Enabling Energy-Efficient, High-Performance DNN Inference Using Approximate DRAM

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

ETH zürich
SAFARI

1: Summary

Problem: Challenges of DNN inference:
- **High DRAM energy consumption** and **latency**

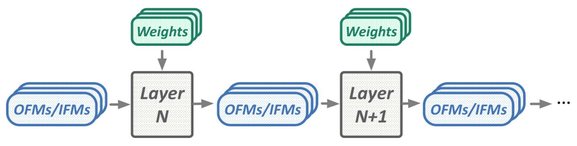
Goal: Reduce **DRAM oltage/timing scaled DRAM** for DNN inference to exploit **error tolerant DNN workloads**

EDEN: DNN Inference Using Approximate DRAM
- Techniques to maintain accuracy through **error tolerance boosting**, **DNN characterization, DNN to DRAM mapping,** and **DRAM error modeling**

Results:
- Average **21% DRAM energy savings**, **8% speedup** on CPU
- Average **37% DRAM energy savings** on GPU
- Average **31% DRAM energy savings** on DNN accelerators

EDEN is **applicable to other parameters and technologies**

2: DNN Basics and DRAM Parameters



- Modern DNNs can have **100+ layers**
- **3 main data types** in a DNN layer: **Weights, Input Feature Maps (IFMs), Output Feature Maps (OFMs)**
- Large # weights/IFMs enable **high learning capacity**
- If weights/IFMs have **bit errors,** the DNN can **still maintain accuracy**

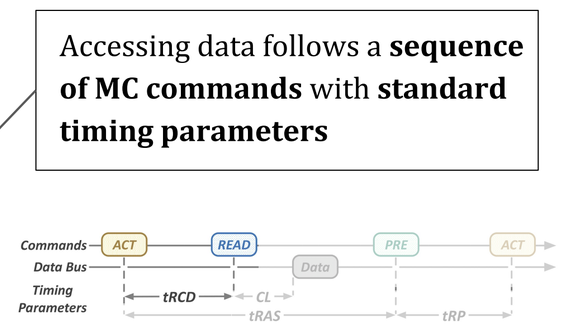
Memory Bus

Memory Controller

CPU, GPU, or DNN Accelerator

DRAM operates at a **standard voltage** (e.g., DDR3 at 1.35V)

Accessing data follows a **sequence of MC commands** with **standard timing parameters**



3: Overview of EDEN

Two Key Observations:

1. DNNs have an **intrinsic robustness to errors** in the weights and feature maps
2. We can **reduce DRAM energy consumption** and **latency** if we tolerate **more bit errors**

Three Step Mechanism:

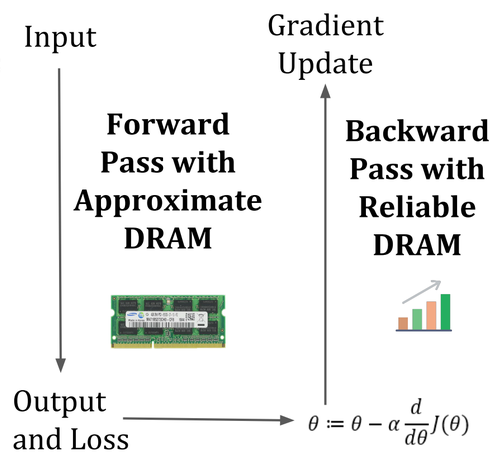
1. **DNN error tolerance boosting**
2. **DNN and DRAM characterization**
3. **DNN to DRAM mapping**

Approximate DRAM (voltage/latency-scaled DRAM) can provide **higher energy-efficiency** and **performance** for **error-tolerant DNN workloads**

4: DNN Error Tolerance Boosting

Goal:
Maintain accuracy when DNN is exposed to bit errors

Mechanism:
1. **Retrain** the DNN with **approximate memory** to adapt the DNN to unreliable cells
2. **Gradually increase the error rate** of the approximate DRAM to build error tolerance
3. **Filter out-of-range values** (e.g., $>10^{15}$) based on knowledge of the DNN weight and IFM distribution



5: DNN Characterization

Goal: Find the **highest tolerable error rates** of the DNN and the **corresponding DRAM parameters**

Mechanism: **Measure error resilience** of **each DNN data type** on the approximate DRAM

Two ways to characterize:

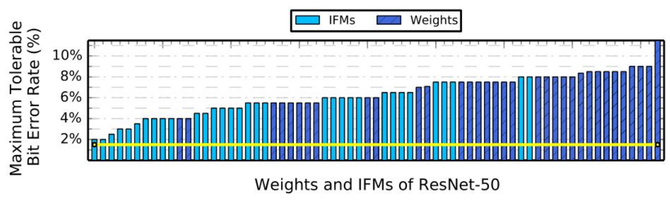
1. **Coarse-grained characterization:** Scale voltage/latency of all DNN data equally
2. **Fine-grained characterization** Different scaling of each DNN layer and data type

6. DNN to DRAM Mapping

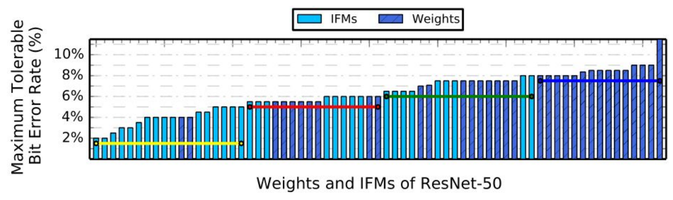
Goal: match error tolerance of DNN with DRAM **error rates**
Coarse-grained: assign the single best voltage/latency value that **meets the target DNN accuracy**
Fine-grained: a **greedy algorithm** that matches the most error sensitive DNN data to the most reliable DRAM partitions first

Example Characterization and Mapping of ResNet-50:

Coarse-Grained Mapping: A single DRAM partition with an error rate shown in yellow



Fine-Grained Mapping: Four DRAM partitions with error rates in yellow, red, green, blue

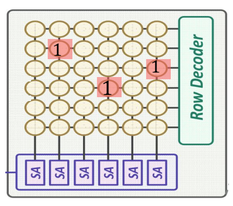


7: DRAM Error Models

Problem: Retraining is **not always feasible** on the approx. DRAM device

Goal: Perform retraining and error characterization **without use of the approximate DRAM device**

Model 0: **Uniform Random**
Model 1: **Bit Value Dependent**
Model 3: **Bitline Correlated**
Model 4: **Wordline Correlated**



Modification to Retraining: Forward Pass with the DRAM Error Model

8: Evaluation

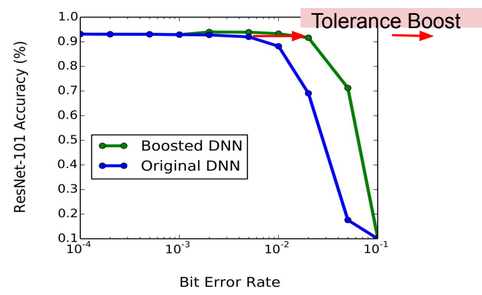
- **8 DNN workloads** with int4, int8, int16, FP32 quantizations
  - **ResNet101, MobileNetV2, VGG16, DenseNet201, SqueezeNet1.1, AlexNet, YOLO, YOLOTiny**
- Custom **PyTorch**-based DNN framework to run DNN inference with error models
- **SoftMC** framework[1] to run inference data accesses on **real DDR3 DRAM modules**
- **Ramulator, ZSim, GPGPUSim, and SCALE-Sim** used for DRAM, CPU, GPU, Eyeriss, and TPU simulation
  - Full configuration can be found in the paper
- Inference libraries from **DarkNet, Intel OpenVINO, TVM**

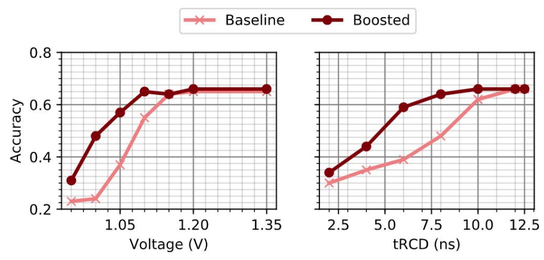
	FP32			int8		
Model	BER	ΔV_{DD}	Δt_{RCD}	BER	ΔV_{DD}	Δt_{RCD}
ResNet101	4.0%	-0.30V	-5.5ns	4.0%	-0.30V	-5.5ns
MobileNetV2	1.0%	-0.25V	-1.0ns	0.5%	-0.10V	-1.0ns

Sample of DNNs and their **tolerable BERs**

Example: Boosting Error Tolerance of **ResNet101**

DNN tolerance boosting **improves** a DNN's **bit error tolerance** by **5-10x**

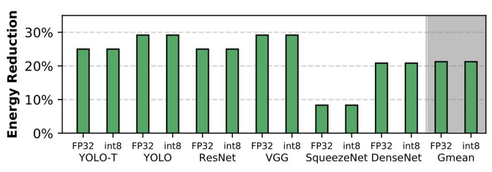




Example: DNN Accuracy of **LeNeT** on **SoftMC**

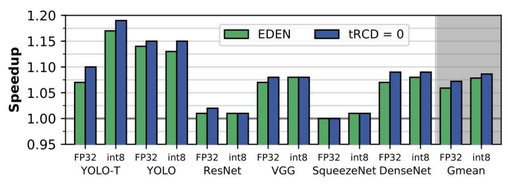
Boosting with error models helps **maintain accuracy** while **reducing voltage and latency** **on real DRAM modules**

DRAM Energy Reductions on CPU



Average 21% DRAM energy reduction maintaining accuracy within 1% of original

Performance Improvements on CPU:



Average 8% system speedup, with some achieving **up to 17% speedup**

DRAM Energy Reduction on GPU and Accelerators

- Average **31% on Eyeriss**
- Average **32% on TPU**
- Average **37% on Titan X**

Other Results in the Paper

- Error resiliencies across different DNNs/ quantizations
- Validation of the boosting mechanism
- Support for error models using real DRAM modules
- Comparison of different DRAM error models
- Breakdown of energy savings on different workloads for GPU and TPU

skoppula.github.io/pdfs/eden.pdf

GPUs and accelerators are **effective at hiding DRAM latency** due to (1) *effective pre-fetching* and (2) *large register banks and SRAM buffers* (exploiting the fixed memory access patterns on DNN inference), so we find little performance improvement reducing tRCD on GPU and DNN accelerators

Hassan, Hasan, et al. "SoftMC: A flexible and practical open-source infrastructure for enabling experimental DRAM studies." *HPCA* 2017.

SAFARI