with modest reductions in power consumption [35]. In this

work, we focus solely on voltage scaling, and study the effects on stability, bit storage reliability, and final neural network performance.

3. PRIOR WORK

This work builds on a large body of related work in (1) DRAM characterization, (2) approximate computing, and (3) neural acceleration. We outline related works in this sections. We briefly summarize the work, their relation to EDEN, and EDEN's key differentiating factors.

3.1 DRAM Characterization

Significant work has been done to study the effect of aggressive scaling on reliability and operation of DRAM, looking solely at the device level. We outline works that attempt to characterize the effects of modifying three DRAM knobs: (1) refresh, (2) read/write latencies, and (3) voltage.

A number of studies have looked at effect of reducing DRAM refresh [54], refresh latency [55], minimally executing refresh [56], or omitting DRAM refresh altogether [57, 58].

Other studies have investigated the error patterns reducing DRAM timing. In particular, Adaptive-Latency DRAM [36] studies the extra timing margins introduced by manufacturers to ensure bit reliability, while [59] and [60] propose a system to exploit latency variation across a bank, through error profiling of the rows. These three works formed the motivation for our work to extend these works into the sub-reliable operation regimes, for the specific goal of neural network inference.

Finally, some papers have studied the effects of reduced-voltage operation. In particular, [61] studies 50 DDR3L DRAM modules to characterize power consumption and create a higher fidelity DRAM power model. [35] studies voltage scaling techniques while maintaining complete bit reliability.

EDEN leverages the profiling techniques developed in [35] and [60] to obtain $V_{dd}/t_{RCD}/t_{RP}$-error relationships used in our specific error-resilient application.

3.2 Approximate Computing

EDEN falls under the large and well-studied field of approximate computing. We outline works that touch upon approximate computation in machine learning, DRAM, and neural network inference.

[62] is one of the first works we found that suggests using approximate DRAM and SRAM for general computing applications; this work does not provide any specific implementation details. [63] and [64] refine this idea by proposing refresh scaling, and studying error rates with refresh mitigation, similar to prior DRAM characterization works. [65] studies the effect of fuzzy arithmetic for SVM applications, while [66] studies using unreliable refresh-scaled DRAM for video analytics. [67] proposes extensions to DRAM circuitry to support bucketing application components into 'reliable' and 'unreliable', using refresh interval scaling.

Both [68] and [69] adjusts numeric bitwidths of neural network parameters, and label this precision reduction as approximate computing of a neural network. The crux of these works was the design of low-precision arithmetic units.

[70] studies effects of ECC, and the theoretical accuracies achieved by neural networks after turning off ECC for a one-time load of the network from disk. [71] studies neural network evaluation with low-precision numbers and error injection with MSB protection. Any efficient implementation of MSB protection is not discussed, and examples beyond the toy MNIST dataset are unfortunately not considered.

[72] propose a specialized ECC variant for patterns present in the weights of shallow neural networks to combat unreliable disk reads. Similarly, [73] presents another fault-mitigation for neural networks that minimizes errors in faulty registers. These works are orthogonal to our works making the model more robust, and focusing on DRAM error models.

Finally, [74], [75], and [76] provide interesting insights into the base error resiliency of neural networks. In particular, they study uniform random SRAM faults. This last manuscript provides interesting analysis of the effect of various numeric representations on bit resiliency, motivating our study across different bit-widths. These works do not study the effect of re-training to improve resiliency, different error models, or try to automatically characterize the highest error resiliency of each network component.

While taking inspiration from these works, we find that EDEN is unique in its focus on multiple DRAM-based error models, its use of neural networks as the tool to absorb reliability problems, and its use of latency and voltage as conduits for improving performance.

3.3 Efficient Neural Network Inference

A significant number of works in the last two years have investigated use of ideas from approximate computing for model inference in hardware. We outline and differentiate all such works that we have found in our literature search.

RANA [77] proposes refresh mitigation of the eDRAM in a custom eDRAM-based neural accelerator. [78] adopts a similar idea. We improve these works to include error models beyond uniform random, automatic optimization, and memory error mechanisms beyond refresh.

[79] and [80] design neuromorphic accelerator chips that use emerging memory technologies (spintronics and memristors) to run small proof-of-concept neural networks in a fuzzy manner. We focus on DRAM error models, and focus on use of simulation to mitigate errors through model re-training.

ThUnderVolt [81] and Minerva [82] are two recent works that proposes voltage underscaling of arithmetic elements and SRAM, respectively, in the spatial array of neural network accelerators. Because we focus on off-chip memory, multiple new error models, and provide a strong automatic model-hardening procedure that can be used in all three works, we find these work quite complementary to ours. Our work is also extensible to hardware platforms beyond neural acceleration ASICs.

4. UNDERSTANDING DNN RESILIENCE

4.1 Our DRAM Error Models

EDEN relies on knowledge of a timing parameter-error curve in order to estimate t_{RCD}/t_{RP} targets that correspond to a tolerable bit-error rate. To obtain a realistic characterization of real DRAM devices, we use the FPGA-based SoftMC infrastructure [83] to run studies on *real DDR4 modules* from three DRAM vendors. SoftMC allows us to execute memory

controller commands to each individual bank. In the results shown below, we analyze bit error rates at room temperature, writing and reading the parameter and IFM values to different banks. Figures 4 and 5 are drawn from data taken through our characterization of DDR4 modules. Figures 3 and 6 are plotted from data gathered in [35] and [59].

In order to define reasonable but simulation-friendly error models, we looked to the data to understand error patterns. What follows in this section is our observations, how this informed our choice of error models, and how this affects the performance of EDEN.

1. EDEN is most interested in regions of these error curves where small increases in error correspond to large (or linear) reductions in voltage/latency. We see this behavior among a subset of memory chips and data patterns (e.g. Chip 3 in Figure 5). In the worst case (e.g. vendor C in Figure 6), we see a considerable discontinuity jumping from 0 to 100% BER with a supply voltage reduction of <0.05V. In this case, EDEN has little room to suggest DRAM or model optimizations.

2. We tested loads and stores with actual model data (in addition to standard test patterns such `0xFF`, `0x00`) to approximate the effects of data pattern dependence (DPD) that might bias our characterization [51]. We tested accessing byte samples from the ResNet101 parameter and IFM set. A sample of our results are shown in Figures 4 and 5. We did not notice significant differences in the error curves compared to the data bytes considered in Figure 3.

3. We see that while some vendors and banks exhibit nearly uniform random errors (Vendor A in Figure 4), it is clear that there are also row-correlated error patterns (along the bit-lines) and column-correlated error patterns (along the word-lines). This is why we consider both uniform bit error models (Error Model 0 and 1), as well as entire adjacent bit-group flips (Error Model 2 and 3)

4. It is beneficial that EDEN performs at least a one-time error profiling of its chosen banks. There is a pronounced vendor dependence, and to a lesser degree, chip and bank dependence on the overall error pattern. Secondly, these error profiles are temperature dependent. For EDEN to operate correctly, profiling must occur at the same temperature at which EDEN runs the optimized model, or at a temperature in which the error model is known.

5. In Figure 3, we see data pattern dependence based on the switching pattern, 0-to-1 or 1-to-0. This is likely due to data dependent behavior in the DRAM sense amplifier: the sense amplifier is more quickly able to latch onto the positively charged readings, reducing the effect of lowering the read latency. *This implies that errors are biased towards 0-to-1 flips*. This is reflected in our Error Models 1 and 3. [35] finds there are no statistically significant data pattern biases present during voltage scaling, through for certain vendors (e.g. Vendor A in Figure 6), we see a small trend towards a one-flip bias. Error Model 1

To summarize, this work uses four error models:

- Error Model 0 is the widely used error model of uniform random bit errors (flipping bits at random)
- Error Model 1 implements Error Model 0, but flips one bits with 2:1 odds against flipping a zero (more one bits are flipped than zero bits). This model is motivated by DRAM cell leakage (which is exacerbated during DRAM refresh omission).
- Error Model 2 is uniform random word flip (row-adjacent bits), flipping all bits in the word, motivated based on observed spatial error patterns.
- Error Model 3 implements Error Model 1, but flips zero bits with 2:1 odds against one bits, motivated by our observations on data patterns from latency-scaling.

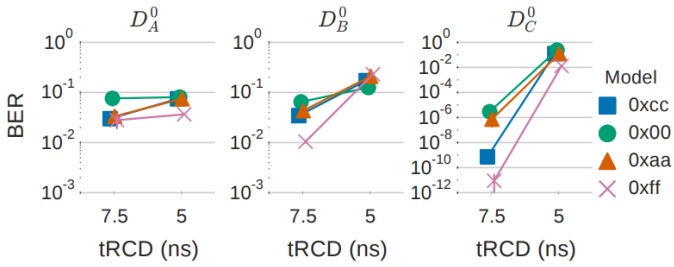

Figure 3: DRAM errors while scaling t_{RCD} [59], using four basic data patterns on a DDR3 module. With decreased t_{RCD}, 0 to 1 flips are more common.

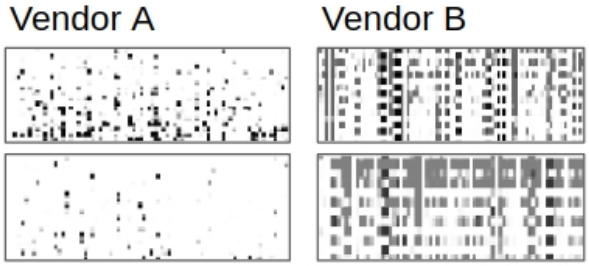

Figure 4: Spatial pattern of errors while scaling t_{RCD} to 50% of nominal, using sampled ResNet101 bytes stored on a DDR4 module. Each rectangle represents a bank. Note that fairly uniform error patterns from Vendor A and word- and bit- line correlated error patterns from Vendor B

In our evaluations, we apply our memory error model in a cache-oblivious manner, ignoring data re-use that would decrease the effective bit error rate. This is to provide a worst-case estimate of the bit error rate, and determine the worst-case impact on neural network accuracy. Our simulations assume their caches are error-free.

4.2 Baseline Network Performance

Table 2 demonstrates the baseline accuracy of our networks on their respective testing datasets. As pre-inference parameter quantization is commonly performed on real systems in order to reduce memory costs, we find it prudent to consider low-bitwidth networks in our error resiliency evaluations. We use the symmetric linear quantization scheme described in [84]. This quantization scheme applies weight-dependent affine scaling to linearly map weights into range $[-2^{b-1}, 2^{b-1}-1]$, where b is the target model weight bitwidth.

Our baseline model accuracies match stated numbers in relevant literature [41, 27, 40, 38, 28]. We find increasing test

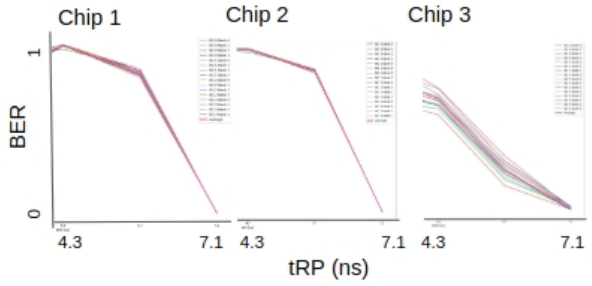

Figure 5: Bit error rate while scaling t_{RP}. We use sampled ResNet101 bytes stored on a DDR4 module. Each graph plots data from a DRAM chip, each line represents a bank within the chip. For banks on Chip 3, we can reduce t_{RP} from 7.2ns to 5.7ns (21%), and obtain a <7% BER. If EDEN finds the chosen networks can be retrained to tolerate this BER, the reduction is enabled.

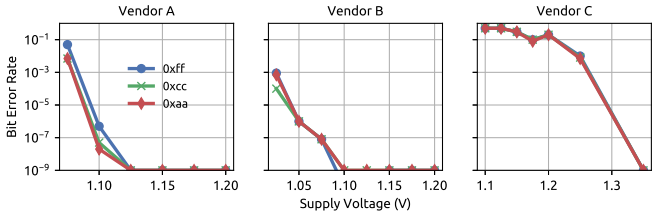

Figure 6: DRAM errors while scaling V_{dd} [35], using three data patterns on a DDR3 module. Based on the vendor, we see different scaling capacities. For Vendor A and B DRAM modules, if EDEN discovers a network tolerance of $> 10^{-2}$ BER, V_{dd} is reduced from a nominal $1.35V$ to $1.05V$.

image recognition accuracy on with increasing bitwidth. Two of the models, DenseNet201 and SqueezeNet1.1, suffer from accuracy collapse at our lowest bitwidth, in the presence of higher levels of quantization error.

In the sections that follow, we provide figures representative examples of patterns and observations across all our evaluation datasets and discuss variances we notice across models. This is to provide a concise and clear exposition. The complete set of results can be found at the anonymized Appendix, `https://goo.gl/imfQch`.

4.3 Usage of a Memory Error Model

Our procedure for implementing error models within network inference libraries follows a memory-as-a-noisy-channel paradigm. The sequence of stages is shown in Figure 7.

We model an errorful write and corresponding errorful read as perturbations on top of the correct, original value. For simulation, we created a framework on top of PyTorch [85] that allows us to modify the loading of parameters and IFMs, and perturb them based on our desired error model.

We find that approximate fault mitigation on a noisy memory reads is necessary for avoiding accuracy collapse on 32-bit floating point networks. This stems from the fact that a single bit error in the exponent bits of a floating point network creates an enormously large value ($>10^8$) that propagates through the network, dominating weights that are significantly smaller ($<10^1$).

Model	4-Bit	8-Bit	16-Bit	32-Bit
ResNet101	89.11%	93.14%	93.11%	94.20%
MobileNetV2	51.00%	70.44%	70.46%	78.35%
VGG16	59.05%	70.48%	70.53%	71.59%
DenseNet201	0.31%	74.60%	74.82%	76.90%
SqueezeNet1.1	8.07%	57.07%	57.39%	58.18%

Table 2: Our baseline accuracies (0% Bit Error Rate) of the networks used in evaluation. The first two networks are evaluated on the CIFAR-10; the last three on Imagenet ILSVRC2012.

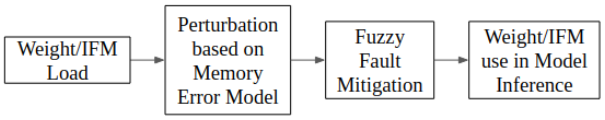

Figure 7: Flow for using a memory error model during inference and re-training. These stages approximate memory as a noisy read/write channel.

Our approximate fault mitigation uses two comparisons with pre-calculated thresholds to attempt a fuzzy correction of values that are not plausible. Both the parameters and IFMs of a provided network are within a known range that can be computed at training time. For every loaded value, we perform a comparison with a network-wide maximum and minimum value, and zeros the value if it out of range. We find that this increases the tolerable bit error rate from 10^{-7} to 10^{-3} to achieve <5% accuracy degradation in baseline FP32 networks, with similar improvements in our FxPt16/8 networks. As we discuss in Section 5, this can be done as a low-overhead stage in memory controller hardware or in software.

4.4 Baseline Error Resilience

In order to study the error resiliencies, we test model accuracy at different bit error rates, for each of our four models. Figure 8 illustrates the results for two of the models. We find that our larger baseline models begin a drop-off in accuracy around 10^{-3} to 10^{-2}. Interestingly, the performance of different bitwidth networks also varies based on the error model, with FP32 performing the worst or near to worst on the bit-flip based error models and the best on word-flip based error models. We speculate that this is because in FP32 models, at a fixed bit error rate, there is a higher likelihood that each parameter is perturbed because of the longer numeric length. At the same BER, if you can guarantee that all errors are isolated to word groups (Error Model 2), FP32 performs quite well, dropping off at 2% BER. We find little generizable similarities in the patterns of Error Models 1 and 4.

Smaller models, such as MobileNetV2 and SqueezeNet1.1, show noisier results and earlier drop-off. They exhibit the same FP32/FxPt16/8/4 trends as noted above.

4.5 Error Resilience After Re-Training

Prior work in quantization and approximate computing note that re-training with a uniform error model can help with network robustness [86, 87, 77, 79]. This body of work purports that re-training conditions the network to become familiar with the noisy weight and IFM estimates.

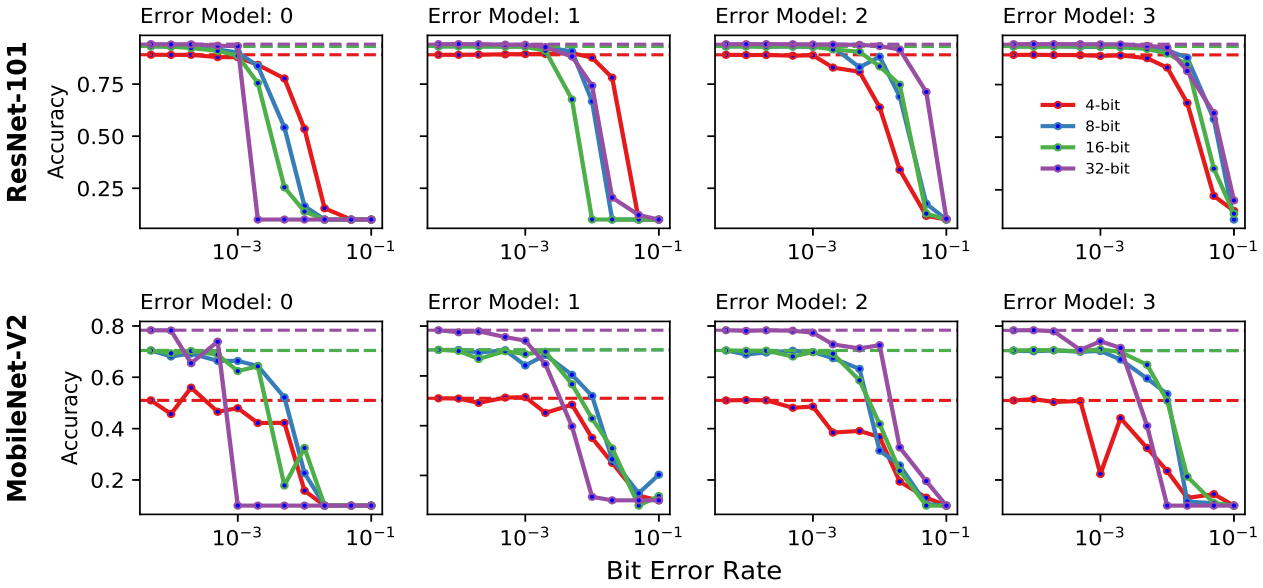

Figure 8: Introducing memory errors into ResNet101 and MobileNetV2. We measure test accuracy after we introduce four different unreliable memory models across different bit error rates. Each point is a different evaluation of the network with different error rate/error model configuration.

We suggest a refinement to this observation: **re-training is only effective with an error model that closely matches the error patterns of the target hardware**. In a number of cases, we noticed that a domain mis-match causes the re-trained model to collapse, and perform poorly in the target error environment.

Figure 9 describes our results re-training ResNet101. In particular, we assume four different underlying error patterns of the underlying hardware (our test-time error model), and re-train with a uniform random error model (Error Model 0). We find that in two of these cases, re-training with the mis-matched error model (blue line) results in improvements over no-retraining, but in two other cases, the re-trained model suffers from accuracy collapse when encountering the test-time error patterns. We notice this pattern across all five models that we tested.

Re-training with the same-error model improves the 5% accuracy degradation point by 78x on average across the error models: from a bit-error rate of roughly 5×10^{-4} to roughly 2×10^{-2}.

In the upper-left graph in Figure 9, the actual error model was Error Model 0, and our 'mis-matched' error model which was used to re-train was Error Model 1. Our complete hyper-parameter settings for re-training (e.g. learning rate decay schedule) can be found in the link to the manuscript's code.

4.6 Error Resilience After Pruning

In our attempts to improve the error resiliency of networks, we explored the effect of pruning networks. Pruning, a common procedure prior to compressing weights, zeroes low-magnitude parameters in the neural network [46]. Given that some of our error models test resilience to flips to zero, we hypothesized that such sparsification would have a positive effect on error resilience.

Interestingly, we find that pruning does not consistently help or hurt error resiliency. Figure 2 in the appendix demonstrates that the resulting error curves varying prune rate levels are roughly the same; this includes error models that bias bits towards zero. Testing our smaller SqueezeNet network yielded similar conclusions: no clear trends related to pruning. As such, we do not include this among EDEN's network optimizations for improved error resilience. We speculate that pruning increases the burden of information propagation to fewer key neurons in each layer; at high error rates, if these neurons are affected, the decreases in accuracy are great.

5. EDEN

We used our analysis of neural network error tolerances and DRAM error characteristics to design and introduce EDEN. EDEN is a system to improve and leverage the error resiliency of neural networks to more aggressively DRAM during inference.

EDEN is composed of main three stages: memory characterization, neural network characterization, and model-memory mapping. Users of EDEN can select between a power-improvement mode or a latency-improvement mode. An overview is provided in Figure 1. After selecting the correct mode (choosing between scaling V_{dd} or $t_{RCD/RP}$), we profile the memory with respect to the mode (Section 5.1). At this point, decide whether the memory architecture is able to be partitioned in a fine-grained manner based on the values of the chosen mode. Based on the partionability, we characterize the pre-trained model with respect to an accuracy target and error model (Section 5.2) to obtain a set of tolerable bit error rates. Then, we link our network characterization and error characterization with a basic network-memory mapping described in Section 5.5. EDEN is supported by memory controller modifications and optional device-level modifications

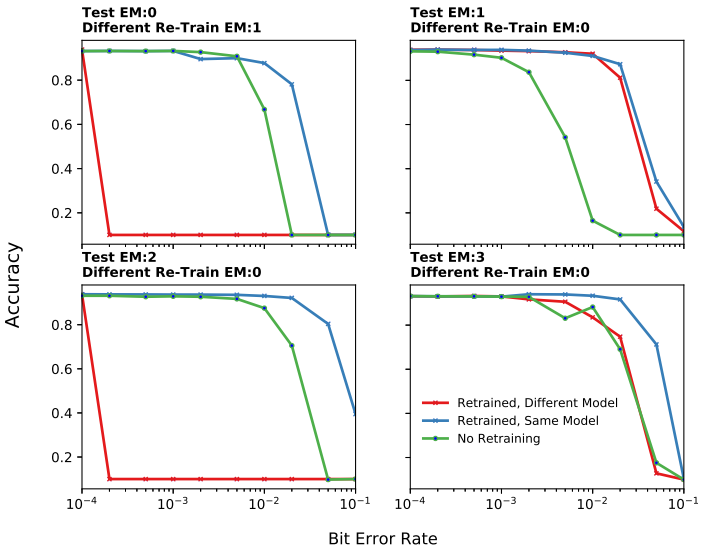

Figure 9: Accuracy of ResNet101 in presence of memory errors, with different error models applied during re-training and testing. Each graph is labeled with the true (test) error model, and re-trained with Error Model 0, with exception of the upper left graph which is re-trained with Error Model 1.

which we describe in Section 5.6.

Algorithm 1 Overview of EDEN

Input: target, network, accuracy_goal, memory, mode
Output: retrained_network, mem_config

curves, err_model ← `mem_profile`(memory, mode)
partition ← `hw_partitionable`(memory, mode)

if partition **then**
    BERs, new_net ← \
     `fine_profile`(network, err_model, accuracy_goal)
    mem_config ← `fine_map`(error_curves, BERs)
else
    BER, new_net ← \
     `flat_profile`(network, err_model, accuracy_goal)
    mem_config ← `flat_map`(error_curves, BER)
return new_net, mem_config

5.1 Memory Characterization

EDEN requires knowledge of the error profile of a memory device in order to properly choose memory settings that correspond to target bit error rates.

For a first-order estimate, without any device specific information, EDEN first re-uses hard-coded error curves derived prior works that characterize vendor-specific error patterns. This provides EDEN an understanding of mean and variance of error curves, of which EDEN uses the worst-case, conservative mapping: mapping lowest characterized BER for each set of $V_{dd}/t_{RCD}/t_{RP}$ in its mapping. In our evaluation, we reference [60, 35, 88] as excellent sources to obtain reasonable first-order estimates of BER curves like those in Section 4.

We find EDEN can achieve significantly more aggressive memory settings by doing a one-time profiling of the target device prior to running inference. In particular, based on our own collected data on real DRAM DIMMs using SoftMC [83], we find that chips from the same vendor possess BERs that differ by one or two orders of magnitude for the same t_{RP} settings. In reverse, to achieve the same BER of 10^{-2}, EDEN operates these chips with a minimum t_{RP} differing by more than 2 nanoseconds. If EDEN solely used reference values without profiling, EDEN would have to assume a worst case t_{RP}/t_{RCD} across all chips that were provided by the reference.

EDEN currently uses of a one-time error profiling step through use of a standard 'brute-force' profiling methodology for a DRAM bank from [89], also detailed by Algorithm 1 in the Appendix. It is important to note that such profiling is possible because of EDEN-enabled firmware and memory controller changes that allow for modification of timing parameter settings and voltages to DIMMs. For EDEN to use fine-grained chip, bank, or sub-array-level neural network partitioning (Section 5.2), EDEN requires the error curve of each partition. This granularity of error modeling is available only through this one-time profiling.

Other error profiling methodologies are potentially available for EDEN that leverage aggressive memory conditions (e.g. high temperatures) to obtain profiles 2-5x faster [89, 88] than the current 10 minutes it takes to profile a ResNet101-sized DIMM partition via SoftMC. We leave such an implementation as a future addition to EDEN.

Profiling also helps EDEN select the closest matching error model for use during re-training. Among our four error models, we evaluate 'closeness' based on high-level characteristics informed through statistics. If EDEN sees a strong bias towards $0 \rightarrow 1$ or $1 \rightarrow 0$ errors (>10% difference), it selects Error Model 1 or 3, respectively. If EDEN sees $> 25\%$ errors are neighboring errors (spatial correlation), it chooses characteristics of Error Model 2. In all other conditions, EDEN chooses its default uniform random Error Model 0.

5.2 Automatic Network Characterization

Matching memory error characteristics to network characteristics requires an understanding of the error tolerances of the input neural network. For this, we provide two alternatives, based on the level of partitioning desired and available DRAM hardware: *flat characterization* and *fine-grained characterization*.

Both modes are provided a specific *accuracy target*, the pre-trained *network*, and estimated memory device *error model*. EDEN's network must find the optimal BER across the entire network that meets that allowable accuracy degradation.

5.3 Flat Characterization

In cases where the DRAM transistor arrays are off-the-shelf, it is unlikely that EDEN can expect to be able to configure a per-chip or per-sub-array V_{dd} or $t_{RCD/RP}$. For instance, t_{RCD} is configurable in some OS BIOS menus, but the selected configuration is applied across the entire DRAM DIMM. This means, that if EDEN selects a particular t_{RCD}, the same worst-case BER must be applied across all weights and IFMs in the network.

Model	BER	Est. Δt_{RCD}	Est. ΔV_{DD}
ResNet101	4.0%	5.5ns	0.25V
MobileNetV2	0.5%	0ns	0.20V
VGG-16	5.0%	6ns	0.30V
DenseNet201	1.5%	0.5ns	0.20V
SqueezeNet1.1	0.01%	0ns	0.10V

Table 3: Maximum tolerable BER across the IFMs and weights of each evaluation network found through EDEN's blanket characterization. Accuracy target is 2.5% degradation. We map the neural network BER to reductions in nominal t_{RCD} and V_{DD} (13ns and 1.25V, respectively) on a pre-characterized DDR4 DRAM module (Vendor B) [35].

Algorithm 2 describes our procedure for conducting blanket network characterization. In short, we perform binary search on the error rates on a logarithmic scale to find the highest BER that still yields the accuracy goal. Binary search is possible because error curves are largely monotonic relationships. Of note, during every iteration of binary search, we re-train the input pre-trained network with the provided error model for ten epochs, before perturbing it and testing the validation accuracy.

Algorithm 2 Blanket (Flat) Network Characterization

```
function FLAT_PROFILE(net, accuracy_goal, err_model)
    err_min, err_max ← 0, 1
    if infer(net, error= 0) < target_accuracy then
        return Accuracy Target Not Possible
    error_rate, net ← \
     binary_search(net, accuracy_goal,
                   err_model, err_min, err_max)
return error_rate, net
```

We run blanket characterization on our five evaluation networks from Section 4, with Error Model 0, tolerable accuracy degradation of 5% for demonstration, and a Vendor A DIMM from our collection. Our results are shown in Table 3.

5.4 Fine-Grained Characterization

Prior work has noted the differences in error tolerances of different network components [76, 75]: the first and last layers, for example, generally have lower tolerance for errors caused by quantization. An example of the differing error tolerances for the convolutional and linear layers in our ResNet101 model is provided in the appendix.

Optionally, EDEN can exploit this variance in error tolerances by partitioning memory into different buckets with different t_{RCD}/V_{DD}, provided there is hardware support for such fine-grained chip-level, bank-level, or sub-array-level configurations. Accordingly, EDEN must characterize the error tolerance of each weight and IFM based on tolerable BER. For this, we apply a simple fine-grained network characterization algorithm described in Algorithm 3. For each parameter and IFM, we search along a 10-point log scale of bit error rates for the maximum tolerable BER. After testing the result of boosting the BER of each parameter, we do a full network re-train with the new BER-error model. In practice, to reduce running time of the procedure, we sample 10% of the validation set during `infer` to obtain the accuracy estimate. We also bootstrap the BERs to the BER found in flat characterization and use a linear scale in 0.5 increments around that value. For ResNet101, this one-time characterization completed in two hours using an Intel Xeon CPU E3-1225. We find that fine-grained partitioning yields individual kernels whose BER is 3x larger than the minimum common denominator BER found by flat characterization.

The exploration space of Algorithm 3 is exponential in network size, so we emphasize that Algorithm 3 provides a greedy solution to the error tolerance search.

Algorithm 3 Fine-Grained Network Characterization

```
function FINE_PROFILE(net, accuracy_goal, err_model)
    log_BERs = [1e-5, 5e-5, 1e-4, ..., 1]
    kernels ← net.weights, net.IFMs
    for kernel ∈ kernels do                    ▷ Initialization
        kernel.log_BERs_index ← 0
    repeat
        for kernel ∈ kernels do
            kernel.log_BERs_index += 1
            acc ← infer(kernels, err_model)
            if acc < accuracy_goal then        ▷ Roll Back
                kernel.log_BERs_index -= 1
        retrain(kernels, err_model)
    until no increases in BER during iteration
return log_BERs[kernels.log_BERs], kernels
```

Note that this characterization procedure is cache unaware, so it will attempt to test the BER of a weight or IFM that may not be read from main memory. As noted above, this modeling choice means that our estimate of tolerable bit error rates is an upper bound, allowing for correct execution in a system with a cache hierarchy that meets the accuracy target.

5.5 Network-to-Memory Mapping

The final step that EDEN performs is linking the memory characterization, network characterization, and device's available memory partitions. In the first setting, the available memory configurations are course-grained; this means that EDEN can only change a global memory parameter, such as t_{RCD} for a whole DIMM. In this case, EDEN assumes the highest-BER error curve among curves describing banks in the DIMM and scales t_{RCD}/V_{DD} down to the point where BER matches the network's tolerable BER. In the case that the memory configuration can be partitioned at a chip, bank, or sub-array level granularity, we apply a straight-forward tensor bucketing procedure.

This is demonstrated in Figure 10 for the weights and IFMs of ResNet-101. We begin with a fine-grain profile of the tolerable BERs of each residual block within the network; this is obtained using our `fine_profile` procedure. We then sort and partition the residual blocks based on the minimum tolerable BER of each group.

5.6 Memory Controller

To obtain the most out of EDEN, we suggest minor memory controller changes in order to enable (1) approximate fault mitigation and (2) fine-grained memory partitioning.

5.6.1 Approximate Fault Mitigation

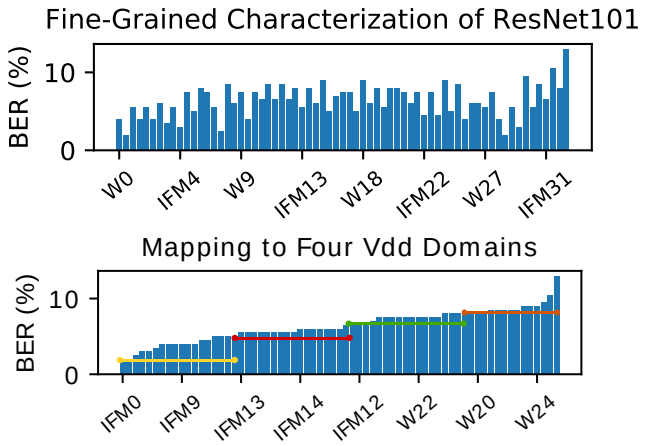

Figure 10: Applying fine-tune characterization of the tolerable BERs of ResNet-101 kernels (top) followed by a mapping into the four colored V_{DD} domains.

In Section 4.3, we describe an accuracy collapse that we witness in FP32 networks when increasing the bit error rate beyond 10^{-6} in large networks. This is primarily caused by bit flips in the MSBs of the exponent segment in a floating-point number which causes a few high numbers near the limits of FP32 numeric range. These large values propagate, spread `NaN`'s throughout the network, and at the end, result in extremely low or `NaN` accuracy for the neural network.

The key insight in our fault counter-measure is that empirical range of values for the IFMs and weights in networks is rather small: most layers in SqueezeNet1.1, for example, have weights within the range [-5,5]. Critically, this allows detection of a number that is likely at fault by checking against this threshold. Upon detection of an error, we tested two low-cost correction techniques: (1) zeroing out the number and (2) saturating the number on the bounds of the range. We found that fault correction technique (1) is superior to (2), and in our tests, (1) obtained higher network accuracies at the same BER across all tasks (difference of 11% in CIFAR-10 recognition and 9% in ImageNet).

While it is possible to do this fuzzy fault mitigation in software with two instructions, we believe that a small addition to the the memory controller could accommodate this fault logic, with significantly less performance overhead. The single-cycle circuit uses two 8-bit comparators to compare the exponent part of the floating point input value against a fixed, neural network specific threshold. The comparator resets the input value if the threshold is exceeded. By storing a per-network value in the memory controller and using an inexpensive implementation of weight bounding within the pipeline, EDEN would be able to improve the BER tolerance of FP32 networks by three orders of magnitude.

5.6.2 *Enabling Fine-Grained Partitioning*

In today's commodity systems, the memory controller often sets DRAM latency values during start-up and does not change these values in run-time due to reliability concerns. As most aggressive usage of EDEN requires dynamic adjustment, we propose additional commands that set latency values in the memory controller.

Cores:	2 & 4 Cores @ 4.0 GHz; 32nm; 4-wide OoO; Buffers: 18-entry fetch, 128-entry decode; 128-entry ROB
L1 Caches:	32KB, 8-way, 2-cycle, Split Data/Instruction
L2 Caches:	512KB per core, 8-way, 4-cycle, Shared Data/Instruction, Stream Prefetcher
L3 Caches:	8MB per core, 16-way,6-cycle, Shared Data/instruction, Stream Prefetcher

Table 4: ZSim Configuration

In principle, a memory controller can use separate t_{RCD} and t_{RP} values for each row. This, however, incurs large overhead to store this meta-data table. However, most commonly used neural network architectures (and the ones evaluated in this paper) have at most to 1024 different error-resilient components. Therefore, it suffices for EDEN to split DRAM into *at most* 2^{10} regions based on their error characteristics. Considering the latency values and the clock frequencies, each of t_{RCD} and t_{RP} can fit into 4-bits. The total size of storing meta-data is upper-bounded by 1KB. Even if DRAM subarray level granularity is needed, the size of meta-data reaches up to only 2kB for an 8GB DDR4 device.

Fine-granularity voltage scaling requires a different set of changes. Voltron [35] analyzes a robust design for voltage scaling at the bank granularity based on modest changes to the power delivery network; in this work, we leverage this work as the underlying hardware platform enabling fine granularity voltage scaling. Note that commodity DDR4 and LPDDR4 chips have 16 and 32 total banks, respectively. As such, the corresponding meta-data by adding EDEN is less than 32B, because voltage level can be represented by 8 bits

6. HARDWARE EVALUATION

6.1 CPU Inference

We evaluate EDEN on top of a multi-core OoO CPU using the simulated core configuration listed in Table 4. We use ZSim [90] and Ramulator [91] to to simulate performance and DRAMPower [92] to estimate memory power consumption for DDR4 and LPDDR3 devices.

We evaluate EDEN's flat characterization procedure. Our improvements from flat characterization are a lower bound to those achievable with fine-grained characterization. For the purposes of our simulation, we use four different neural network inference benchmarks: two from the Intel OpenVINO toolkit (running VGG16 and a VGG16-based Faster-RCNN model) [93] and two from the DarkNet framework (running a ResNet101-based YOLOv3 and a YOLOv3-Tiny model) [94]. We choose these inference models as they closely match the tolerable BER values of our evaluated models from Section 4. We use the parameter scaling listed in Table 3; in particular, (1) we test the impact of latency by reducing each of t_{RCD} and t_{RP} to 75% and 50%; (2) we scale V_{DD} by 30%.

Figure 11 summarizes our read-latency evaluation. We find that while most networks exhibit no overall cycle speed-up from these latency reductions, there are a few networks that witness a 7-15% speed-up from our read latency changes.

Further investigation using Intel VTune reveal that majority of the networks (Faster-RCNN in particular) are *not* memory latency bound (bottlenecked by the floating point unit or memory bandwidth in most cases we examined), explaining the near 0% end-to-end speed-up for many configurations. We demonstrate though, that there are networks, that can achieve improvement through $t_{RCD/RP}$ reduction because of memory latency bottlenecks.

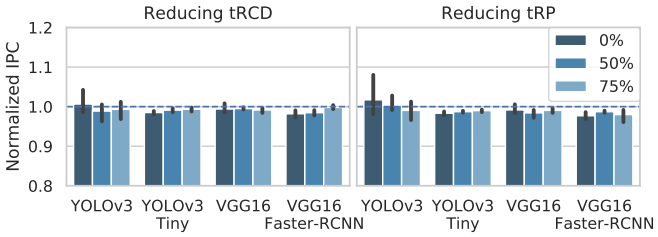

Figure 11: CPU Cycle Count Reduction Scaling t_{RCD}

In our DRAMPower studies, we find reductions in power across all inference benchmarks. Using DDR4, we find the mean average power reduction is 30.12% with variance 0.1% across all benchmarks. Using LPDDR4, we find the mean average power reduction is 29.98% with variance 0.07% across all benchmarks. This is expected when we scale supply voltage.

6.2 GPU Inference

Next, we evaluate EDEN on top of a GPGPU. For this, we use the cycle-accurate GPGPU simulator, GPGGPU-Sim [95], and its inbuilt power modeling/breakdown tool, GPUWattch. Our GPU configuration models an NVIDIA GTX480 and can be found in Table 5. For power evaluations, we apply the same voltage scaling factors as before. We adapted our DarkNet binaries to run on GPGPUSim, providing us with evaluations for the YOLOv3 and YOLOv3-Tiny networks. GPGPU-Sim does not provide a CUDA Driver API in simulation, and this is used by most mainstream GPU-enabled neural network frameworks (Tensorflow, TVM, Caffe), limiting our benchmarks to our two DarkNet binaries.

We find that for the YOLOv3-Tiny network, DRAM power consumption decreases by 40.57% between our nominal and reduced voltage settings. For the YOLOv3 network, we see a 31.0% DRAM power reduction between nominal and reduced.

In our latency studies, we find that with a 75% decreased t_{RCD}, the total cycle count of YOLOv3-Tiny execution drops 4%. With the same reduced t_{RCD}, the YOLOv3 benchmark retains the same exact total cycle count. We suspect that this is because on a NVIDIA GTX480, the YOLOv3 network is not memory latency bound.

6.3 Neural Network Inference Accelerators

Finally, we study voltage scaling in the off-chip memory of two commonly used neural network accelerator architectures: Eyeriss [10] and Google's Tensor Processing Unit [96]. We use the recently released cycle-accurate systolic array simulator SCALE-Sim [97] to model the two architectures, and DRAMPower to obtain power estimates from memory traces produced by SCALE-Sim. We use the built-in AlexNet and YOLOv3-Tiny models.

Shader Core	15 Clusters, 700 MHz, 32 SIMT Width, 48 Warps per Core, 2 GTO Schedulers per Core
Private L1 Cache	16 KB, 8-way, Cache Block Size 128B
Shared Memory	48 KB, 32 Banks
Shared L2 Cache	786 KB, 8-way
DRAM	GDDR5, 924MHz, 6 channels, 16 banks, $T_{RCD} = 12$, $T_{RAS} = 28$, $T_{RP} = 12$

Table 5: GPGPU-Sim Configuration

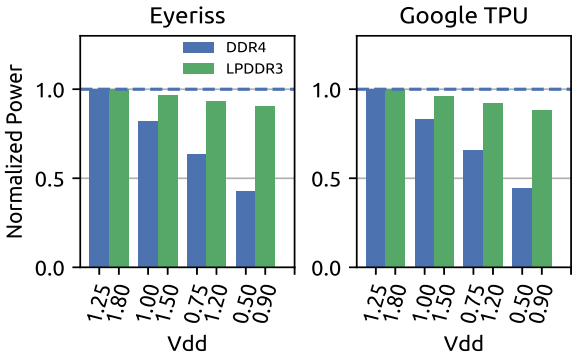

Figure 12: Scaling V_{DD} on the two neural network accelerators, Eyeriss and Tensor Processing Unit.

Figure 12 displays the results for our YOLOv3-Tiny benchmarks. We find that scaling V_{DD} by 20% yields a 19% reduction in power. This target V_{DD} scale corresponds to an estimated BER in which both networks suffer from a <3% accuracy reduction. Our AlexNet benchmarks yield similar power reduction results (within 2% difference).

7. CONCLUSION

This paper introduces EDEN: a cross-platform framework that enables the use of approximate DRAM to achieve better energy efficiency and execution time for neural network inference. EDEN exploits supply voltage scaling and read latency scaling in DRAM, but we stress that the core principles of EDEN and its techniques of boosting neural network error resilience generalize across memory technologies, ECC settings, and other memory configurations.

We study of error resiliency in state-of-art computer vision networks and demonstrate the importance of accurate, empirically-derived memory error models. We hope that EDEN is a fruitful starting point for further research on the use of practical, approximate memory for machine learning workloads.

8. REFERENCES

[1] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, *et al.*, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural networks: the statistical mechanics perspective*, vol. 261, p. 276, 1995.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[3] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International Conference on Machine Learning*, pp. 1764–1772, 2014.

[4] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*, pp. 184–199, Springer, 2014.

[5] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[7] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks.," in *ICASSP*, vol. 14, pp. 4087–4091, Citeseer, 2014.

[8] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[9] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, *et al.*, "Google workloads for consumer devices: mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 316–331, ACM, 2018.

[10] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[11] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 20, IEEE Press, 2016.

[12] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 1–13, IEEE Press, 2016.

[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 243–254, IEEE, 2016.

[14] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM Sigplan Notices*, vol. 49, no. 4, pp. 269–284, 2014.

[15] L. Cavigelli and L. Benini, "Origami: A 803-gop/s/w convolutional network accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, 2017.

[16] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 27–40, ACM, 2017.

[17] O. Mutlu, "Main memory scaling: Challenges and solution directions," in *More than Moore Technologies for Next Generation Computer Design*, pp. 127–153, Springer, 2015.

[18] M. Peemen, A. A. Setio, B. Mesman, H. Corporaal, *et al.*, "Memory-centric accelerator design for convolutional neural networks.," in *ICCD*, vol. 2013, pp. 13–19, 2013.

[19] Intel and D. Levinthal, "Performance analysis guide for intelÂő coreâĎć i7 processor and intelÂő xeonâĎć 5500 processors," 2009.

[20] M. Horowitz, "Energy table for 45nm process."

[21] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5987–5995, IEEE, 2017.

[22] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[23] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The reversible residual network: Backpropagation without storing activations," in *Advances in Neural Information Processing Systems*, pp. 2214–2224, 2017.

[24] M. Nazemi, G. Pasandi, and M. Pedram, "Nullanet: Training deep neural networks for reduced-memory-access inference," *arXiv preprint arXiv:1807.08716*, 2018.

[25] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[26] I. Kokkinos, "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory.," in *CVPR*, vol. 2, p. 8, 2017.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[29] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *arXiv preprint arXiv:1803.04783*, 2018.

[30] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 27–39, IEEE Press, 2016.

[31] Y. Long, T. Na, and S. Mukhopadhyay, "Reram-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, pp. 1–14, 2018.

[32] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.

[33] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[34] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 751–764, 2017.

[35] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 10, 2017.

[36] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency dram: Optimizing dram timing for the common-case," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 489–501, IEEE, 2015.

[37] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and K. Goossens, "Exploiting expendable process-margins in drams for run-time performance optimization," in *Proceedings of the conference on Design, Automation & Test in Europe*, p. 173, European Design and Automation Association, 2014.

[38] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[39] N. Ramachandran, S. C. Hong, M. J. Sime, and G. A. Wilson, "Diabetic retinopathy screening using deep neural network," *Clinical & experimental ophthalmology*, vol. 46, no. 4, pp. 412–416, 2018.

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

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[42] S. S. Du and J. D. Lee, "On the power of over-parametrization in neural networks with quadratic activation," *arXiv preprint arXiv:1803.01206*, 2018.

[43] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "Towards understanding the role of over-parametrization in generalization of neural networks," *arXiv preprint arXiv:1805.12076*, 2018.

[44] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: an empirical study," *arXiv preprint arXiv:1802.08760*, 2018.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[46] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[47] C. De Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré, "High-accuracy low-precision training," *arXiv preprint arXiv:1803.03383*, 2018.

[48] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[49] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. Fei-Fei, "Ilsvrc-2012, 2012," *URL http://www. image-net. org/challenges/LSVRC*, 2012.

[50] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[51] B. Keeth and R. J. Baker, *DRAM Circuit Design: A Tutorial*. Wiley-IEEE Press, 2000.

[52] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, pp. 1–8, 2010.

[53] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," in *ACM SIGPLAN Notices*, vol. 46, pp. 225–238, ACM, 2011.

[54] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern dram devices: Implications for retention time profiling mechanisms," in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 60–71, ACM, 2013.

[55] A. Das, H. Hassan, and O. Mutlu, "Vrl-dram: improving dram performance via variable refresh latency," in *Proceedings of the 55th Annual Design Automation Conference*, p. 171, ACM, 2018.

[56] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 1–12, IEEE Computer Society, 2012.

[57] M. Jung, É. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn, "Omitting refresh: A case study for commodity and wide i/o drams," in *Proceedings of the 2015 International Symposium on Memory Systems*, pp. 85–91, ACM, 2015.

[58] A. Rahmati, M. Hicks, D. Holcomb, and K. Fu, "Refreshing thoughts on dram: Power saving vs. data integrity," in *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[59] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern dram chips: Experimental characterization, analysis, and optimization," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, pp. 323–336, ACM, 2016.

[60] K. K. Chang, "Understanding and improving the latency of dram-based memory systems," *arXiv preprint arXiv:1712.08304*, 2017.

[61] S. Ghose, A. G. Yağlıkçı, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, *et al.*, "What your dram power models are not telling you: Lessons from a detailed experimental study," *arXiv preprint arXiv:1807.05102*, 2018.

[62] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[63] X. Zhang, Y. Zhang, B. Childers, and J. Yang, "Award: Approximation-aware restore in further scaling dram," in *Proceedings of the Second International Symposium on Memory Systems*, pp. 322–324, ACM, 2016.

[64] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Approximate computing with partially unreliable dynamic random access memoryâĂŤapproximate dram," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pp. 1–4, IEEE, 2016.

[65] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, p. 113, ACM, 2013.

[66] S. Advani, N. Chandramoorthy, K. Swaminathan, K. Irick, Y. C. P. Cho, J. Sampson, and V. Narayanan, "Refresh enabled video analytics (reva): Implications on power and performance of dram supported embedded visual systems," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pp. 501–504, IEEE, 2014.

[67] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate dram," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1172–1187, 2017.

[68] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: an approximate computing framework for artificial neural network," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 701–706, EDA Consortium, 2015.

[69] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*, pp. 27–32, ACM, 2014.

[70] M. Qin, C. Sun, and D. Vucinic, "Robustness of neural networks against storage media errors," *arXiv preprint arXiv:1709.06173*, 2017.

[71] J. Marques, J. Andrade, and G. Falcao, "Unreliable memory operation on a convolutional neural network processor," in *Signal Processing Systems (SiPS), 2017 IEEE International Workshop on*, pp. 1–6, IEEE, 2017.

[72] W. Shi, Y. Wen, Z. Liu, X. Zhao, D. Boumber, R. Vilalta, and L. Xu, "Fault resilient physical neural networks on a single chip," in *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, p. 24, ACM, 2014.

[73] B. Salami, O. Unsal, and A. Cristal, "On the resilience of rtl nn accelerators: Fault characterization and mitigation," *arXiv preprint arXiv:1806.09679*, 2018.

[74] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[75] A. H. Salavati and A. Karbasi, "Multi-level error-resilient neural networks," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pp. 1064–1068, IEEE, 2012.

[76] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "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*, p. 8, ACM, 2017.

[77] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "Rana: towards efficient neural acceleration with refresh-optimized embedded dram," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 340–352, IEEE Press, 2018.

[78] D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pp. 1–5, IEEE, 2018.

[79] P. Panda, A. Sengupta, S. S. Sarwar, G. Srinivasan, S. Venkataramani, A. Raghunathan, and K. Roy, "Cross-layer approximations for neuromorphic computing: from devices to circuits and systems," in *Proceedings of the 53rd Annual Design Automation Conference*, p. 98, ACM, 2016.

[80] Y. Kim, *Energy efficient and error resilient neuromorphic computing in VLSI*. PhD thesis, MIT, 2013.

[81] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: Enabling aggressive voltage underscaling and timing error resilience

for energy efficient deep neural network accelerators," *arXiv preprint arXiv:1802.03806*, 2018.

[82] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 267–278, IEEE Press, 2016.

[83] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "Softmc: A flexible and practical open-source infrastructure for enabling experimental dram studies," in *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 241–252, IEEE, 2017.

[84] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, pp. 2849–2858, 2016.

[85] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," *Online.*, 2017.

[86] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[87] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[88] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Understanding and exploiting design-induced latency variation in modern dram chips," *arXiv preprint arXiv:1610.09604*, 2016.

[89] M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (reaper): Enabling the mitigation of dram retention failures via profiling at aggressive conditions," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 255–268, 2017.

[90] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *ACM SIGARCH Computer architecture news*, vol. 41, pp. 475–486, ACM, 2013.

[91] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator.," *Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[92] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www. drampower. info*, vol. 22, 2012.

[93] A. Kozlov and D. Osokin, "Development of real-time adas object detector for deployment on cpu," *arXiv preprint arXiv:1811.05894*, 2018.

[94] J. Redmon, "Darknet: Open source neural networks in c," *http://pjreddie. com/darknet*, vol. 2016, 2013.

[95] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 163–174, IEEE, 2009.

[96] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pp. 1–12, IEEE, 2017.

[97] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator," in *ArXiV*, 2018.