# EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM

Skanda Koppula Lois Orosa A. Giray Yağlıkçı
Roknoddin Azizi Taha Shahroodi Konstantinos Kanellopoulos Onur Mutlu
ETH Zürich

## ABSTRACT

The effectiveness of deep neural networks (DNN) in vision, speech, and language processing has prompted a tremendous demand for energy-efficient high-performance DNN inference systems. Due to the increasing memory intensity of most DNN workloads, main memory can dominate the system's energy consumption and stall time. One effective way to reduce the energy consumption and increase the performance of DNN inference systems is by using approximate memory, which operates with reduced supply voltage and reduced access latency parameters that violate standard specifications. Using approximate memory reduces reliability, leading to higher bit error rates. Fortunately, neural networks have an intrinsic capacity to tolerate increased bit errors. This can enable energy-efficient and high-performance neural network inference using approximate DRAM devices.

Based on this observation, we propose EDEN, the first general framework that reduces DNN energy consumption and DNN evaluation latency by using approximate DRAM devices, while strictly meeting a user-specified target DNN accuracy. EDEN relies on two key ideas: 1) retraining the DNN for a target approximate DRAM device to increase the DNN's error tolerance, and 2) efficient mapping of the error tolerance of each individual DNN data type to a corresponding approximate DRAM partition in a way that meets the user-specified DNN accuracy requirements.

We evaluate EDEN on multi-core CPUs, GPUs, and DNN accelerators with error models obtained from real approximate DRAM devices. We show that EDEN's DNN retraining technique reliably improves the error resiliency of the DNN by an order of magnitude. For a target accuracy within 1% of the original DNN, our results show that EDEN enables 1) an average DRAM energy reduction of 21%, 37%, 31%, and 32% in CPU, GPU, and two different DNN accelerator architectures, respectively, across a variety of state-of-the-art networks, and 2) an average (maximum) speedup of 8% (17%) and 2.7% (5.5%) in CPU and GPU architectures, respectively, when evaluating latency-bound neural networks.

## CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; • **Computer systems organization** → *Neural networks*; Special purpose systems; • **Hardware** → *Dynamic memory*.

## KEYWORDS

deep neural networks, error tolerance, energy efficiency, machine learning, DRAM, memory systems


*MICRO-52, October 12–16, 2019, Columbus, OH, USA*

ACM ISBN 978-1-4503-6938-1/19/10.
https://doi.org/10.1145/3352460.3358280

**ACM Reference Format:**
Skanda Koppula, Lois Orosa, A. Giray Yağlıkçı, 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 (MICRO-52)*. Columbus, OH, USA, October 12-16, 2019, 16 pages. https://doi.org/10.1145/3352460.3358280

## 1 INTRODUCTION

Deep neural networks (DNNs) [87, 90] are an effective solution to challenges in computer vision, speech recognition, language translation, drug discovery, robotics, particle physics, and a number of other domains [9, 22, 41, 49, 50, 68, 84, 87, 102, 137, 149]. DNNs and their various flavors (convolutional neural networks [84], fully-connected neural networks [88], and recurrent neural networks [49]) are commonly evaluated in settings with edge devices that demand low energy and real-time responses [13, 146]. Unfortunately, DNNs have high computational and memory demands that make these energy and performance requirements difficult to fulfill. As such, neural networks have been the subject of many recent accelerators and DNN-focused architectures. Recent works (e.g., [6–8, 14, 25, 26, 29, 52, 54, 70, 100, 126, 152, 154, 158, 181]) focus on building specialized architectures for efficient computation scheduling and dataflow to execute DNNs [86, 109].

Improvements to accelerator efficiency [27, 85], DNN-optimized GPU kernels [23, 28], and libraries designed to efficiently leverage instruction set extensions [23, 83] have improved the computational efficiency of DNN evaluation. However, improving the memory efficiency of DNN evaluation is an on-going challenge [13, 29, 37, 54, 107, 148]. The memory intensity of DNN inference is increasing, and the sizes of state-of-art DNNs have grown dramatically in recent years. The winning model of the 2017 ILSVRC image recognition challenge [140], ResNeXt, contains 837M FP32 parameters (3.3 GB) [171]. This is 13.5x the parameter count of AlexNet, the winning model in 2012 [84]. More recent models have broken the one billion FP32 parameter mark (3.7 GB) [153]. As the machine learning community trends towards larger, more expressive neural networks, we expect off-chip memory problems to bottleneck DNN evaluation.

The focus of this work is to alleviate two main issues (energy and latency) of off-chip DRAM for neural network workloads. First, DRAM has high energy consumption. Prior works on DNN accelerators report that between 30 to 80% of system energy is consumed by DRAM [26, 99, 113, 130]. Second, DRAM has high latency. A load or store that misses the last level cache (LLC) can take 100x longer time to service compared to an L1 cache hit [32, 43, 44, 96, 114–116, 159]. Prior work in accelerator design has targeted DRAM latency as a challenge for sparse and irregular DNN inference [165].

To overcome both DRAM energy and latency issues, recent works use three main approaches. First, some works reduce numeric bitwidth, reuse model weights, and use other algorithmic strategies to reduce the memory requirements of the DNN workload [24, 48, 55, 60, 64, 66, 82, 103, 117, 146, 172, 175, 176, 184]. Second, other works propose new DRAM designs that offer lower en-

ergy and latency than commodity DRAM [19–21, 57, 91, 93, 94, 168]. Third, some works propose processing-in-memory approaches that can reduce data movement and access data with lower latency and energy [13, 29, 40, 45, 65, 101, 107, 148, 150, 152]. In this work, we propose an approach that is orthogonal to these existing works: we customize the major operational parameters (e.g., voltage, latency) of *existing* DRAM chips to better suit the intrinsic characteristics of a DNN. Our approach is based on two key insights:

(1) DNNs demonstrate remarkable robustness to errors introduced in input, weight, and output data types. This error tolerance allows accurate DNN evaluation on unreliable hardware if the DNN error tolerance is accurately characterized and bit error rates are appropriately controlled.
(2) DRAM manufacturers trade performance for reliability. Prior works show that reducing DRAM supply voltage and timing parameters improves the DRAM energy consumption and latency, respectively, at the cost of reduced reliability, i.e., increased bit error rate [15, 19, 21, 47, 94].

To exploit these two insights, we propose EDEN[1]: the first framework that improves energy efficiency and performance for DNN inference by using approximate DRAM, which operates with reduced DRAM parameters (e.g., voltage and latency). EDEN strictly meets a user-specified target DNN accuracy by providing a general framework that 1) uses a new retraining mechanism to improve the accuracy of a DNN when executed on approximate DRAM, and 2) maps the DNN to the approximate DRAM using information obtained from rigorous characterizations of the DNN error tolerance and DRAM error properties.

EDEN is based on three key steps. First, EDEN *improves the error tolerance* of the target DNN by retraining the DNN using the error characteristics of the approximate DRAM module. Second, EDEN *profiles* the improved DNN to identify the error tolerance levels of all DNN data (e.g., different layer weights of the DNN). Third, EDEN *maps* different DNN data to different DRAM partitions that best fit each datum's characteristics, and accordingly selects the voltage and latency parameters to operate each DRAM partition. By applying these three steps, EDEN can map an arbitrary DNN workload to an arbitrary approximate DRAM module to evaluate a DNN with low energy, high performance, *and* high accuracy.

To show example benefits of our approach, we use EDEN to run inference on DNNs using approximate DRAM with 1) reduced DRAM supply voltage ($V_{DD}$) to decrease DRAM energy consumption, and 2) reduced DRAM latency to reduce the execution time of latency-bound DNNs. EDEN adjusts the DRAM supply voltage and DRAM latency through interaction with the memory controller firmware. For a target accuracy within 1% of the original DNN, our results show that EDEN enables 1) an average DRAM energy reduction of 32% across CPU, GPU and DNN accelerator (e.g., Tensor Processing Unit [69]) architectures, and 2) cycle reductions of up to 17% when evaluating latency-bound neural networks.

Our evaluation indicates that the larger benefits of EDEN would stem from its capacity to run on most hardware platforms in use today for neural network inference, including CPUs, GPUs, FPGAs, and DNN accelerators. Because EDEN is a general approach, its principles can be applied 1) on any platform that uses DRAM, and 2) across memory technologies that can trade-off different parameters (e.g., voltage, latency) at the expense of reliability. Although our evaluation examines supply voltage and access latency reductions, the EDEN framework can be used also to improve performance and energy in other ways: for example, EDEN could increase the effective memory bandwidth by increasing the data bus frequency at the expense of reliability.

This paper makes the following five key contributions:

- We introduce EDEN, the first general framework that increases the energy efficiency and performance of DNN inference by using approximate DRAM that operates with reduced voltage and latency parameters at the expense of reliability. EDEN provides a systematic way to scale main memory parameters (e.g., supply voltage and latencies) while achieving a user-specified DNN accuracy target.
- We introduce a methodology to retain DNN accuracy in the presence of approximate DRAM. Our evaluation shows that EDEN increases the bit error tolerance of a DNN by 5-10x (depending on the network) through a customized retraining procedure called *curricular retraining*.
- We provide a systematic, empirical characterization of the resiliency of state-of-art DNN workloads [63, 64, 146] to the errors introduced by approximate DRAM. We examine error resiliency across different numeric precisions, pruning levels, and data types (e.g. DNN layer weights). We find that 1) lower precision levels and DNN data closer to the first and last layers exhibit lower error resiliency, and 2) magnitude-based pruning does not have a significant impact on error resiliency.
- We propose four error models to represent the common error patterns that an approximate DRAM device exhibits. To do so, we characterize the bit flip distributions that are caused by reduced voltage and latency parameters on eight real DDR4 DRAM modules.
- We evaluate EDEN on multi-core CPUs, GPUs, and DNN accelerators. For a target accuracy within 1% of the original DNN, our results show that EDEN enables 1) an average DRAM energy reduction of 21%, 37%, 31%, and 32% in CPU, GPU, and two different DNN accelerator architectures, respectively, across a variety of state-of-the-art networks, and 2) an average (maximum) speedup of 8% (17%) and 2.7% (5.5%) in CPU and GPU architectures, respectively, when evaluating latency-bound neural networks. For a target accuracy the same as the original, EDEN enables 16% average energy savings and 4% average speedup in CPU architectures.

## 2 BACKGROUND

### 2.1 Deep Neural Networks

A deep neural network (DNN) is a neural network with more than two layers [87]. DNNs are composed of a variety of different layers, including convolutional layers, fully-connected layers, and pooling layers. Figure 1 shows the three main data types of a DNN layer, and how three DNN layers are connected with each other. Each of these layers is defined by a weight matrix learned via a one-time training process that is executed before the DNN is ready for inference. The three DNN data types that require loads and stores from main memory include each layer's input feature maps (IFMs), output feature maps (OFMs), and the weights. Each layer processes its IFMs using the layer's weights, and produces OFMs. The OFMs of a layer are fed to the next layer as the next layer's IFMs. In this work, we explore the introduction of bit errors into the three data types of each layer.

Modern DNNs contain hundreds of layers, providing the DNN with a large number of trainable weights. The existence of such a large number of weights is commonly referred to as *overparameterization*, and is, in part, the source of a DNN's accuracy [42].

[1] **E**nergy-Efficient **De**ep **N**eural Network Inference Using Approximate DRAM

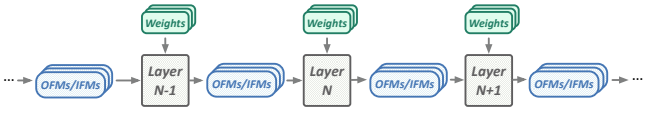


**Figure 1: Example of three DNN layers. Each layer is composed of its weights, input feature maps (IFMs), and output feature maps (OFMs).**

Overparameterization allows the model to have sufficient learning capacity so that the network can approximate complex input-output functions, and adequately capture high-level semantics (e.g., the characteristics of a cat in an input image) [120]. Importantly, overparameterization allows the network to obtain some level of error resilience, generalize across different inputs, and be robust to insignificant changes to the input (e.g., background pixels in an image) [123]. Common training-time techniques such as *adding white noise* and *input feature map dropout* try to force the network to *not* rely on any single OFM element and enable robustness in the presence of statistical variance in the IFMs [160]. In this work, we show that we can also adapt DNNs and their training procedure to achieve partial error robustness against bit errors caused by approximate DRAM, by fundamentally taking advantage of the overparameterization in the DNN.

**Quantization**. Quantizing floating-point weights and OFMs into low-precision fixed-point numbers can greatly improve performance and energy consumption of DNNs [62]. Many prior works demonstrate that it is possible to quantize DNNs to limited numeric precision (e.g., eight-bit integers) without significantly affecting DNN accuracy [33, 55, 62, 66, 103, 165, 170, 184]. In our evaluations, we quantize all DNN models to four different numeric precisions : int4 (4-bit), int8 (8-bit), int16 (16-bit), and FP32 (32-bit).

**Pruning**. Pruning [34] reduces the memory footprint of a DNN by sparsifying the weights and feature maps. This is done by zeroing the lowest magnitude weights and retraining [55, 98, 177]. We study the effects of pruning in our evaluations.

**Training**. Training is the process of estimating the best set of weights that maximize the accuracy of DNN inference. Training is usually performed with an iterative gradient descent algorithm [139] using a particular training dataset. The training dataset is divided into batches. One iteration is the number of batches needed to complete one epoch. One epoch completes when the entire dataset is passed once through the training algorithm.

## 2.2 DRAM Organization and Operation

**DRAM Organization**. A DRAM device is organized hierarchically. Figure 2a shows a *DRAM cell* that consists of a *capacitor* and an *access transistor*. A capacitor encodes a bit value with its charge level. The DRAM cell capacitor is connected to a *bitline* via an access transistor that is controlled by a *wordline*. Figure 2b shows how the DRAM cells are organized in the form of a 2D array (i.e., a *subarray*). Cells in a column of a subarray share a single bitline. Turning on an access transistor causes charge sharing between the capacitor and the bitline, which shifts the bitline voltage up or down based on the charge level of the cell's capacitor. Each bitline is connected to a *sense amplifier* (SA) circuit that detects this shift and amplifies it to a full 0 or 1. The cells that share the same wordline in a subarray are referred to as a DRAM *row*. A *row decoder* drives a *wordline* to enable all cells in a DRAM row. Therefore, charge sharing and sense amplification operate at row granularity. The array of sense amplifiers in a subarray is referred to as *row buffer*. Each subarray

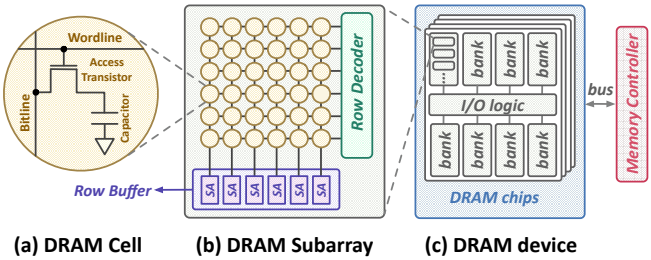


**Figure 2: DRAM organization.**

typically consists of 512-1024 rows each of which is typically as large as 2-8KB.

Figure 2c shows the organization of subarrays, banks, and chips that form a *DRAM device*. Each bank partially decodes a given row address and selects the corresponding subarray's *row buffer*. On a read operation, the I/O logic sends the requested portion of the target row from the corresponding subarray's row buffer to the memory controller. A *DRAM chip* contains multiple banks that can operate in parallel. A DRAM device is composed of multiple DRAM chips that share the same command/address bus and are simultaneously accessed to provide high bandwidth and capacity. In a typical system, each memory controller interfaces with a single DRAM bus. We refer the readers to [17, 19–21, 56, 57, 73, 79, 91, 93, 94, 105, 106, 151, 162, 168, 182] for more detail on DRAM structure and design.

**DRAM Operation**. Accessing data stored in each row follows the sequence of memory controller commands illustrated in Figure 3. First, the activation command (ACT) activates the row by pulling up the wordline and enabling sense amplification. After a manufacturer-specified $t_{RCD}$ nanoseconds, the data is reliably sensed and amplified in the *row buffer*. Second, the read command (READ) reads the data from the row buffer to the IO circuitry. After a manufacturer-specified $CL$ nanoseconds, the data is available on the memory bus. Third, the precharge command (PRE) prepares the DRAM bank for activation of another row. A precharge command can be issued a manufacturer-specified $t_{RAS}$ nanoseconds after an activation command, and an activation command can be issued $t_{RP}$ nanoseconds after a precharge command. $t_{RCD}$, $t_{RAS}$, $t_{RP}$, and $CL$ are examples of DRAM timing parameters and their nominal values provided in DRAM DDR4 datasheets are 12.5ns, 32ns, 12.5ns, and 12.5ns respectively [67].

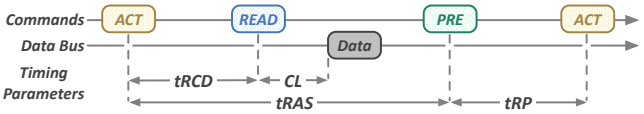


**Figure 3: DRAM read timing. We explore reductions of $t_{RCD}$, $t_{RAS}$, and $t_{RP}$ as part of EDEN's evaluation. $CL$ is a characteristic of the device, and not adjustable in the memory controller [67].**

## 2.3 Reducing DRAM Parameters

We build on a large body of work on characterizing DRAM behavior in sub-reliable operation regimes of *supply voltage* and *latency* parameters [15, 18, 19, 21, 58, 76–78, 93, 94, 105, 128].

**DRAM Voltage Reduction**. Voltage reduction is critical to reducing DRAM power consumption since power is proportional to the square of supply voltage (i.e., $V_{DD}{}^2 \times f$) [112, 167]. Prior research [21, 36] shows that reducing voltage increases the propagation delay of signals, which can cause errors when using unmodified

timing parameters. One work avoids these errors by increasing the $t_{RCD}$ and $t_{RP}$ latencies [21] to ensure *reliable* operation. In contrast, our goal in this work is to aggressively reduce power consumption and latency by decreasing both supply voltage and timing parameters, which inevitably causes errors in the form of bit flips in the weakest cells of DRAM, making DRAM approximate. Resulting error patterns often exhibit locality. Chang et al. [21] observe that these bit flips accumulate in certain regions (e.g., banks and rows) of DRAM.

**DRAM Access Latency Reduction**. Latency reduction is critical to increase system performance, as heavily emphasized by a recent study on workload-DRAM interactions [46]. Previous works characterize real DRAM devices to find the minimum reliable *row activation* ($t_{RCD}$) and *precharge* ($t_{RP}$) latency values [18, 19, 76, 93, 94]. According to these studies, the minimum DRAM latency values are significantly smaller than the values that datasheets report, due to conservative guardbands introduced by DRAM manufacturers. Further reducing these latency values cause bit flips in weak or unstable DRAM cells.

**DRAM Refresh Rate Reduction**. Other than voltage and latency, previous research also shows that reducing the refresh rate of DRAM chips both can increase performance and reduce energy consumption at the cost of introducing errors [35, 58, 72, 74, 75, 105, 106, 128, 129] that are tolerable by many workloads that can tolerate bit errors [5, 71, 122, 183].

## 3 EDEN FRAMEWORK

To efficiently solve the energy and latency issues of off-chip DRAM for neural network workloads, we propose EDEN. EDEN is the first general framework that improves energy efficiency and performance for neural network inference by using approximate DRAM. EDEN is based on two main insights: 1) neural networks are tolerant to errors, and 2) DRAM timing parameters and voltage can be reduced at the cost of introducing more bit errors.

We first provide an overview of EDEN in Section 3.1, and explain EDEN's three steps in Sections 3.2, 3.3, and 3.4. Finally, Section 3.5 explains the changes required by the target DNN inference system to support a DNN generated by EDEN.

### 3.1 EDEN: A High Level Overview

EDEN enables the effective execution of DNN workloads using approximate DRAM through three key steps: 1) boosting DNN error tolerance, 2) DNN error tolerance characterization, and 3) DNN-DRAM mapping. These steps are repeated iteratively until EDEN finds the most aggressive DNN and DRAM configuration that meets the target accuracy requirements. EDEN transforms a DNN that is trained on reliable hardware into a device-tuned DNN that is able to run on a system that uses approximate DRAM at a target accuracy level. EDEN allows tight control of the trade-off between accuracy and performance by enabling the user/system to specify the maximal tolerable accuracy degradation. Figure 4 provides an overview of the three steps of EDEN, which we describe next.

**1. Boosting DNN Error Tolerance**. EDEN introduces *curricular retraining*, a new retraining mechanism that boosts a DNN's error tolerance for a target approximate DRAM module. Our curricular retraining mechanism uses the error characteristics of the target approximate DRAM to inject errors into the DNN training procedure and boost the DNN accuracy. The key novelty of curricular retraining is to inject errors at a progressive rate during the training process with the goal of increasing DNN error tolerance while avoiding accuracy collapse with error correction. EDEN boosts

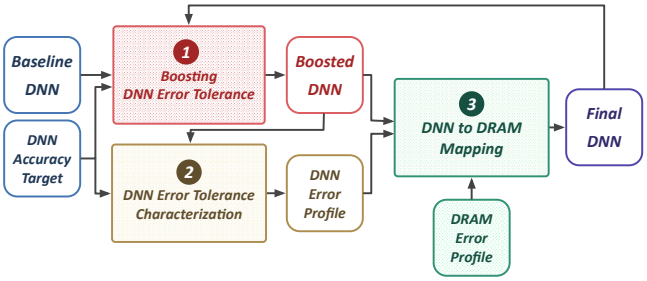


**Figure 4: Overview of the EDEN framework.**

the intrinsic bit error tolerance of the baseline DNN by 5-10x. We describe our boosting mechanism in Section 3.2.

**2. DNN Error Tolerance Characterization**. EDEN characterizes the error resilience of each *boosted DNN data type* (i.e., IFMs, OFMs, and DNN weights) to identify the limits of bit error tolerance. EDEN measures the effect of bit errors on overall accuracy using the DNN validation dataset. We describe error tolerance characterization in Section 3.3.

**3. DNN to DRAM Mapping**. EDEN maps the error tolerance of each DNN data type to a corresponding approximate DRAM partition (e.g., chip, bank, or subarray) in a way that meets the specified accuracy requirements, while maximizing performance. We describe DNN to DRAM mapping in Section 3.4.

Together, the three steps of EDEN enable a baseline DNN to become a specialized DNN that is error-tolerant and device-tuned to a target approximate DRAM. EDEN enables energy efficient, high-performance DNN inference on the target approximate DRAM with a user-defined accuracy.

### 3.2 Boosting DNN Error Tolerance

According to our evaluations, the error tolerance of common DNNs is not sufficient to enable significant DRAM voltage and timing parameter reductions. To overcome this issue, we propose *curricular retraining*, a new retraining mechanism that improves the error tolerance of a DNN when running with approximate DRAM that injects errors into memory locations accessed by the DNN.

The key idea of curricular retraining is based on the observation that introducing high error rates immediately at the beginning of retraining process occasionally causes training divergence and a phenomenon called *accuracy collapse*. To mitigate this problem, *curricular retraining* slowly increases the error rate of the approximate DRAM from 0 to a target value in a step-wise fashion. In our experiments, we observe a good training convergence rate when we increase the error rate every two epochs (i.e., two passes of the entire training dataset). EDEN uses approximate DRAM in the forward pass, and it uses reliable DRAM for the backward pass.

We demonstrate in Section 6.4 that our curricular retraining mechanism is effective at improving the accuracy of DNN inference executed on systems with approximate DRAM.

Our experiments show that curricular retraining does not help to improve DNN accuracy on *reliable* DRAM. This implies that introducing bit error is not a regularization technique,[2] but rather, a way of obtaining congruence between the DNN training algorithms and the errors injected by approximate DRAM.

**Correcting Implausible Values**. While executing curricular retraining, a single bit error in the exponent bits of a floating point

[2]Regularization is a technique that makes slight modifications to the training algorithm such that the DNN model generalizes better.

value can cause *accuracy collapse* in the trained DNN. For example, a bit error in the exponent of a weight creates an enormously large value (e.g., $>10^8$) that propagates through the DNN layers, dominating weights that are significantly smaller (e.g., <10).

To avoid this issue, we propose a mechanism to avoid accuracy collapse caused by bit errors introduced by approximate DRAM. The key idea of our mechanism is to correct the values that are *implausible*. When a value is loaded from memory, our mechanism probabilistically detects that a data type likely contains an error by comparing its value against predefined thresholds. The thresholds of the curricular retraining data types are computed during training of the baseline DNN on DRAM with nominal parameters. Those thresholds usually have rather small values (e.g., most weights in SqueezeNet1.1 are within the range [-5,5]).

Upon detection of an error (i.e, the fact that a value is out of the threshold range) during curricular retraining, EDEN 1) corrects the erroneous value by zeroing the value, and 2) uses the corrected value for curricular retraining.

Our mechanism for correcting implausible values can be implemented in two ways. First, a software implementation that modifies the DNN framework to include extra instructions that correct implausible values resulting from each DNN memory access. Second, a hardware implementation that adds a simple hardware logic to the memory controller that corrects implausible values resulting from each approximate DRAM memory request. Section 5 describes our low cost hardware implementation.

In our experiments, we find that our mechanism for correcting implausible values increases the tolerable bit error rate from $10^{-7}$ to $10^{-3}$ to achieve <1% accuracy degradation in the eight FP32 DNNs we analyze. We evaluate an alternative mechanism for error correction that saturates an out-of-threshold value (by resetting to the closest threshold value) instead of zeroing it. We observe that saturating obtains lower DNN accuracy than zeroing at the same approximate DRAM bit error rate across all DNN models (e.g., 8% on CIFAR-10 and 7% on ImageNet). We also correct implausible values during the execution of DNN inference to improve the inference accuracy (Section 3.5).

## 3.3 DNN Error Tolerance Characterization

EDEN aims to guarantee that the accuracy of a DNN meets the minimum value required by the user. To this end, EDEN characterizes the boosted DNN (obtained from our boosting mechanism in Section 3.2) to find the maximum tolerable bit error rate (BER) by progressively decreasing the approximate DRAM parameters, i.e., voltage and latency. EDEN performs either a *coarse-grained* or a *fine-grained* DNN error tolerance characterization.

**Coarse-Grained Characterization**. EDEN's coarse-grained characterization determines the highest BER that can be applied uniformly to the entire DNN, while meeting the accuracy requirements of the user. This characterization is useful for mapping the DNN to commodity systems (see Section 3.4) that apply reduced DRAM parameters to an entire DRAM module (without fine-grained control).

To find the highest BER that satisfies the accuracy goal, our coarse-grained characterization method performs a logarithmic-scale binary search on the error rates. We can use binary search because we found that DNN error-tolerance curves are monotonically decreasing. To adjust the BER while doing this characterization, EDEN can either 1) tune the parameters of approximate DRAM, or 2) use DRAM error models for injecting bit errors into memory locations (see Section 4). EDEN optimizes the error resiliency of a DNN by repeating cycles of DNN error tolerance boosting (Section 3.2), coarse-grained DNN characterization, and DNN to DRAM mapping (Section 3.4) until the highest tolerable BER stops improving. We evaluate our coarse-grained characterization mechanism in Section 6.5.

**Fine-Grained Characterization**. EDEN can exploit variation in the error tolerances of different DNN data types by clustering the data according to its error tolerance level, and assigning each cluster to a different DRAM partition whose error rate matches the error tolerance level of the cluster (see Section 3.4). For example, we find that the first and the last convolutional layers have tolerable BERs 2-3x smaller than the average middle layer in a DNN (in agreement with prior work [166, 180]).

To conduct a fine-grained DNN characterization, EDEN searches for the highest tolerable BER of each weight and IFM that still yields an acceptable DNN accuracy. This search space is exponential with respect to the DNN's layer count. To tackle the search space challenge, EDEN employs a DNN data sweep procedure that performs iterations over a list of DNN data types. The mechanism tries to increase the tolerable error rate of a data type by a small amount, and tests if the DNN still meets the accuracy requirements. When a DNN data type cannot tolerate more increase in error rate, it is removed from the sweep list. We evaluate our fine-grained characterization mechanism in Section 6.6.

**Effect of Pruning**. EDEN does not include pruning (Section 2.1) as part of its boosting routine due to two observations. First, we find that DNN sparsification does not improve the error tolerance. Our experiments show that when we create 10%, 50%, 75%, and 90% sparsity through energy-aware pruning [175], error tolerance of FP32 and int8 DNNs, DNN error tolerance does not improve significantly. Second, the zero values in the network, which increase with pruning, are sensitive to memory error perturbations.

## 3.4 DNN to DRAM Mapping

After characterizing the error tolerance of each DNN data type, EDEN maps each data type to the appropriate DRAM partition (with the appropriate voltage and latency parameters) that satisfies the data type's error tolerance. Our mechanism aims to map a data type that is very tolerant (intolerant) to errors into a DRAM partition with the highest (lowest) BER, matching the error tolerance of the DNN and the BER of the DRAM partition as much as possible.

**DRAM Bit Error Rate Characterization**. To obtain the BER characteristics of a DRAM device (both in aggregate and for each partition), we perform reduced voltage and reduced latency tests for a number of data patterns. For each voltage level, we iteratively test two consecutive rows at a time. We populate these rows with inverted data patterns for the worst-case evaluation. Then, we read each bit with reduced timing parameters (e.g., tRCD). This characterization requires fine-grained control of the DRAM timing parameters and supply voltage level. EDEN's characterization mechanism is very similar to experimental DRAM characterization mechanisms proposed and evaluated in prior works for DRAM voltage [21, 47] and DRAM latency [18, 19, 58, 76, 93, 94].

**Coarse-grained DNN to DRAM module mapping**. All DNN data types stored within the same DRAM module are exposed to the *same* DRAM voltage level and timing parameters. These parameters are tuned to produce a bit error rate that is tolerable by *all* DNN data types that are mapped to the module.

Under coarse-grained mapping, the application does *not* need to be modified. Algorithms used in DNN inference are oblivious to

the DRAM mapping used by the memory controller. The memory controller maps all inference-related requests to the appropriate approximate DRAM module. Data that cannot tolerate bit errors at any reduced voltage and latency levels is stored in a separate DRAM module whose voltage and latency parameters follow the manufacturer specifications.

Coarse-grained mapping can be easily supported by existing systems that allow the modification of $V_{dd}$ and/or $t_{RCD/RP}$ parameters in the BIOS across the entire DRAM module. Section 5 describes the simple hardware changes required to support coarse-grained mapping. We evaluate our coarse-grained mapping mechanism in Section 6.5.

**Fine-grained DNN to DRAM module mapping**. DNN data types stored in different DRAM partitions can be exposed to *different* DRAM voltage levels and/or timing parameters. DRAM can be partitioned at chip, rank, bank, or subarray level granularities. Algorithm 1 describes our algorithm for fine-grained mapping of DNN data to DRAM partitions. Our algorithm uses rigorous DRAM characterization and DNN characterization to iteratively assign DNN data to DRAM partitions in three basic steps. First, our mechanism looks for DRAM partitions that have BERs lower than the tolerable BER of a given DNN data type. Second, we select the DRAM partition with the largest parameter reduction that meets the BER requirements. Third, if the partition has enough space available, our mechanism assigns the DNN data type to the DRAM partition. We evaluate our fine-grained mapping mechanism in Section 6.6.

**Algorithm 1** Fine-grained DNN to DRAM mapping

```
function DNN_to_DRAM_Mapping(DNN_characterization,
    DRAM_characterization):
  sorted_data = sort_DNN_data(DNN_characterization)
  for (target_BER, DNN_data) in sorted_data:
    # Find the DRAM partition that has the least
      voltage/latency at target_BER, and can fit
      the DNN_data
    for DRAM_partition in DRAM_characterization
      partition_params =
        get_voltage_latency(DRAM_partition,
        target_BER)
      if DNN_data.size < DRAM_partition.size :
        if partition_params < best_parameters:
          best_parameters = partition_params
          chosen_partition = DRAM_partition
          DRAM_partition.size -= DNN_data.size
    final_mapping[chosen_partition].append(DNN_data)
  return final_mapping
```

A system that supports fine-grained mapping requires changes in the memory controller (for voltage and latency adjustment) and in DRAM (for only voltage adjustment). We describe the hardware changes required to support fine-grained mapping in Section 5.

## 3.5 DNN Inference with Approximate DRAM

EDEN generates a boosted DNN for running inference in a target system that uses approximate DRAM. EDEN does not require any modifications in DNN inference hardware, framework, or algorithm, except for *correcting implausible values*. Similar to what happens in our curricular retraining (Section 3.2), a single bit error in the exponent bits of a floating point value can cause *accuracy collapse* during DNN inference. We use the same mechanism for correcting implausible values in our curricular retraining mechanism (i.e., we zero the values that are outside of a predefined threshold range) to avoid accuracy collapse caused by bit errors introduced by approximate DRAM during DNN inference.

# 4 ENABLING EDEN WITH ERROR MODELS

EDEN requires extensive characterization of the target approximate DRAM device for boosting DNN error tolerance (Section 3.2), characterization of DNN error tolerance (Section 3.3), and mapping of the DNN to the approximate DRAM device (Section 3.4). However, applying EDEN in a target system where DNN inference can be performed is not always feasible or practical. For example, a low-cost DNN inference accelerator [26] might perform very slowly when executing our curricular retraining mechanism, because it is *not* optimized for training. Similarly, the target hardware might not be available, or might have very limited availability (e.g., in the pre-production phase of a new approximate hardware design).

To solve this problem and enable EDEN even when target DRAM devices are not available for characterization, we propose to execute the EDEN framework in a system that is different from the target approximate system. We call this idea *EDEN offloading*. The main challenge of offloading EDEN to a different system is how to faithfully emulate the errors injected by the *target* approximate DRAM into the DNN. To address this challenge, we use four different error models that are representative of most of the error patterns that are observed in real approximate DRAM modules.

**EDEN's DRAM Error Models**. EDEN uses four probabilistic error models that closely fit the error patterns observed in a real approximate DRAM module. Our models contain information about the location of weak cells in the DRAM module, which is used to decide the spatial distribution of bit errors during DNN error tolerance boosting. We create four different types of error models from the data we obtain based on our characterization of existing DRAM devices using SoftMC [58] and a variety of DDR3 and DDR4 DRAM modules. Our error models are consistent with the error patterns observed by prior works [19, 21, 76–78, 94]. In addition, our error models are parameterizable and can be tuned to model individual DRAM chips, ranks, banks, and subarrays from different vendors.

- **Error Model 0:** the bit errors follow a uniform random distribution across a DRAM bank. Several prior works observe that reducing activation latency ($t_{RCD}$) and precharge latency ($t_{RP}$) can cause randomly distributed bit flips due to manufacturing process variation at the level of DRAM cells [15, 19, 93, 94]. We model these errors with two key parameters: 1) $P$ is the percentage of *weak cells* (i.e., cells that fail with reduced DRAM parameters), and 2) $F_A$ is the probability of an error in any weak cell. Such uniform random distributions are already observed in prior works [10, 53, 133, 164].
- **Error Model 1:** the bit errors follow a vertical distribution across the bitlines of a DRAM bank. Prior works [19, 21, 76, 93] observe that some bitlines experience more bit flips than others under reduced DRAM parameters due to: 1) manufacturing process variation across sense amplifiers [19, 21, 76], and 2) design-induced latency variation that arises from the varying distance between different bitlines and the row decoder [93]. We model this error distribution with two key parameters: 1) $P_B$ is the percentage of weak cells in bitline $B$, and 2) $F_B$ is the probability of an error in the weak cells of bitline $B$.

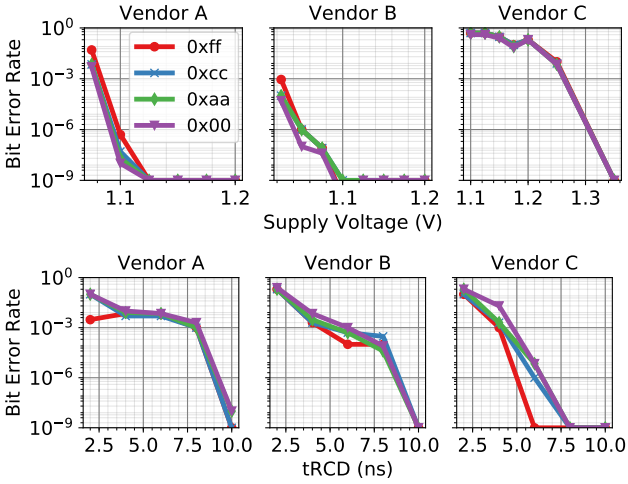


**Figure 5: Bit error rates depend on the data pattern stored in DRAM, with reduced supply voltage [21] and reduced $t_{RCD}$ [19, 21, 76, 93, 94], motivating Error Model 3. Data is based on DDR3 DRAM modules from three major vendors.**

- **Error Model 2:** the bit errors follow a horizontal distribution across the wordlines of a DRAM bank. Prior works [19, 21, 76, 93] observe that some DRAM rows experience more bit flips than others under reduced DRAM parameters due to 1) manufacturing process variation across DRAM rows [19, 21, 76], and 2) design-induced latency variation that arises from the varying distance between different DRAM rows and the row buffer [93]. We model this error distribution with two key parameters: 1) $P_W$ is the percentage of weak cells in wordline $W$, and 2) $F_W$ is the probability of an error in the weak cells of wordline $W$.
- **Error Model 3:** the bit errors follow a uniform random distribution that depends on the content of the cells (i.e., this is a data-dependent error model). Figure 5 illustrates how the bit error rates depend on the data pattern stored in DRAM, for reduced voltage (top) and reduced $t_{RCD}$ (bottom). We observe that 0-to-1 flips are more probable with $t_{RCD}$ scaling, and 1-to-0 flips are more probable with voltage scaling. Prior works provide rigorous analyses of data patterns in DRAM with reduced voltage [21] and timing parameters [19] that show results similar to ours. This error model has three key parameters: 1) $P$ is the percentage of weak cells, 2) $F_{V1}$ is the probability of an error in the weak cells that contain a 1 value, and 3) $F_{V0}$ is the probability of an error in the weak cells that contain a 0 value.

**Model Selection**. EDEN applies a maximum likelihood estimation (MLE) [128] procedure to determine 1) the parameters ($P$, $F_A$, $P_B$, $F_B$, $P_W$, $F_W$, $F_{V1}$ and $F_{V0}$) of each error model, and 2) the error model that is most likely to produce the errors observed in the real approximate DRAM chip. In case two models have very similar probability of producing the observed errors, our selection mechanism chooses Error Model 0 if possible, or one of the error models randomly otherwise. Our selection mechanism favors Error Model 0 because we find that it the is error model that performs better. We observe that generating and injecting errors by software with Error Model 0 in both DNN retraining and inference is 1.3x faster than injecting errors with other error models in our experimental setup. We observe that Error Model 0 provides 1) a reasonable approximation of Error Model 1, if $max(F_B) - min(F_B) < 0.05$ and $P_B \approx P$, and 2) a reasonable approximation of Error Model 2, if $max(F_W) - min(F_W) < 0.05$ and $P_W \approx P$.

**Handling Error Variations**. Error rates and error patterns depend on two types of factors. First, factors intrinsic to the DRAM device. The most common intrinsic factors are caused by manufacturer [21, 105], chip, and bank variability [77, 94]. Intrinsic factors are established at DRAM fabrication time. Second, factors extrinsic to the DRAM device that depend on environmental or operating conditions. The most common extrinsic factors are aging [111, 147], data values [74], and temperature [51]. Extrinsic factors can introduce significant variability in the error patterns.

EDEN can capture intrinsic factors in the error model with a unique DRAM characterization pass. However, capturing extrinsic factors in the error model is more challenging. Our DNN models capture three factors extrinsic to the DRAM device.

First, EDEN can capture data dependent errors by generating different error models for different DNN models (i.e., different IFM and weight values in memory). For each DNN model, EDEN stores the actual weight and IFM values in the target approximate DRAM before characterization to capture data dependencies.

Second, EDEN can capture temperature variations by generating different error models for the same approximate DRAM operating at different temperatures. Errors increase with higher temperatures [94, 105], so the model must match the temperature of DNN inference execution.

Third, EDEN can capture DRAM aging by periodically regenerating new error models. In our experiments with real DRAM modules, we find that the errors are temporally consistent and stable for days of continuous execution (with ±5°C deviations from the profiling temperature), without requiring re-characterization. Prior works [76, 94] report similar results.

We find in our evaluation that our error models are sufficiently expressive to generate a boosted DNN that executes on real approximate DRAM with minimal accuracy loss (Section 6.4). Our four error models are also sufficiently expressive to encompass the bit-error models proposed in prior works [12, 128].

## 5 MEMORY CONTROLLER SUPPORT

To obtain the most out of EDEN, we modify the memory controller to 1) correct implausible values during both curricular retraining and DNN inference, 2) support coarse-grained memory mapping, and 3) support fine-grained memory mapping.

**Hardware Support for Correcting Implausible Values**. We correct implausible values that cause accuracy collapse during both curricular retraining (Section 3.2) and DNN inference (Section 3.5). Our mechanism 1) compares a loaded value to an upper-bound and a lower-bound threshold, and 2) sets the value to zero (i.e., supplies the load with a zero result) in case the value is out of bounds. Because these operations are done for *every* memory access that loads a DNN value, it can cause significant performance degradation if performed in software. To mitigate this issue, we incorporate simple hardware logic in the memory controller that we call *bounding logic*. Our bounding logic 1) compares the exponent part of the loaded floating point value to DNN-specific upper-bound and lower-bound thresholds, and 2) zeros the input value if the value is out of bounds. In our implementation, the latency of this logic is only 1 cycle and its hardware cost is negligible.

**Enabling Coarse-Grained Mapping**. Coarse-grained mapping applies the same voltage and timing parameters to the entire DRAM for executing a particular DNN workload. However, different DNN

workloads might require applying different sets of DRAM parameters to maximize energy savings and performance. In many existing commodity systems, the memory controller sets the DRAM voltage and the timing parameters at start-up, and it is not possible to change them at runtime. To overcome this limitation, the memory controller requires minimal hardware support for changing the DRAM parameters of each DRAM module at runtime.

**Enabling Fine-Grained Mapping**. Fine-grained mapping applies different voltage and/or timing parameters to different DRAM partitions.

To apply different voltages to different memory partitions, EDEN 1) adopts the approach used by Voltron [21] to implement a robust design for voltage scaling at the bank granularity based on modest changes to the power delivery network, and 2) tracks which memory partition is operating at what voltage. To implement this mechanism in commodity DDR4/LPDDR4 chips with 16/32 banks, EDEN requires at most 32B of meta-data to represent all 8-bit voltage step values.

To apply different timing parameters to different memory partitions, EDEN requires memory controller support for 1) configuring the target memory partition to operate at specific timing parameters, and 2) tracking which memory partition is operating at what latency. For the timing parameter we tested in our evaluation ($t_{RCD}$), 4-bits are enough to encode all possible values of the parameter with enough resolution.

It is sufficient for EDEN to split DRAM into at most $2^{10}$ partitions, because most commonly used DNN architectures have at most 1024 different types of error-resilient IFMs and weights. EDEN requires 1KB of metadata to support $2^{10}$ partitions. To support mappings at subarray level granularity (i.e., the finest supported granularity), EDEN needs a larger amount of metadata. For example, for an 8GB DDR4 DRAM module with 2048 subarrays, EDEN needs to store 2KB of metadata.

# 6 DNN ACCURACY EVALUATION

In this section, we evaluate EDEN's ability to improve DNN accuracy in approximate DRAM. We explain our methodology (Section 6.1), evaluate the accuracy of our error models (Section 6.2), evaluate the error tolerance of the DNN baselines (Section 6.3), and analyze the accuracy of our curricular retraining mechanism (Section 6.4).

## 6.1 Methodology

We use an FPGA-based infrastructure running SoftMC [3, 58] to reduce DRAM voltage and timing parameters. SoftMC allows executing memory controller commands on individual banks, and modifying $t_{RCD}$ and other DRAM timing parameters. We perform all our experiments at room temperature. Using this infrastructure, we can obtain characteristics of real approximate DRAM devices. However, our infrastructure also has some performance limitations caused by delays introduced with SoftMC's FPGA buffering, host-FPGA data transmission, and instruction batching on the FPGA.

To overcome these performance limitations, we emulate real approximate DRAM modules by using the error models described in Section 4. To ensure that our evaluation is accurate, we validate our error models against real approximate DRAM devices (Section 6.2).

We incorporate EDEN's error models into DNN inference libraries by following the methodology described in Figure 6. We create a framework on top of PyTorch [127] that allows us to modify the loading of weights and IFMs. Our PyTorch implementation 1) injects errors into the original IFM and weight values using our DRAM error models, and 2) applies our mechanism to correct implausible values caused by bit errors in IFMs and weights (Section 3.2). Our DRAM error models are implemented as custom GPU kernels for efficient and simple integration into PyTorch. This simulation allows us to obtain DNN accuracy estimates 80-90x faster than with the SoftMC infrastructure.


**Figure 6: Methodology to incorporate DRAM error models in the DNN evaluation framework.**

**DNN Baselines**. We describe the DNN baselines that we use in the evaluation of the three EDEN steps (Sections 3.2, 3.3, and 3.4). Table 1 lists the eight modern and commonly-used DNN models we evaluate. We target both small (e.g., CIFAR-10 [4]) and large-scale (e.g., ILSVRC2012 [140]) image classification datasets. ResNet101 [59], VGG-16 [156], and DenseNet201 [63] models are top-five winners of past ImageNet ILSVRC competitions [84, 140]. We use Google MobileNetV2 [146] to test smaller, mobile-optimized networks that are widely used on mobile platforms, and SqueezeNet [64] to test embedded, real-time applications. Table 1 also shows the summed sizes of all IFMs and weights of each network for processing one input, which is a good indicator of the memory intensity of each DNN model.

| Model | Dataset | Model Size | IFM+Weight Size |
|---|---|---|---|
| ResNet101 [59] | CIFAR10 [4] | 163.0MB | 100.0MB |
| MobileNetV2 [146] | CIFAR10 [4] | 22.7MB | 68.5MB |
| VGG-16 [156] | ILSVRC2012 [140] | 528.0MB | 218.0MB |
| DenseNet201 [63] | ILSVRC2012 [140] | 76.0MB | 439.0MB |
| SqueezeNet1.1 [64] | ILSVRC2012 [140] | 4.8MB | 53.8MB |
| Alexnet [84] | CIFAR10 [4] | 233.0MB | 208.0MB |
| YOLO [137] | MSCOCO [104] | 237.0MB | 360.0MB |
| YOLO-Tiny [137] | MSCOCO [104] | 33.8MB | 51.3MB |
| LeNet$^\star$ [89] | CIFAR10 [4] | 1.65MB | 2.30MB |

$^\star$ we use this small model in some evaluations where the experimental setup does not support large models.

**Table 1: DNN models used in our evaluations. The listed total model size and summed IFM+weight sizes are for the FP32 variant of each model.**

Table 2 shows the accuracy we obtain in our experiments for our baseline networks across four different numeric precisions (int4, int8, int16 and FP32), using *reliable* commodity DRAM. We quantize using the popular symmetric linear DNN quantization scheme [103]. This quantization scheme applies weight-dependent affine scaling to linearly map weights into the range $[-2^{b-1},\ 2^{b-1}-1]$, where $b$ is the target model weight bit precision. YOLO and YOLO-Tiny's framework only support int8 and FP32 numeric precisions.

Our baseline accuracies match stated numbers in relevant literature [59, 63, 64, 146, 156]. Two of the models, DenseNet201 and SqueezeNet1.1, suffer from accuracy collapse at 4-bit precision. We did not use hyper-parameter tuning in our baselines or subsequent experiments. All results use the default DNN architectures and learning rates.

## 6.2 Accuracy Validation of the Error Models

EDEN uses errors obtained from real DRAM devices to build and select accurate error models. We profile the DRAM 1) before running

| Model | int4 | int8 | int16 | FP32 |
|---|---|---|---|---|
| ResNet101 [59] | 89.11% | 93.14% | 93.11% | 94.20% |
| MobileNetV2 [146] | 51.00% | 70.44% | 70.46% | 78.35% |
| VGG-16 [156] | 59.05% | 70.48% | 70.53% | 71.59% |
| DenseNet201 [63] | 0.31% | 74.60% | 74.82% | 76.90% |
| SqueezeNet1.1 [64] | 8.07% | 57.07% | 57.39% | 58.18% |
| Alexnet [84] | 83.13% | 86.04% | 87.21% | 89.13% |
| YOLO★ [137] | – | 44.60% | – | 55.30% |
| YOLO-Tiny★ [137] | – | 14.10% | – | 23.70% |
| LeNet [89] | – | 61.30% | – | 67.40% |

★ these models use mean average precision (mAP) instead of the accuracy metric.

**Table 2: Baseline accuracies of the networks used in our evaluation with reliable DRAM memory (no bit errors) using different numeric precisions.**

DNN inference, and 2) when the environmental factors that can affect the error patterns change (e.g., when temperature changes). We find that an error model can be accurate for many days if the environmental conditions do not change significantly, as also observed in prior work [76, 93, 94].

We derive our probabilistic error models (Section 4) from data obtained from eight real DRAM modules. We use the same FPGA infrastructure as the one described in Section 6.1. We find that complete profiling of a 16-bank, 4GB DDR4 DRAM module takes under 4 minutes in our evaluation setup. We can speed up the profiling time by 2-5x using more sophisticated DRAM profiling methodologies [129].

We validate our error models by comparing the DNN accuracy obtained after injecting bit errors using our DRAM error models to the accuracy obtained with each real approximate DRAM module. Figure 7 shows an example of the DNN accuracy obtained using DRAM modules from three major vendors with reduced voltage and $t_{RCD}$, and the DNN accuracy obtained using our Error Model 0. We use Error Model 0 because it is the model that fits better the errors observed in the three tested DRAM modules. Our main observation is that the DNN accuracy obtained with our model is very similar to that obtained with real approximate DRAM devices. We conclude that our error models mimic very well the errors observed in real approximate DRAM devices.

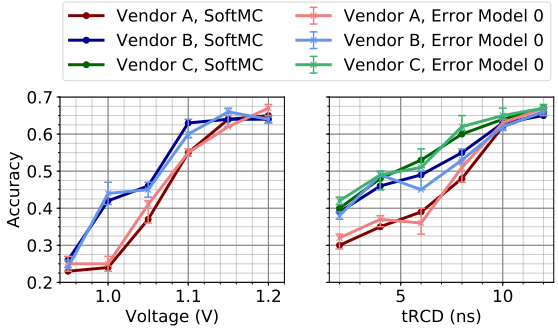


**Figure 7: LeNet/CIFAR-10 accuracies obtained using real approximate DRAM devices (via SoftMC) and using our Error Model 0. Error bars show the 95% confidence interval of Error Model 0.**

## 6.3 Error Tolerance of Baseline DNNs

To better understand the baseline error tolerance of each DNN (before boosting the error tolerance), we examine the error tolerance of the baseline DNNs. This also shows us how differences in quantization, best-fit error model, and BER can potentially affect the final DNN accuracy.

Figure 8 shows the accuracy of ResNet101 at different precision levels and BERs using all four error models. We see that all DNNs exhibit an accuracy drop at high BER ($> 10^{-2}$), but different error models cause the drop-off for all DNNs to be higher or lower. This is rooted in how each error model disperses bit errors into the DNN IFMs and weights. A good example of this is Error Model 1, which exhibits the most early and extreme drop-offs, especially for FP32 DNNs. We find that the cause of this is that, in our experimental setup, IFMs and weights are aligned in DRAM, so the MSBs of different DNN data types are mapped to the same bitline B. If the percentage of weak cells in bitline B ($P_B$) is high, the DNN suffers many MSB failures. However, Error Model 0 distributes these weak cell failures uniformly and randomly across the bank, causing far fewer MSB failures. In general, the way in which each error model captures the distribution of weak cells across data layout in memory greatly affects its impact on the error curve.

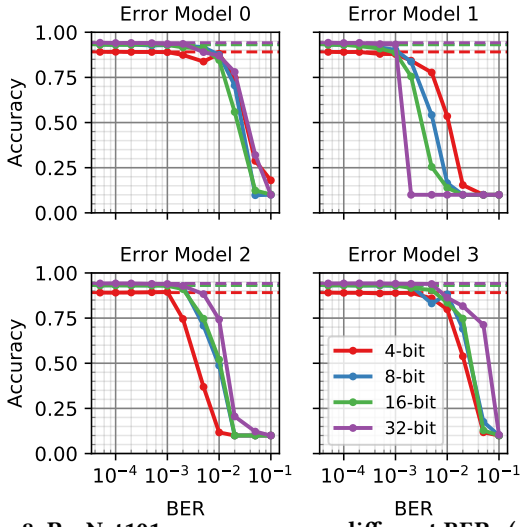


**Figure 8: ResNet101 accuracy across different BERs ($x$-axis) and quantization levels when we use four error models to inject bit errors. We fit the parameters of the error models to the errors observed by reducing $tRCD$ in a real DRAM device from Vendor A.**

**Quantization**. Precision also affects the error model and the error tolerance curve. For example, in Error Model 2, we observe that the int-4 DNN has the weakest error tolerance curve. We find that this is because Error Model 2 clusters weak cells along a row: a large number of neighboring 4-bit values end up corrupted when Error Model 2 indicates a weak wordline. This is in contrast to larger precisions, which might have numbers distributed more evenly across rows, or error models that do not capture error locality (e.g., Error Model 0). In general, we find that clusters of erroneous values cause significant problems with accuracy (the errors compound faster as they interact with each other in the DNN). Such locality of errors is more common in low-bitwidth precisions and with spatial correlation-based error models (Error Models 1 and 2).

**DNN Size**. We observe that larger DNNs (e.g., VGG16) are more error resilient. Larger models exhibit an accuracy drop-off at higher BER ($> 10^{-2}$) as compared to smaller models (e.g. SqueezeNet1.1, $< 10^{-3}$). These results are not plotted.

**Accuracy Collapse**. We can observe the accuracy collapse phenomenon caused by implausible values (see Section 3.2) when we increase the bit error rate over $10^{-6}$ in large networks. These implausible values propagate, and in the end, they cause accuracy collapse in the DNN.

## 6.4 Curricular Retraining Evaluation

We run DNN inference on real DRAM devices using the boosted DNN model generated by our curricular retraining mechanism. To our knowledge, this is the first demonstration of DNN inference on real approximate memory. We also evaluate our curricular retraining mechanism using our error models (see Section 4).

**Experimental Setup**. We evaluate curricular retraining using real DRAM devices by running LeNet [89] on the CIFAR-10 [4] validation dataset. We use SoftMC [58] to scale $V_{DD}$ and $t_{RCD}$ on an FPGA-based infrastructure connected to a DDR3 DRAM module. We also evaluate curricular retraining using our error models by running ResNet [59] on the CIFAR-10 validation dataset.

**Results with Real DRAM**. Figure 9 shows the accuracy of 1) baseline LeNet without applying any retraining mechanism (Baseline), and 2) LeNeT boosted with our curricular retraining mechanism (Boosted), as a function of DRAM supply voltage and $t_{RCD}$. We make two observations. First, EDEN's boosted LeNet allows a voltage reduction of ~0.25V and a $t_{RCD}$ reduction of 4.5ns, while maintaining accuracy values equivalent to those provided by nominal voltage (1.35V) and nominal $t_{RCD}$ (12.5ns). Second, the accuracy of baseline LeNet decreases very quickly when reducing voltage and $t_{RCD}$ below the nominal values. We conclude that our curricular retraining mechanism can effectively boost the accuracy of LeNeT on approximate DRAM with reduced voltage and $t_{RCD}$.

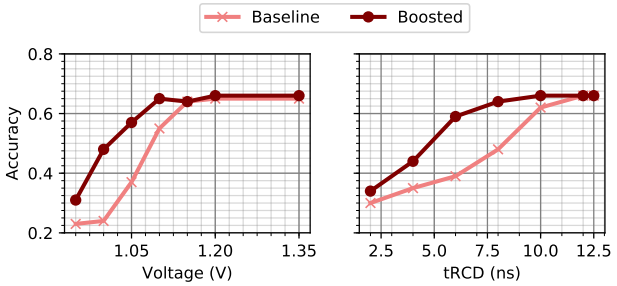


**Figure 9: LeNet accuracy using baseline and boosted DNNs.**

**Results with Error Models**. Figure 10 (left) shows an experiment that retrains ResNet101 with two different models: 1) a good-fit error model (that closely matches the tested device) and 2) a poor-fit error model. We make two observations. First, retraining using a poor-fit error model (red), yields little improvement over the baseline (no retraining, green). Second, retraining with a good-fit error model (blue) improves BER at the 89% accuracy point by >10x (shifting the BER curve right). We conclude that using a good-fit error model in the retraining mechanism is critical to avoid accuracy collapse.

Figure 10 (right) shows the effectiveness of our curricular retraining mechanism using a good-fit error model. We make two observations. First, the accuracy of the DNN with regular retraining (purple) collapses, compared to the baseline DNN (no retraining, green). Second, the DNN trained with our curricular retraining (orange) exhibits a boosted error tolerance. We conclude that our curricular retraining mechanism is effective at boosting the DNN accuracy in systems that use approximate DRAM.

Running this retraining process for 10-15 epochs is sufficient to boost tolerable BERs by 5-10x to achieve the same DNN accuracy

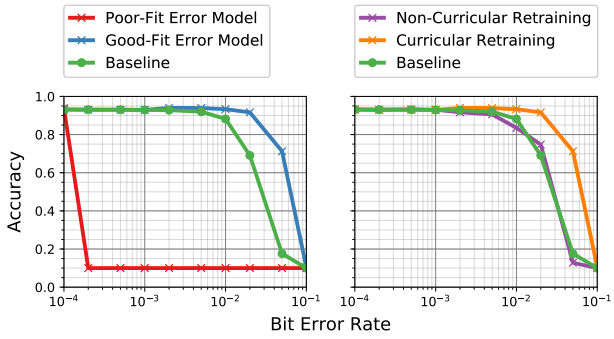


**Figure 10: Accuracy of boosted ResNet101 DNNs in presence of memory errors. Left: accuracy of poor-fit and good-fit error models. Right: accuracy of non-curricular and curricular retraining using a good-fit error model.**

as the baseline DNN executed in DRAM with nominal parameters. For our ResNet101 on CIFAR-10 with an NVIDIA Tesla P100, this one-time boosting completes within 10 minutes.

## 6.5 Coarse-Grained DNN Characterization and Mapping

In this section, we show the results of EDEN's coarse-grained DNN characterization (see Section 3.3) and how the target DNN model maps to an approximate DRAM with optimized parameters for a target accuracy degradation of < 1%.

**Characterization**. Table 3 shows the DNN's maximum tolerable BER for eight DNN models with FP32 and int8 numeric precisions.

| | FP32 | | | int8 | | |
|---|---|---|---|---|---|---|
| **Model** | BER | $\Delta V_{DD}$ | $\Delta t_{RCD}$ | BER | $\Delta V_{DD}$ | $\Delta t_{RCD}$ |
| **ResNet101** | 4.0% | -0.30V | -5.5ns | 4.0% | -0.30V | -5.5ns |
| **MobileNetV2** | 1.0% | -0.25V | -1.0ns | 0.5% | -0.10V | -1.0ns |
| **VGG-16** | 5.0% | -0.35V | -6.0ns | 5.0% | -0.35V | -6.0ns |
| **DenseNet201** | 1.5% | -0.25V | -2.0ns | 1.5% | -0.25V | -2.0ns |
| **SqueezeNet1.1** | 0.5% | -0.10V | -1.0ns | 0.5% | -0.10V | -1.0ns |
| **AlexNet** | 3.0% | -0.30V | -4.5ns | 3.0% | -0.30V | -4.5ns |
| **YOLO** | 5.0% | -0.35V | -6.0ns | 4.0% | -0.30V | -5.5ns |
| **YOLO-Tiny** | 3.5% | -0.30V | -5.0ns | 3.0% | -0.30V | -4.5ns |

**Table 3: Maximum tolerable BER for each DNN using EDEN's coarse-grained characterization, and DRAM parameter reduction to achieve the maximum tolerable BER. Nominal parameters are $V_{DD} = 1.35V$ and $t_{RCD} = 12.5ns$.**

We observe that the maximum tolerable BER demonstrates significant variation depending on the DNN model. For example, YOLO tolerates 5% BER and SqueezeNet tolerates only 0.5%. We conclude that 1) the maximum tolerable BER highly depends on the DNN model, and 2) DNN characterization is required to optimize approximate DRAM parameters for each DNN model.

**Mapping**. EDEN maps each DNN model to an approximate DRAM module that operates with the maximum reduction in voltage ($\Delta V_{DD}$) and $t_{RCD}$ ($\Delta t_{RCD}$) that leads to a BER below the maximum DNN tolerable BER for that DNN model. Table 3 shows the maximum reduction in DRAM voltage ($\Delta V_{DD}$) and $t_{RCD}$ ($\Delta t_{RCD}$) that causes a DRAM BER below the maximum tolerable BER, for a target DRAM module from vendor A. The nominal DRAM parameters for this DRAM module are $V_{DD} = 1.35V$ and $t_{RCD} = 12.5ns$ . We make two observations. First, the tolerable BER of a network

is directly related to the maximum tolerable $V_{DD}$ and $t_{RCD}$ reductions. Second, the reductions in $V_{DD}$ and $t_{RCD}$ are very significant compared to the nominal values. For example, EDEN can reduce voltage by 26% and $t_{RCD}$ by 48% in YOLO while maintaining the DNN accuracy to be within 1% of the original accuracy.

### 6.6 Fine-Grained DNN Characterization and Mapping

**Characterization**. We characterize the ResNet101 DNN model with our fine-grained DNN characterization procedure (see Section 3.3). For each IFM and weight, we iteratively increase the bit error rate until we reach the maximum tolerable BER of the data type for a particular target accuracy degradation. We perform a full network retraining in each iteration. To reduce the runtime of our procedure , we sample 10% of the validation set during each inference run to obtain the accuracy estimate. We also bootstrap the BERs to the BER found in coarse-grained DNN characterization and use a linear scale in 0.5 increments around that value. For ResNet101, this one-time characterization completes in one hour using an Intel Xeon CPU E3-1225 [1].

Figure 11 shows the maximum tolerable BER for each IFM and weight in ResNet101 obtained with our fine-grained DNN characterization method (Section 3.3), assuming a maximum accuracy loss of <1%. Each bar in the figure represents the BER tolerance of an IFM or weight, and they are ordered by their depth in the DNN, going deeper from left to right. We make three observations. First, fine-grained characterization enables individual IFMs and weights to tolerate up to 3x BER (13% for the last weight) of the maximum tolerable BER of the coarse-grained approach (4% for ResNet101 in Table 3). Second, weights usually tolerate more errors than IFMs. Third, the maximum tolerable BER is smaller in the first layers than in the middle layers of the DNN. We conclude that fine-grained DNN characterization enables a significant increase in the maximum tolerable BER compared to coarse-grained characterization.

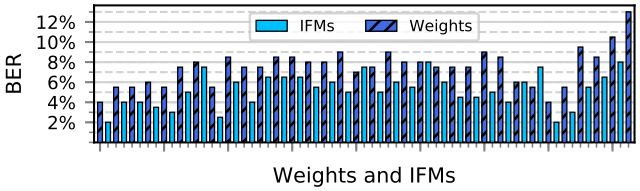


**Figure 11: Fine-grained characterization of the tolerable BERs of ResNet101 IFMs and weights. Deeper layers are on the right.**

**Mapping**. We map each individual IFM or weight into different DRAM partitions based on 1) the BER tolerance of each IFM and weight, and 2) the BER of each DRAM partition, using our algorithm in Section 3.4. Figure 12 shows an example that maps the ResNet101 IFMs and weights from Figure 11 into 4 different DRAM partitions with different voltage parameters that introduce different BERs (four horizontal colored bars), following the algorithm in Section 3.4.

We conclude that the wide range of tolerable BERs across all ResNet101 data types enables the use of both 1) DRAM partitions with significant voltage reduction (e.g., horizontal red line), and 2) DRAM partitions with moderate voltage reduction (e.g., horizontal blue line).

## 7 SYSTEM LEVEL EVALUATION

We evaluate EDEN in three different DNN inference architectures: CPUs, GPUs, and inference accelerators.

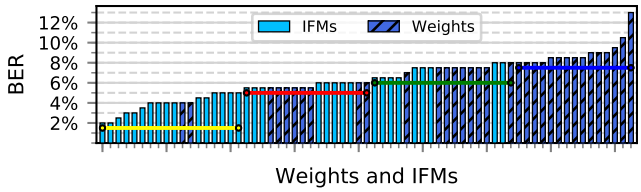


**Figure 12: Mapping of ResNet101 IFMs and weights into four partitions with different $V_{DD}$ values (colored horizontal lines)**

### 7.1 CPU Inference

**Experimental Setup**. We evaluate EDEN on top of a multi-core OoO CPU using the simulated core configuration listed in Table 4. We use ZSim [145] and Ramulator [81] to simulate the core and the DRAM subsystem, respectively. We use DRAMPower [16] to estimate energy consumption for DDR4 devices. We use a 2-channel, 32-bank 8GB DDR4-2133 DRAM device.

| | |
|---|---|
| **Cores** | 2 Cores @ 4.0 GHz, 32nm, 4-wide OoO, Buffers: 18-entry fetch, 128-entry decode, 128-entry reorder buffer, |
| **L1 Caches** | 32KB, 8-way, 2-cycle, Split Data/Instr. |
| **L2 Caches** | 512KB per core, 8-way, 4-cycle, Shared Data/Instr., Stream Prefetcher |
| **L3 Caches** | 8MB per core, 16-way, 6-cycle, Shared Data/Instr., Stream Prefetcher |
| **Main Memory** | 8GB DDR4-2133 DRAM, 2 channels, 16 banks/channel |

**Table 4: Simulated system configuration.**

We use twelve different inference benchmarks: eight from the Intel OpenVINO toolkit [83] and four from the AlexeyAB-fork of the DarkNet framework [136]. For each DNN, we study the FP32 and the `int8`-quantized variant. We use 8-bit quantization in our baselines, because it is commonly used for production CPU workloads [66]. We evaluate EDEN's coarse-grained DNN characterization procedure and target a < 1% accuracy degradation. Table 3 lists the reduced $V_{DD}$ and $t_{RCD}$ values.

**DRAM Energy**. Figure 13 shows the DRAM energy savings of EDEN, compared to a system with DRAM operating at nominal voltage and nominal latency. We make two observations. First, EDEN achieves significant DRAM energy savings across different DNN models. The average DRAM energy savings is 21% across all workloads, and 29% each for YOLO and VGG. Second, the DRAM energy savings for FP32 and int8 are roughly the same, because the voltage reduction is very similar for both precisions (see Table 3).

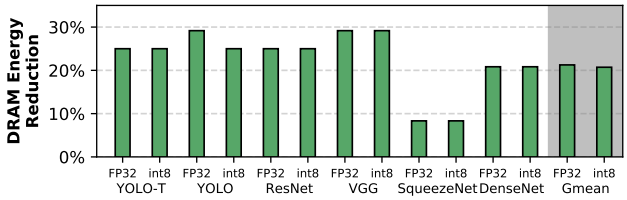


**Figure 13: DRAM energy savings of EDEN. We use FP32 and quantized int8 networks.**

We also perform evaluations for a target accuracy that is the same as the original. Our results show that EDEN enables an average DRAM energy reduction of 16% (up to 18%).

We conclude that EDEN is effective at saving DNN inference energy by reducing voltage while maintaining the DNN accuracy within 1% of the original.

**Performance**. Figure 14 shows the speedup of EDEN when we reduce $t_{RCD}$, and the speedup of a system with a DRAM module that has ideal $t_{RCD} = 0$, compared to a system that uses DRAM with nominal timing parameters. We make three observations. First, YOLO DNNs exhibit high speedup with EDEN, reaching up to 17% speedup. The results of YOLO are better than the average because YOLO is more sensitive to DRAM latency. This is because some steps in YOLO (e.g., Non-Maximum Suppression [61, 119], confidence and IoU thresholding [137, 138]) perform arbitrary indexing into matrices that lead to random memory accesses, which cannot easily be predicted by the prefetchers . Second, the average speedup of EDEN (8%) is very close to the average speedup of the ideal system with $t_{RCD} = 0$ (10%). Third, we find that SqueezeNet1.1 and ResNet101 exhibit very little maximum theoretical speedup because they are not bottlenecked by memory latency.

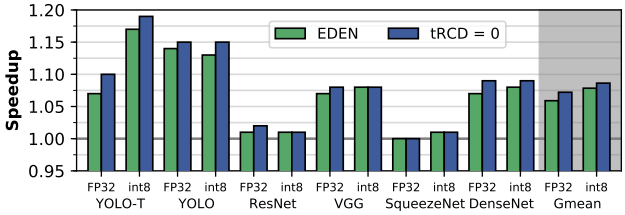


**Figure 14: Speedup of EDEN over baseline and versus a system with ideal activation latency. We use FP32 and an quantized int8 networks.**

We also perform evaluations for a target accuracy that is the same as the original. Our results show that EDEN enables an average performance gain of 4% (up to 7%).

We conclude that EDEN is effective at improving DNN inference performance by reducing DRAM latency while maintaining the DNN accuracy within 1% of the original, especially on DNNs that are sensitive to memory latency.

## 7.2 Accelerators

We evaluate EDEN on three different accelerators: GPU [124], Eyeriss [26], and TPU [69].

**GPU Inference**. We evaluate EDEN on a GPU using the cycle-accurate GPGPU-Sim simulator [11]. We use GPUWattch [95] to evaluate the overall GPU energy consumption. Table 5 details the NVIDIA Titan X GPU model [2] we use in our evaluation. We use the reduced $t_{RCD}$ and $V_{DD}$ values that provide $<$ 1% accuracy degradation (as listed in Table 3). We adapt four DarkNet-based binaries to run inference on the FP32/int8 YOLO and YOLO-Tiny DNNs.

| | |
|---|---|
| **Shader Core** | 28 SMs, 1417 MHz, 32 SIMT Width, 64 Warps per SM, 4 GTO Schedulers per Core |
| **Private L1 Cache** | 24 KB per SMM, Cache Block Size 128B |
| **Shared Memory** | 96 KB, 32 Banks. Shared L2 Cache: 3MB |
| **Main Memory** | GDDR5, 2500MHz, 6 channels, 24 chips |

**Table 5: Simulated NVIDIA Titan X GPU configuration**

Our results show that EDEN provides 37% average energy reduction (41.7% for YOLO-Tiny, and 32.6% for YOLO) compared to a GPU that uses DRAM with nominal parameters.

Our results also show that EDEN provides 2.7% average speedup (5.5% for the YOLO-Tiny, and 0% for YOLO) compared to a GPU that uses DRAM with nominal parameters. DRAM with ideal *tRCD* ($t_{RCD} = 0$) provides 6% speedup for YOLO-Tiny and 2% speedup for YOLO. These results indicate that 1) the YOLO DNN family is not DRAM latency bound in our evaluation configuration, and 2) EDEN can achieve close to the ideal speedup of zero activation latency when the DNN is latency bound.

**Neural Network Inference Accelerators**. We evaluate EDEN on Eyeriss [26] and Google's Tensor Processing Unit (TPU) [69] using the cycle-accurate SCALE-Sim simulator [144]. We use DRAMPower [16] to obtain DRAM energy consumption from memory traces produced by SCALE-Sim. We use the built-in int8 AlexNet and YOLO-Tiny models and their accelerator-specific dataflows. We use DRAM parameters that yield a maximum accuracy loss of 1% (Table 3). Table 6 details the configuration of the Eyeriss and TPU inference accelerators. Eyeriss has an array of 12x14 processing elements (PEs) with a 324KB SRAM buffer for all data types (i.e., IFMs, weights and OFMs), and the TPU has an array of 256x256 PEs with a 24MB SRAM buffer for all data types. We evaluate both accelerators with DDR4 and LPDDR3 DRAM configurations, using Alexnet and YOLO-Tiny workloads.

| | **Eyeriss** | **TPU** |
|---|---|---|
| **Array** | 12 × 14 PEs | 256 × 256 PEs |
| **SRAM Buffers** | 324 KB | 24 MB |
| **Main Memory** | 4GB DDR4-2400 | 4GB DDR4-2400 |
| | 4GB LPDDR3-1600 | 4GB LPDDR3-1600 |

**Table 6: Simulated Eyeriss and TPU configurations.**

Our results show that reducing the voltage level in DDR4 DRAM leads to significant DRAM energy reductions on both Eyeriss and TPU accelerators. EDEN provides 1) 31% average DRAM energy savings on Eyeriss (31% for YOLO-Tiny, and 32% for Alexnet), and 2) 32% average DRAM energy savings on TPU (31% for YOLO-Tiny, and 34% for Alexnet).

Our results with a reduced voltage level in LPDDR3 are similar to those with DDR4. EDEN provides an average DRAM energy reduction of 21% for both Eyeriss and TPU accelerators running YOLO-Tiny and Alexnet. By using the accelerator/network/cache/-DRAM energy breakdown provided by the Eyeriss evaluations on AlexNet [161], we estimate that EDEN can provide 26.8% system-level energy reduction on fully-connected layers and 7% system-level energy reduction on convolutional layers.

Our results with reduced $t_{RCD}$ in LPDDR3 and DDR4 show that Eyeriss and TPU exhibit no speedup from reducing $t_{RCD}$. We observe that prefetchers are very effective in these architectures because the memory access patterns in the evaluated DNNs are very predictable.

# 8 RELATED WORK

To our knowledge, this paper is the first to propose a general framework that reduces energy consumption and increases performance of DNN inference by using approximate DRAM with reduced voltage and latency. EDEN introduces a new methodology to improve DNN's tolerance to approximate DRAM errors which is based on DNN error tolerance characterization and a new curricular retraining mechanism. We demonstrate the effectiveness of EDEN by using error patterns that occur in real approximate DRAM devices.

In this section, we discuss closely related work on 1) approximate computing hardware for DNN workloads, and 2) modifying DRAM parameters.

**Approximate Computing Hardware for DNN Workloads**. Many prior works propose to use approximate computing hardware for executing machine learning workloads [30, 38, 97, 110, 118, 131, 132, 134, 141, 143, 155, 163, 166, 169, 173, 174, 179, 180]. All these

works propose techniques for improving DNN tolerance for different types of approximate hardware mechanisms and error injection rates. Compared to these works, EDEN is unique in 1) being the first work to use approximate DRAM with reduced voltage and latency, 2) being the first demonstration of DNN inference using error characterization of real approximate DRAM devices, 3) using a novel curricular retraining mechanism that is able to customize the DNN for tolerating *high* error rates injected by the target approximate DRAM, and 4) mapping each DNN data type to a DRAM partition based on the error tolerance of the DNN data type and the bit error rate of the DRAM partition. We classify related works on approximate hardware for DNN workloads into six categories.

First, works that reduce DRAM refresh to save DNN energy [121, 122, 164]. RANA [164] and St-DRC [121] propose to reduce DRAM refresh rate in the embedded DRAM (eDRAM) memory of DNN accelerators. Nguyen et al. [122] propose to apply similar refresh optimization techniques to off-chip DRAM in DNN accelerators. These mechanisms use customized retraining mechanisms to improve the accuracy of the DNN in the presence of a moderate amount of errors.

Second, works that study the error tolerance of neural networks to uniform random faults in SRAM memory [97, 118, 134, 142, 143]. For example, Li et al. [97] analyze the effect of various numeric representations on error tolerance. Minerva [135] proposes an algorithm-aware fault mitigation technique to mitigate the effects of low-voltage SRAM in DNN accelerators.

Third, works that study approximate arithmetic logic in DNN workloads [141, 141, 178, 179]. ThUnderVolt [178] proposes to underscale the voltage of arithmetic elements. Salami et al. [141] and Zhang et al. [179] present fault-mitigation techniques for neural networks that minimize errors in faulty registers and logic blocks with pruning and retraining.

Fourth, works that study approximate emerging memory technologies for neural network acceleration. Panda et al. [125] and Kim [80] propose neuromorphic accelerators that use spintronics and memristors to run a proof-of-concept fuzzy neural network.

Fifth, works that study the effects of approximate storage devices on DNN workloads [132, 155]. Qin et al. [132] study the error tolerance of neural networks that are stored in approximate non-volatile memory (NVM) media. The authors study the effects of turning the ECC off in parts of the NVM media that store the neural network data. Wen et al. [155] propose to mitigate the effects of unreliable disk reads with a specialized ECC variant that aims to mitigate error patterns present in weights of shallow neural networks.

Sixth, works that study the intrinsic error resilience of DNNs by injecting randomly-distributed errors in DNN data [110, 110, 141, 163, 166, 179, 180]. These works assume that the errors can come from any component of the system (i.e., they do not target a specific approximate hardware component). Marques et al. [110] study the accuracy of DNNs under different error injection rates and propose various error mitigation techniques. This work uses a simple probabilistic method to artificially inject errors into the DNN model. ApproxANN [180] uses an algorithm that optimizes the DNN accuracy by taking into account the error tolerance and the criticality of each component of the network. The quality-configurable Neuromorphic Processing Engine (qcNPE) [166] uses processing elements with dynamically configurable accuracy for executing approximate neural networks.

**Modifying DRAM Parameters**. Many prior works study the effects of modifying DRAM parameters on reliability, performance and energy consumption. We already discuss some prior works that reduce DRAM voltage, access latency, and refresh rate in Section 2.3. EDEN leverages the characterization techniques introduced in Voltron [21] and Flexible-Latency DRAM [19] to perform the DRAM characterization required to map a DNN to approximate DRAM with reduced voltage and reduced latency (Section 3.4). We classify other related works that modify DRAM parameters into three categories.

First, works that aim to characterize and reduce energy consumption at reduced supply voltage levels [21, 36, 39, 47]. David et al. [36] propose memory dynamic voltage and frequency scaling (DVFS) to reduce DRAM power. MemScale [39] provides dynamic voltage and/or frequency scaling in main memory to reduce energy consumption, while meeting a maximum tolerable performance degradation. Voltron [21] studies voltage reduction in real DRAM devices in detail and proposes solutions to reduce voltage reliably based on observed error characteristics and system performance requirements. VAMPIRE [47] proposes a new DRAM power model that is based on the characteristics of real DRAM devices.

Second, works that investigate DRAM characteristics under reduced access latency [15, 19, 21, 76–78, 93, 94]. Adaptive-Latency DRAM [94] characterizes the guardbands present in timing parameters defined by DRAM manufacturers, and exploits the extra timing margins to reliably reduce DRAM latency across different chips and temperatures. Flexible-Latency DRAM [19] analyzes the spatial distribution of reduced-latency-induced cell failures, and uses this information to reliably access different regions of DRAM with different timing parameters. DIVA-DRAM [93] proposes an automatic method for finding the lowest reliable operation latency of DRAM, via a combination of runtime profiling and ECC.

Third, works that aim to reduce DRAM latency by modifying the microarchitecture of DRAM or the memory controller [31, 56, 57, 91, 92, 108, 157, 168]. These works reduce latency without introducing bit errors.

## 9 CONCLUSION

This paper introduces EDEN, the first general framework that enables energy-efficient and high-performance DNN inference via approximate DRAM, while strictly meeting a target DNN accuracy. EDEN uses an iterative mechanism that profiles the DNN and the target approximate DRAM with reduced voltage and timing parameters. EDEN improves DNN accuracy with a novel curricular retraining mechanism that tolerates high bit error rates. We evaluate EDEN in both simulation and on real hardware. Our evaluation shows that EDEN enables 1) an average DRAM energy reduction of 21%, 37%, 31%, and 32% in CPU, GPU, Eyeriss, and TPU architectures, respectively, across a variety of state-of-the-art DNNs, and 2) average (maximum) performance gains of 8% (17%) in CPUs and 2.7% (5.5%) in GPUs, for latency-bound DNNs. We expect that the core principles of EDEN generalize well across different memory devices, memory parameters, and memory technologies. We hope that EDEN enables further research and development on the use of approximate memory for machine learning workloads.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for feedback. We thank the SAFARI Research Group members for feedback and the stimulating intellectual environment they provide. We acknowledge the generous funding provided by our industrial partners: Alibaba, Facebook, Google, Huawei, Intel, Microsoft, and VMware. This research was supported in part by the Semiconductor Research Corporation. Skanda Koppula was supported by the Fulbright/Swiss Government Excellent Scholarship 2018.0529.

# REFERENCES


[1] "Intel Xeon CPU E3-1225," https://ark.intel.com/content/www/us/en/ark/products/52270/intel-xeon-processor-e3-1225-6m-cache-3-10-ghz.html.
[2] "NVIDIA Titan X GPU," https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/.
[3] "SoftMC Source Code." https://github.com/CMU-SAFARI/SoftMC
[4] "The CIFAR-10 Dataset." https://www.cs.toronto.edu/~kriz/cifar.html
[5] 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 *ICCD*, 2014.
[6] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," in *ISCA*, 2016.
[7] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-Layer CNN Accelerators," in *MICRO*, 2016.
[8] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration," *TCAD*, 2017.
[9] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, and P. Vahle, "A Convolutional Neural Network Neutrino Event Classifier," *JINST*, 2016.
[10] S. Baek, S. Cho, and R. Melhem, "Refresh Now and Then," *TC*, 2013.
[11] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA Workloads using a Detailed GPU Simulator," in *ISPASS*, 2009.
[12] E. Baseman, N. Debardeleben, S. Blanchard, J. Moore, O. Tkachenko, K. Ferreira, T. Siddiqua, and V. Sridharan, "Physics-Informed Machine Learning for DRAM Error Modeling," in *DFT*, 2018.
[13] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.
[14] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *TCSVT*, 2017.
[15] 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 *DATE*, 2014.
[16] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, "DRAMPower: Open-source DRAM Power & Energy Estimation Tool," 2012.
[17] K. K. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.
[18] K. K. Chang, "Understanding and Improving the Latency of DRAM-Based Memory Systems," *Ph.D. dissertation, Carnegie Mellon Univ.*, 2017.
[19] 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 *SIGMETRICS*, 2016.
[20] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.
[21] 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," *SIGMETRICS*, 2017.
[22] G. Chen, C. Parada, and G. Heigold, "Small-Footprint Keyword Spotting using Deep Neural Networks." in *ICASSP*, 2014.
[23] T. Chen, T. Moreau, Z. Jiang, L. Zheng, S. Jiao, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *OSDI*, 2018.
[24] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training Deep Nets with Sublinear Memory Cost," *arXiv*, 2016.
[25] 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," *ASPLOS*, 2014.
[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *JSSC*, 2017.
[27] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *JETCAS*, 2019.
[28] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *arXiv*, 2014.
[29] 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 *ISCA*, 2016.
[30] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and Characterization of Inherent Application Resilience for Approximate Computing," in *DAC*, 2013.
[31] J. Choi, W. Shin, J. Jang, J. Suh, Y. Kwon, Y. Moon, and L.-S. Kim, "Multiple Clone Row DRAM: A Low Latency and Area Optimized DRAM," in *ISCA*, 2015.
[32] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," in *ISCA*, 2004.
[33] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to+ 1 or-1," *arXiv*, 2016.
[34] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *NIPS*, 1990.
[35] A. Das, H. Hassan, and O. Mutlu, "VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency," in *DAC*, 2018.
[36] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC*, 2011.
[37] C. De Sa, M. Leszczynski, J. Zhang, A. Marzoev, C. R. Aberger, K. Olukotun, and C. Ré, "High-Accuracy Low-Precision Training," *arXiv*, 2018.
[38] J. Deng, Y. Rang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu, "Retraining-Based Timing Error Mitigation for Hardware Neural Networks," in *DATE*, 2015.
[39] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active Low-Power Modes for Main Memory," in *ASPLOS*, 2011.
[40] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference," in *DAC*, 2018.
[41] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a Deep Convolutional Network for Image Super-Resolution," in *ECCV*, 2014.
[42] S. S. Du and J. D. Lee, "On the Power of Over-parametrization in Neural Networks with Quadratic Activation," *arXiv*, 2018.
[43] J. Dundas and T. Mudge, "Improving Data Cache Performance by Pre-executing Instructions Under a Cache Miss," in *ICS*, 1997.
[44] J. D. Dundas, "Improving Processor Performance by Dynamically Pre-Processing the Instruction Stream," University of Michigan, Tech. Rep., 1999.
[45] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," *ASPLOS*, 2017.
[46] S. Ghose, T. Li, N. Hajinazar, D. Senol Cali, and O. Mutlu, "Demystifying Complex Workload-DRAM Interactions: An Experimental Study," in *SIGMETRICS*, 2019.
[47] S. Ghose, A. G. Yağlıkçı, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, M. O'Connor, and O. Mutlu, "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study," *SIGMETRICS*, 2018.
[48] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, "The Reversible Residual Network: Backpropagation without Storing Activations," in *NIPS*, 2017.
[49] A. Graves and N. Jaitly, "Towards End-to-End Speech Recognition with Recurrent Neural Networks," in *ICML*, 2014.
[50] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," in *ICRA*, 2017.
[51] M. Guan and L. Wang, "Temperature Aware Refresh for DRAM Performance Improvement in 3D ICs," in *ISQED*, 2015.
[52] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA," *TCAD*, 2017.
[53] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *TED*, 1998.
[54] 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 *ISCA*, 2016.
[55] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv*, 2015.
[56] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.
[57] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. Mansouri Ghiasi, S. Ghose, and O. Mutlu, "CROW : A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," in *ISCA*, 2019.
[58] 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 *HPCA*, 2017.
[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.
[60] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *ECCV*, 2018.
[61] J. Hosang, R. Benenson, and B. Schiele, "Learning Non-maximum Suppression," in *CVPR*, 2017.
[62] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *IMLR*, 2017.
[63] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "DenseNet: Implementing Efficient ConvNet Descriptor Pyramids," *arXiv*, 2014.
[64] 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*, 2016.
[65] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "RAPIDNN: In-Memory Deep Neural Network Acceleration Framework," *arXiv*, 2018.
[66] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *CVPR*, 2018.
[67] JEDEC Standard, "DDR4 SDRAM Specification (JESD79-4)," 2012.
[68] W. Jin, C. Coley, R. Barzilay, and T. Jaakkola, "Predicting Organic Reaction Outcomes with Weisfeiler-Lehman Network," in *NIPS*, 2017.
[69] 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 *ISCA*, 2017.

[70] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *MICRO*, 2016.
[71] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Approximate Computing with Partially Unreliable Dynamic Random Access Memory-Approximate DRAM," in *DAC*, 2016.
[72] 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 *MEMSYS*, 2015.
[73] B. Keeth and R. J. Baker, *DRAM Circuit Design: A Tutorial*. Wiley-IEEE Press, 2000.
[74] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[75] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," in *SIGMETRICS*, 2014.
[76] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," in *ICCD*, 2018.
[77] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.
[78] J. S. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.
[79] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.
[80] Y. Kim, "Energy Efficient and Error Resilient Neuromorphic Computing in VLSI," Ph.D. dissertation, MIT, 2013.
[81] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator." *CAL*, 2016.
[82] I. Kokkinos, "UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory." in *CVPR*, 2017.
[83] A. Kozlov and D. Osokin, "Development of Real-time ADAS Object Detector for Deployment on CPU," in *IntelliSys*, 2019.
[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.
[85] H. Kwon, M. Pellauer, and T. Krishna, "MAESTRO: An Open-Source Infrastructure for Modeling Dataflows within Deep Learning Accelerators," *arXiv*, 2018.
[86] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *ASPLOS*, 2018.
[87] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, 2015.
[88] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *NIPS*, 1990.
[89] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, 1998.
[90] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition," *CTP-PBSRI*, 1995.
[91] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[92] D. Lee, L. Subramanian, R. Ausavarungnirun, J. Choi, and O. Mutlu, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[93] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," *SIGMETRICS*, 2017.
[94] 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 *HPCA*, 2015.
[95] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *ISCA*, 2013.
[96] D. Levinthal, "Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors ," https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf, 2009.
[97] 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 *SC*, 2017.
[98] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient Convnets," *arXiv*, 2016.
[99] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "SmartShuttle: Optimizing Off-Chip Memory Accesses for Deep Learning Accelerators," in *DATE*, 2018.
[100] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator." in *MICRO*, 2018.
[101] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A Dram-Based Reconfigurable In-Situ Accelerator," in *MICRO*.
[102] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," *arXiv*, 2015.
[103] D. Lin, S. Talathi, and S. Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks," in *ICML*, 2016.
[104] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *ECCV*, 2014.
[105] 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 *ISCA*, 2013.
[106] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
[107] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," *TVLSI*, 2018.
[108] S.-L. Lu, Y.-C. Lin, and C.-L. Yang, "Improving DRAM Latency with Dynamic Asymmetric Subarray," in *MICRO*, 2015.
[109] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks," in *HPCA*, 2017.
[110] J. Marques, J. Andrade, and G. Falcao, "Unreliable Memory Operation on a Convolutional Neural Network Processor," in *SiPS*, 2017.
[111] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.
[112] Micron, "TN-40-07: Calculating Memory Power for DDR4 SDRAM." https://www.micron.com/-/media/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf
[113] O. Mutlu, "Main Memory Scaling: Challenges and Solution Directions," in *More than Moore Technologies for Next Generation Computer Design*, 2015.
[114] O. Mutlu, H. Kim, and Y. N. Patt, "Techniques for Efficient Processing in Runahead Execution Engines," in *ISCA*, 2005.
[115] O. Mutlu, H. Kim, J. Stark, and Y. N. Patt, "On Reusing the Results of Pre-Executed Instructions in a Runahead Execution Processor," in *CAL*, 2005.
[116] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors," in *HPCA*, 2003.
[117] M. Nazemi, G. Pasandi, and M. Pedram, "NullaNet: Training Deep Neural Networks for Reduced-Memory-Access Inference," *arXiv*, 2018.
[118] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar, "A Reliability Study on CNNs for Critical Embedded Systems," in *ICCD*, 2018.
[119] A. Neubeck and L. Van Gool, "Efficient Non-maximum Suppression," in *ICPR*, 2006.
[120] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks," *arXiv*, 2018.
[121] D.-T. Nguyen, N.-M. Ho, and I.-J. Chang, "St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs," in *DAC*, 2019.
[122] 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 *ISCAS*, 2018.
[123] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and Generalization in Neural Networks: An Empirical Study," *arXiv*, 2018.
[124] K.-S. Oh and K. Jung, "GPU Implementation of Neural Networks," *JPRR*, 2004.
[125] 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 *DAC*, 2016.
[126] 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 *ISCA*, 2017.
[127] 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," *NIPS-W*, 2017.
[128] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices," in *DSN*, 2019.
[129] M. Patel, J. S. Kim, and O. Mutlu, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," *ISCA*, 2017.
[130] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-Centric Accelerator Design for Convolutional Neural Networks." in *ICCD*, 2013.
[131] D. S. Phatak and I. Koren, "Complete and Partial Fault Tolerance of Feedforward Neural Nets," *TNN*, 1995.
[132] M. Qin, C. Sun, and D. Vucinic, "Robustness of Neural Networks against Storage Media Errors," *arXiv*, 2017.
[133] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[134] 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 *DAC*, 2018.
[135] 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 *ISCA*, 2016.
[136] J. Redmon, "Darknet: Open Source Neural Networks in C," https://pjreddie.com/darknet/, 2013.

[137] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv*, 2017.
[138] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression," in *CVPR*, 2019.
[139] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, 1951.
[140] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, 2015.
[141] B. Salami, O. Unsal, and A. Cristal, "On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation," *arXiv*, 2018.
[142] B. Salami, O. S. Unsal, and A. C. Kestelman, "Comprehensive Evaluation of Supply Voltage Underscaling in FPGA On-chip Memories," in *MICRO*, 2018.
[143] A. H. Salavati and A. Karbasi, "Multi-Level Error-Resilient Neural Networks," in *ISIT*, 2012.
[144] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN Accelerator," in *arXiv*, 2018.
[145] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *ISCA*, 2013.
[146] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: The Next Generation of On-Device Computer Vision Networks," in *CVPR*, 2018.
[147] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," in *SIGMETRICS*, 2009.
[148] 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*, 2018.
[149] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, "Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks," *ACS central science*, 2017.
[150] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.
[151] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," *arXiv*, 2019.
[152] 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," *ISCA*, 2016.
[153] 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*, 2017.
[154] Y. Shen, M. Ferdman, and P. Milder, "Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer," in *FCCM*, 2017.
[155] 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 *CASES*, 2014.
[156] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv*, 2014.
[157] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA*, 2013.
[158] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-Brain: A Deep Learning Accelerator that Tames the Diversity of CNNs through Adaptive Data-Level Parallelization," in *DAC*, 2016.
[159] E. Sprangle and D. Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," in *ISCA*, 2002.
[160] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *JMLR*, 2014.
[161] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A tutorial and Survey," *Proceedings of the IEEE*, 2017.
[162] X. Tang, M. Kandemir, P. Yedlapalli, and J. Kotra, "Improving Bank-Level Parallelism for Irregular Applications," in *MICRO*, 2016.
[163] O. Temam, "A Defect-Tolerant Accelerator for Emerging High-Performance Applications," in *ISCA*, 2012.
[164] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM," in *ISCA*, 2018.
[165] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: A 7.49 TOPS Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96MB 3D SRAM Using Inductive-Coupling Technology in 40nm CMOS," in *ISSCC*, 2018.
[166] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-Efficient Neuromorphic Systems using Approximate Computing," in *ISLPED*, 2014.
[167] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO*, 2010.
[168] Y. Wang, A. Tavakkol, L. Orosa, S. Ghose, N. M. Ghiasi, M. Patel, J. S. Kim, H. Hassan, M. Sadrosadati, and O. Mutlu, "Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration," in *MICRO*, 2018.
[169] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G. Wei, "14.3 A 28nm SoC with a 1.2GHz 568nJ/Prediction Sparse Deep-Neural-Network Engine with >0.1 Timing Error Rate Tolerance for IoT Applications," in *ISSCC*, 2017.
[170] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," in *CVPR*, 2016.
[171] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *CVPR*, 2017.
[172] H. Yang, Y. Zhu, and J. Liu, "ECC: Platform-Independent Energy-Constrained Deep Neural Network Compression via a Bilinear Regression Model," in *CVPR*, 2019.
[173] L. Yang and B. Murmann, "Approximate SRAM for Energy-Efficient, Privacy-Preserving Convolutional Neural Networks," in *ISVLSI*, 2017.
[174] L. Yang and B. Murmann, "SRAM Voltage Scaling for Energy-Efficient Convolutional Neural Networks," in *ISQED*, 2017.
[175] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," in *CVPR*, 2017.
[176] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *ECCV*, 2018.
[177] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism," in *ISCA*, 2017.
[178] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: Enabling Aggressive Voltage Underscaling and Timing Error Resilience for Energy Efficient Deep Learning Accelerators," in *DAC*, 2018.
[179] J. J. Zhang, T. Gu, K. Basu, and S. Garg, "Analyzing and Mitigating the Impact of Permanent Faults on a Systolic Array Based Neural Network Accelerator," in *VTS*, 2018.
[180] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An Approximate Computing Framework for Artificial Neural Network," in *DATE*, 2015.
[181] 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 *MICRO*, 2016.
[182] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: A High-Bandwidth and Low-power DRAM Architecture from the Rethinking of Fine-grained Activation," in *ISCA*, 2014.
[183] X. Zhang, Y. Zhang, B. Childers, and J. Yang, "AWARD: Approximation-aWAre Restore in Further Scaling DRAM," in *MEMSYS*, 2016.
[184] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," *arXiv*, 2016.