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

EDEN

presented by Nina Richter

Venue

- MICRO-52 (52nd IEEE/ACM International Symposium on Microarchitecture)
- October 12–16, 2019

Outline

- EDEN summary
  - Background
    - Neural networks & approximate DRAM
  - Problem: Memory intensity of DNN
  - EDEN framework
  - Results of testing EDEN
- Critique
- Thoughts & takeaways
- Discussion

EDEN

Executive Summary

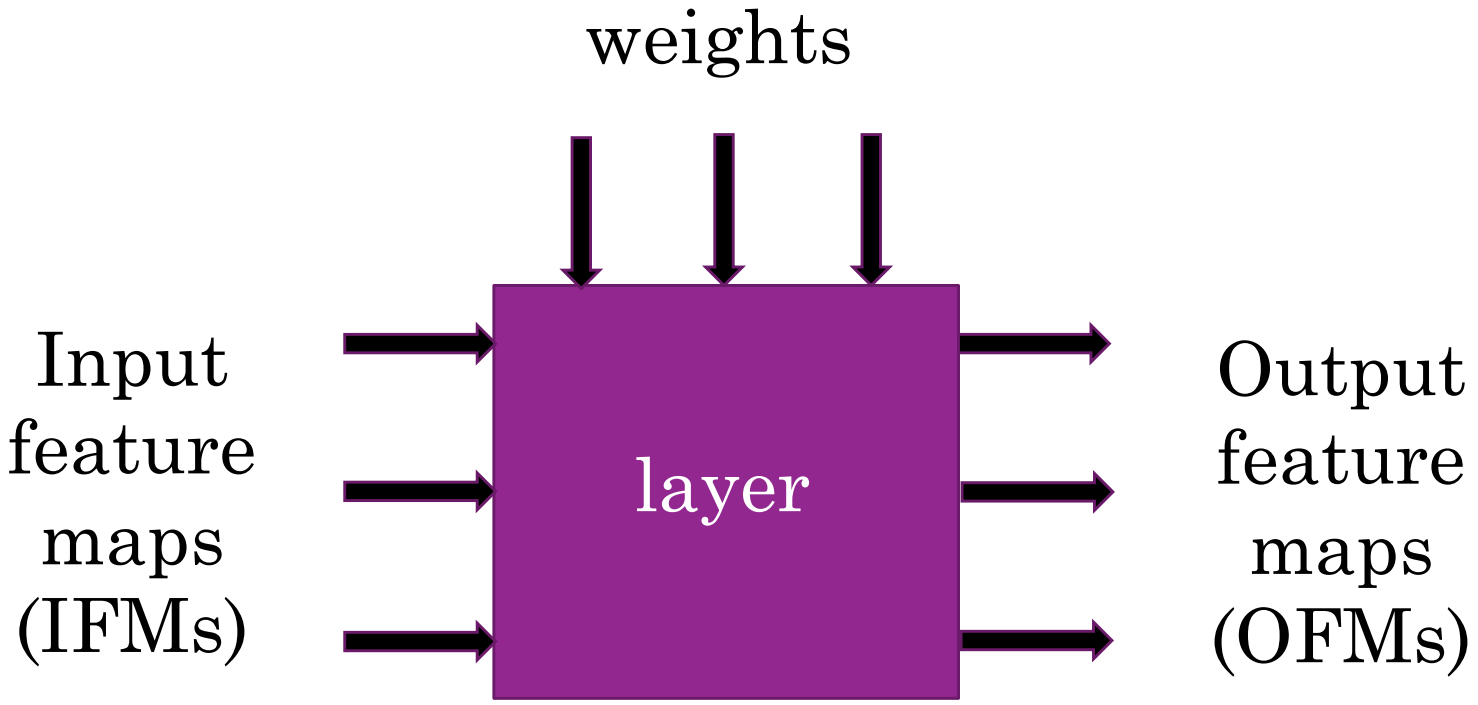
DNNs have a high memory footprint, resulting in a high need for energy and time.

Using approximate DRAM can relieve both these issues.

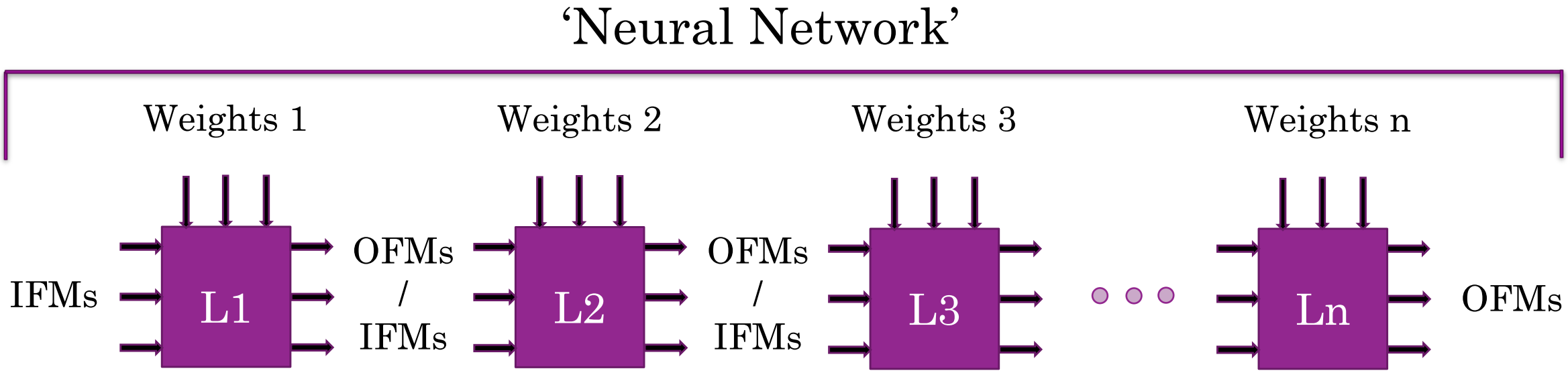
EDEN provides a framework to enable DNNs to handle the bit errors approximate DRAM causes.

BACKGROUND

Neural Network



DNN (Deep Neural Network)



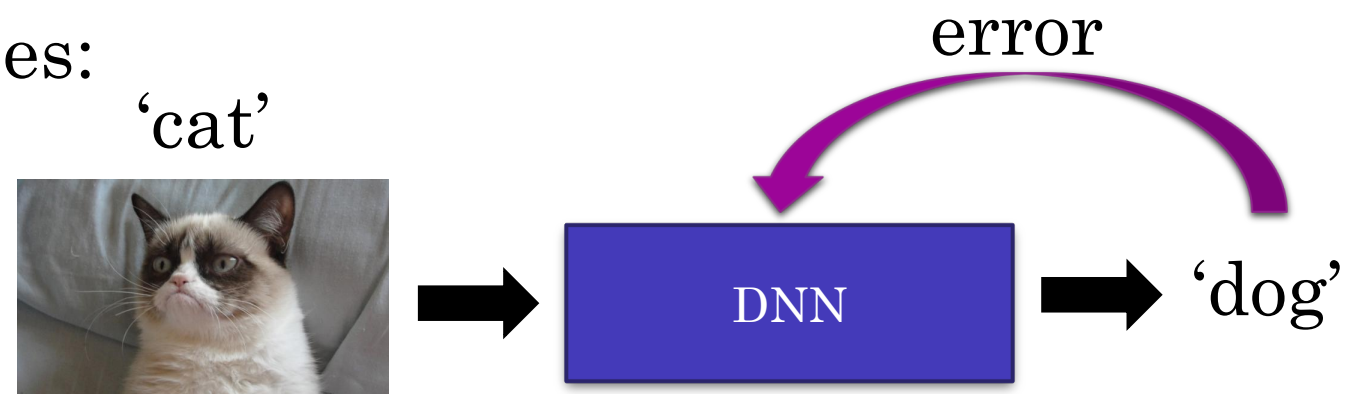
DNN = Neural Network with >2 layers

modern DNNs: 100s of layers!

How to Use a DNN

Two main phases:

- Training



- Inference

DNN Training

Goal: determine best weight matrices, so that DNN generates wanted output

- Training dataset = example data & correct outputs
- Forward pass = computing output of example input
- Backward pass = sending error compared to correct output back through layers, adjust weights accordingly
- Load/stores from main memory for all data types, and for both passes

DNN Inference

Goal: use the DNN to generate new output

- Only forward pass, faster than training
- Percentage of correct outputs = accuracy of DNN
- Load/stores from main memory for all data types

Approximate DRAM

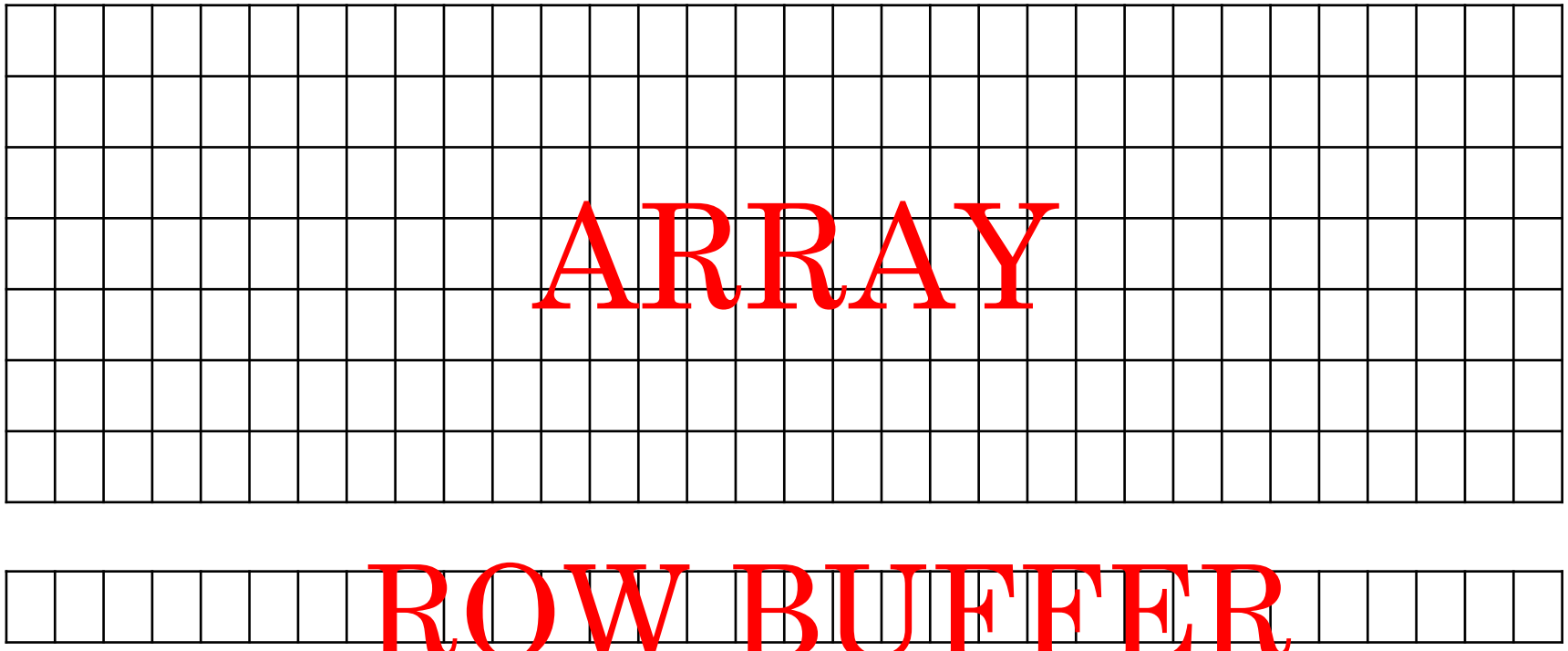
‘DRAM operating under non-nominal parameters’

Result: DRAM not reliable

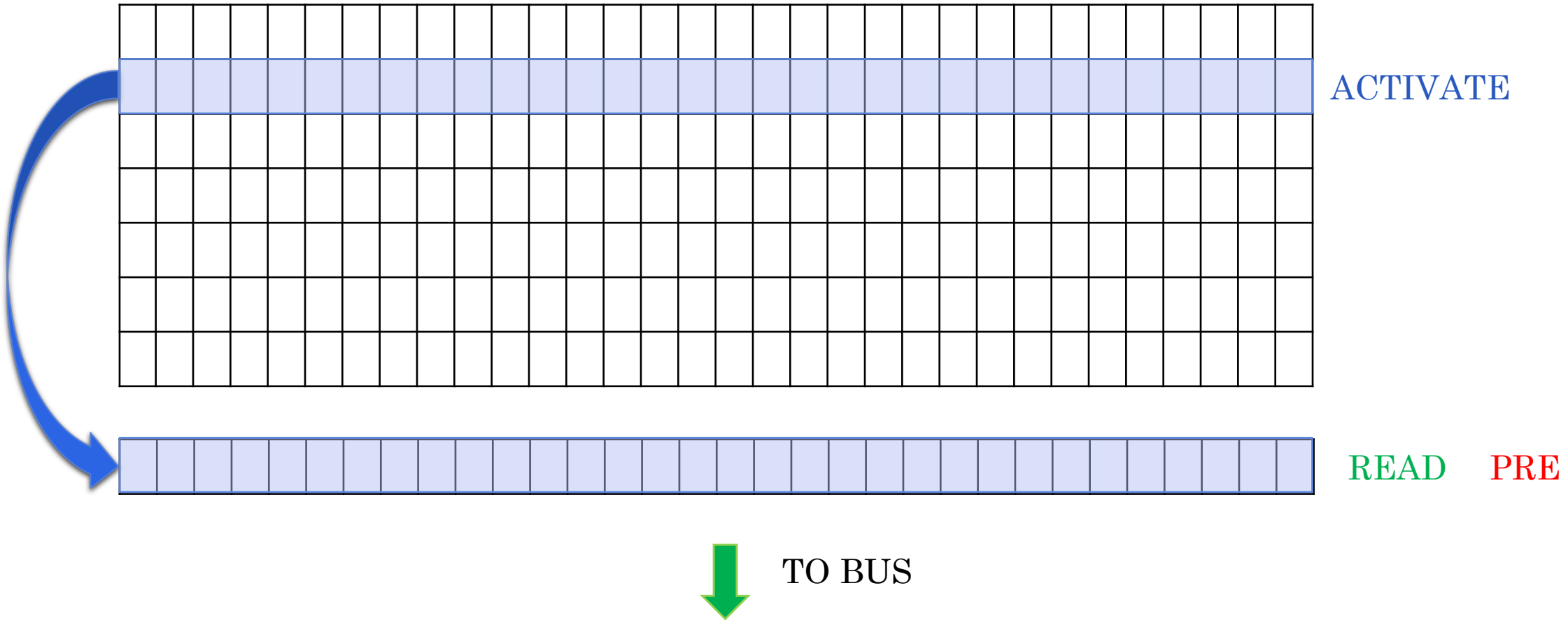
Motivation: exploit performance/reliability tradeoff

How does DRAM ensure reliability?

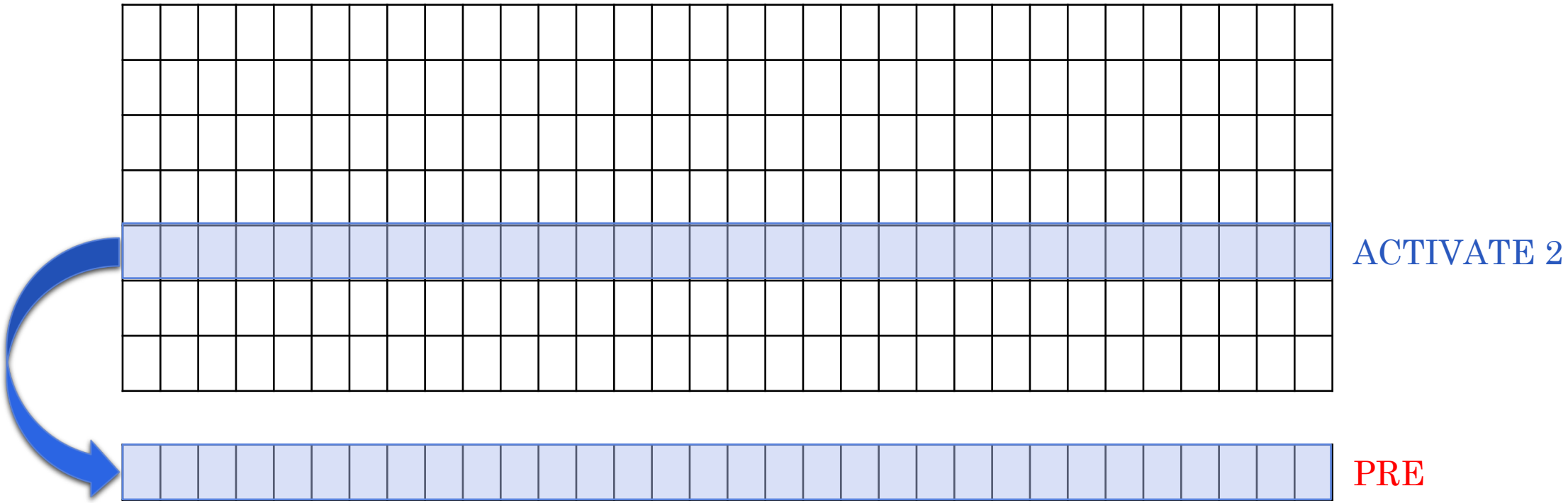
Reliable DRAM



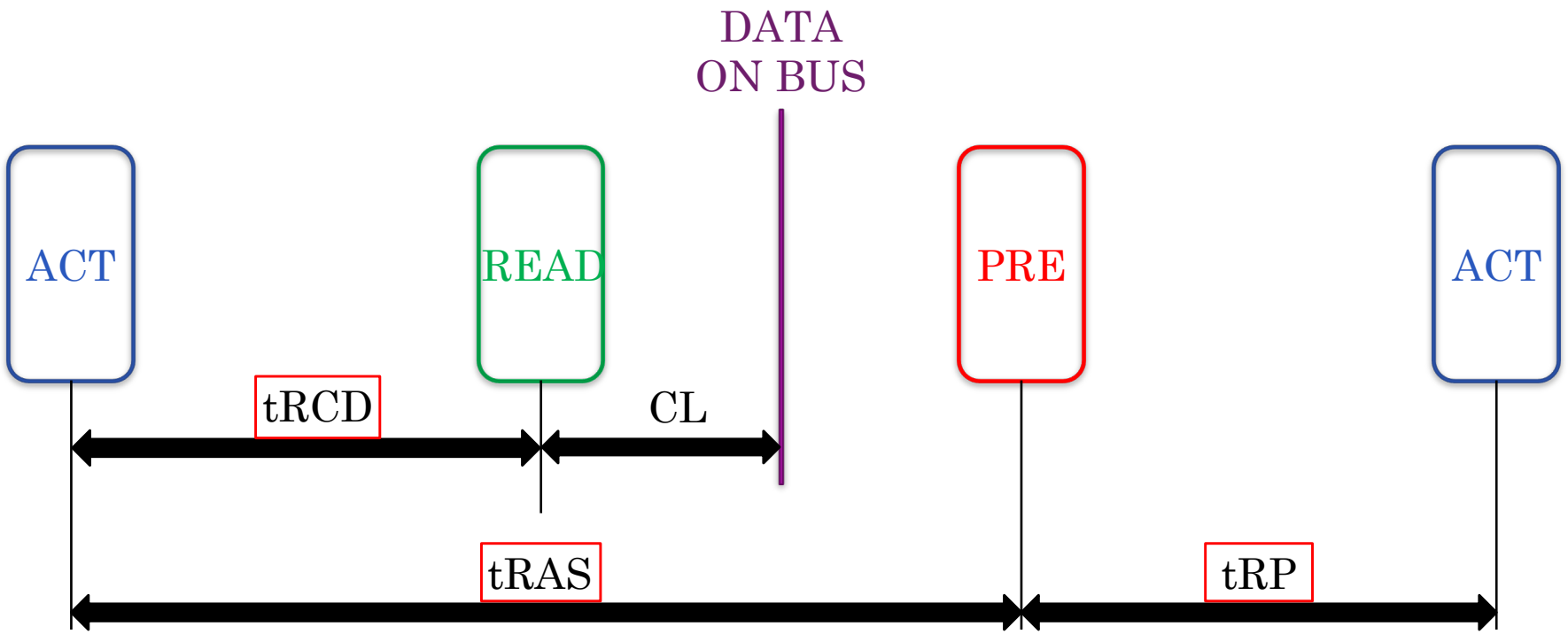
Reliable DRAM



Reliable DRAM



Timing Specifications



Energy influence: Voltage down means latency up!

Where do Bit Errors happen?

Answer: in ‘weak cells’ of the data array

Every cell has a certain probability for bit flips

In weak cells this probability is much higher

High probability caused by two kinds of factors:

- Intrinsic (‘created at production’)
- Extrinsic (‘created by circumstance’)
  - Examples: device age, temperature, cell value

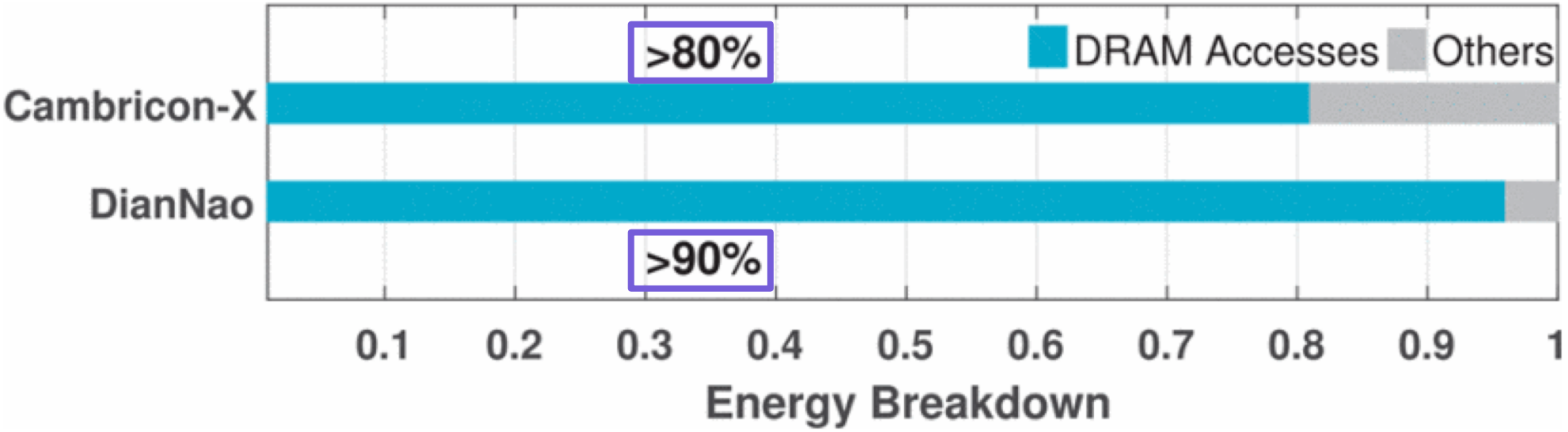
THE PROBLEM

DNN Memory Demand

- DNNs have a high and ever-growing memory demand
- Memory accesses need
  - Time
  - Energy

Due to the increasing memory intensity of most DNN workloads, main memory can dominate the system's energy consumption and stall time.

Example 1: DRAM Energy Consumption



Graph from SmartShuttle [1]
Energy used by 2 state-of-the-art accelerators

Example 2: DNN Growth

- ILSVRC image recognition challenge
  - Winning model 2017: 837M of FP32 parameters (ResNeXt, 3.3GB)
  - **13.5x** parameter count of winning model 2012 (AlexNet)
- Some newer models: **>1B** parameters

Observations

EDEN is based on two insights:

- DNNs are error tolerant
  - Input, weight, output data types
- DRAM trades performance for reliability

Key Idea

Goal: reduce time & energy demand of memory

Method: Use approximate DRAM

- Reduced voltage
- Shortened timing parameters

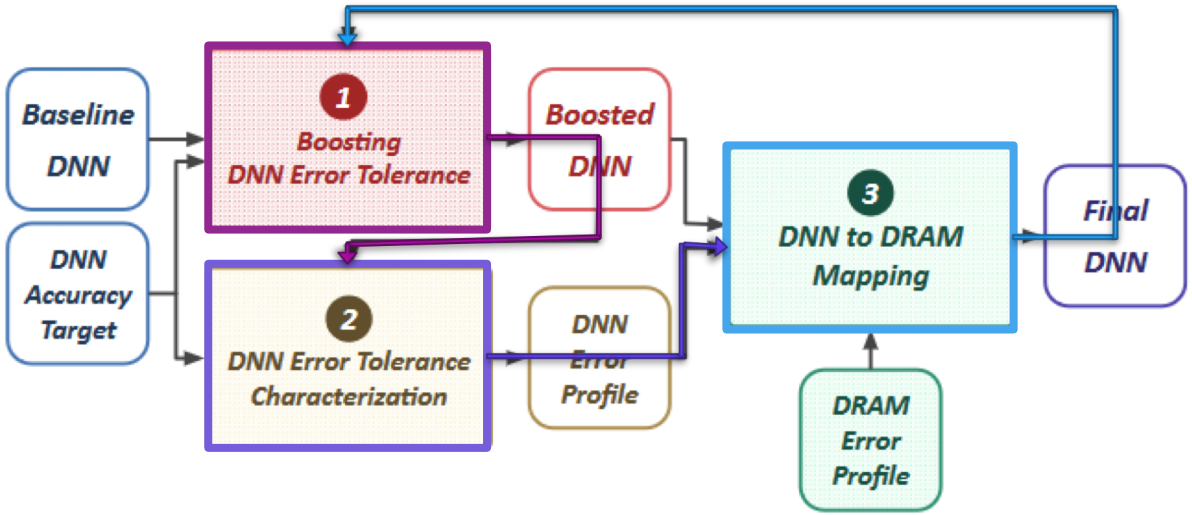
Challenge: switching to approximate DRAM and increasing the error rate without other changes causes accuracy collapse!

EDEN FRAMEWORK

EDEN Main Components

- DNN retraining
- DNN profiling
- Mapping of DNN data types to approximate DRAM partitions

EDEN iterates over these steps:



DNN Retraining

Goal: improve error tolerance

Method: 'curricular retraining':

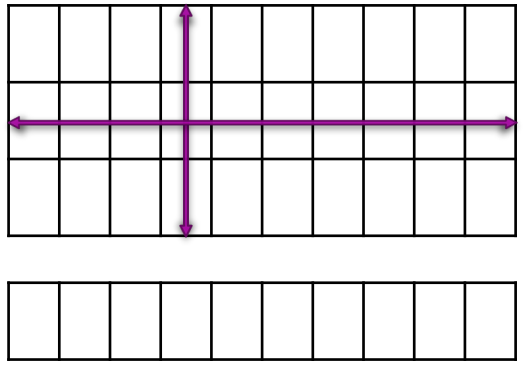
Slowly increase error rate (fast increase causes accuracy collapse!)

- Forward pass using approximate DRAM
- Backward pass using reliable DRAM
- Increase error rate by one step after whole training dataset passed

What if we can't use our DRAM Device for Retraining?

'EDEN offloading': use another system for retraining, inject errors based on weak cell locations of target device

How do we know where weak cells are?



DRAM error models: weak cells follow

- Uniform random distribution
- Clustered along bitlines
- Clustered along wordlines
- Uniform random distribution, depending on cell value

DNN Profiling

What can our DNN handle? (and keep target accuracy)

- Coarse grained
  - Same Bit Error Rate (BER) for whole DNN
- Fine grained
  - Cluster Data Types by BER tolerance

Map DNN to DRAM

- Needs BER characterization of DRAM device

- Coarse grained mapping
  - Same voltage/timing parameters for whole DRAM (BER tolerable by whole DNN)
- Fine grained mapping
  - Partition DRAM
  - Map DNN Data Types to partitions
  - Each partition with own parameters

One Underlying Issue

Problem: some bit flips change the original value much more than others!

Insight: most data types have a small range of 'viable' values

EDEN's Solution:

- Define 'acceptable range' for all data types
- Compare actual value to this range on every memory access (both training and inference!)
- Correct 'implausible values' by setting them to zero

Hardware Support

Correcting implausible values

Coarse grained mapping

- Runtime adjustments of DRAM parameters

Fine grained mapping

- Configure timing parameters/track latency of partitions → 1KB metadata for 2^{10} partitions

Memory controller

- Enable & track voltage scaling (bank granularity) → ≤32B metadata

Power delivery network[1]

[1]based on Voltron[2]

RESULTS

Methodology

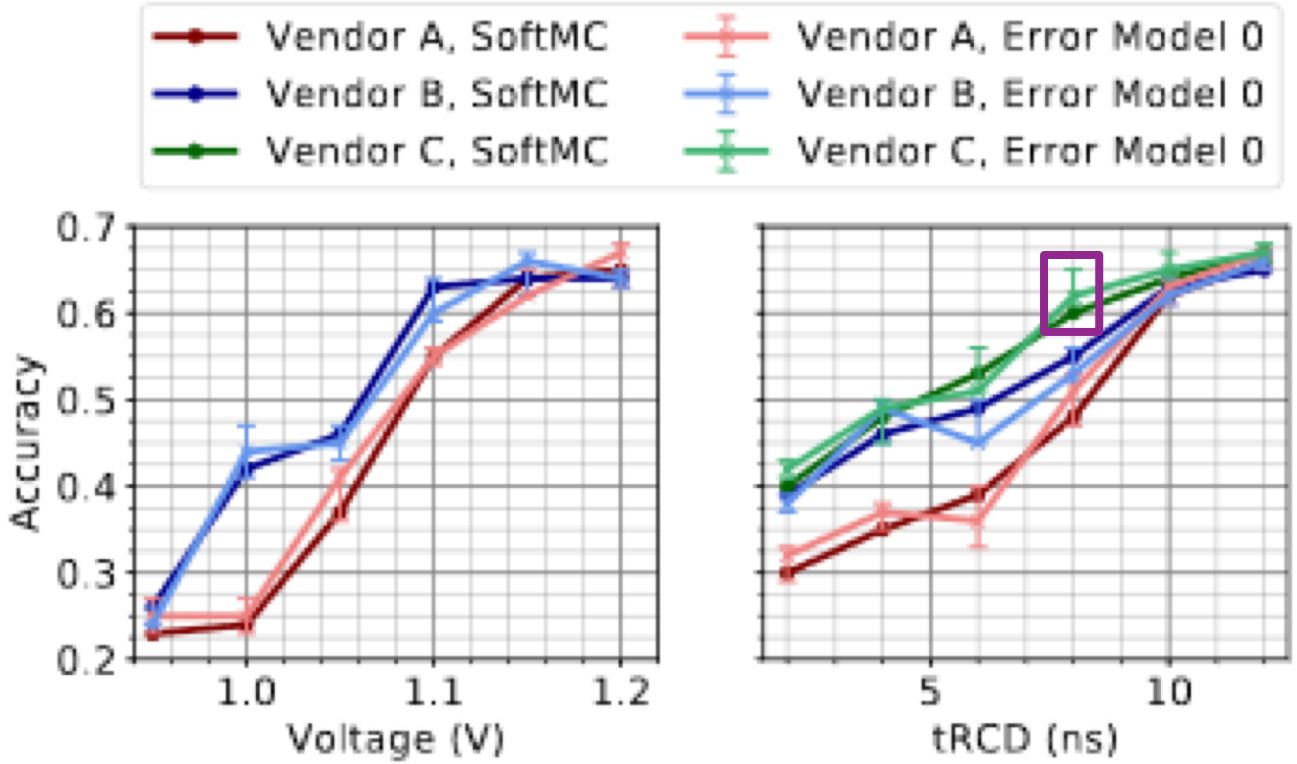
FPGA based infrastructure (SoftMC)

- Room temperature
- Real approximate DRAM: reduce voltage & tRCD
- limitations caused by SoftMC

Framework on top of PyTorch

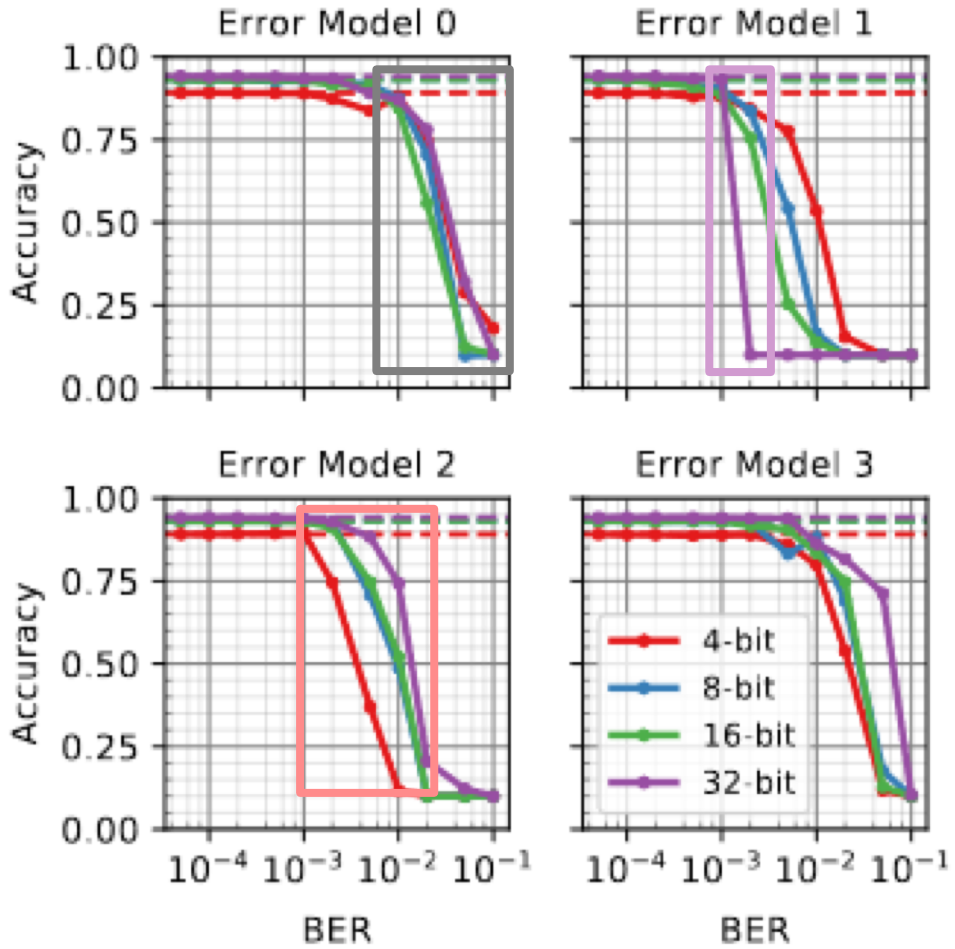
- Using Error Models
- 80-90x faster DNN accuracy estimation than SoftMC

Error Model Evaluation



- Error simulation (lighter colored curves) closely follows measured behavior of modelled DRAM device
- Error bars for 95% confidence

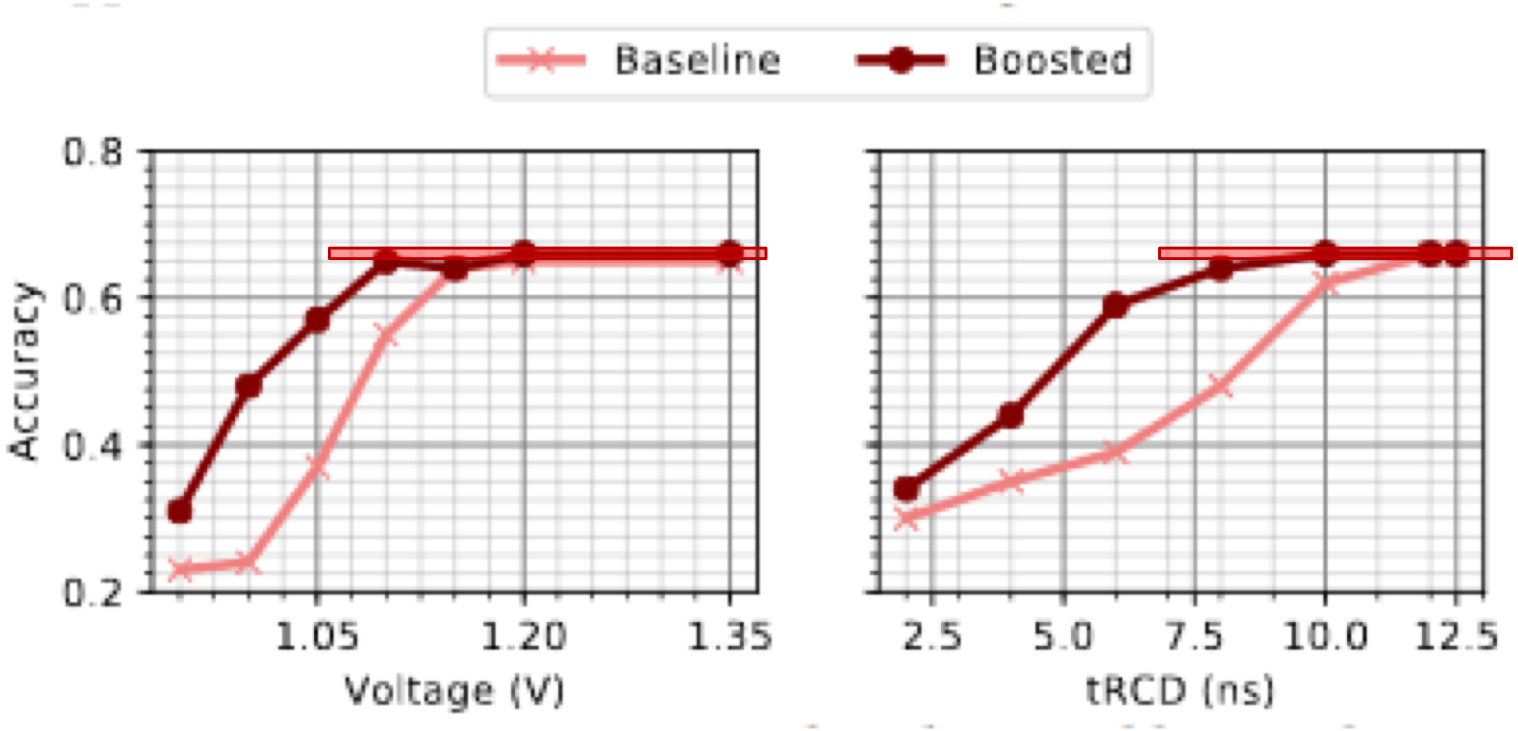
Error Models & Quantization



DNN: ResNet101

- Model 1 (weak cells along bitlines): bad for FP32
- Model 2 (along wordlines): bad for INT4
- Model 0 (random distribution): similar for either quantization
- Conclusion: different DRAM devices may offer best gains for different DNNs

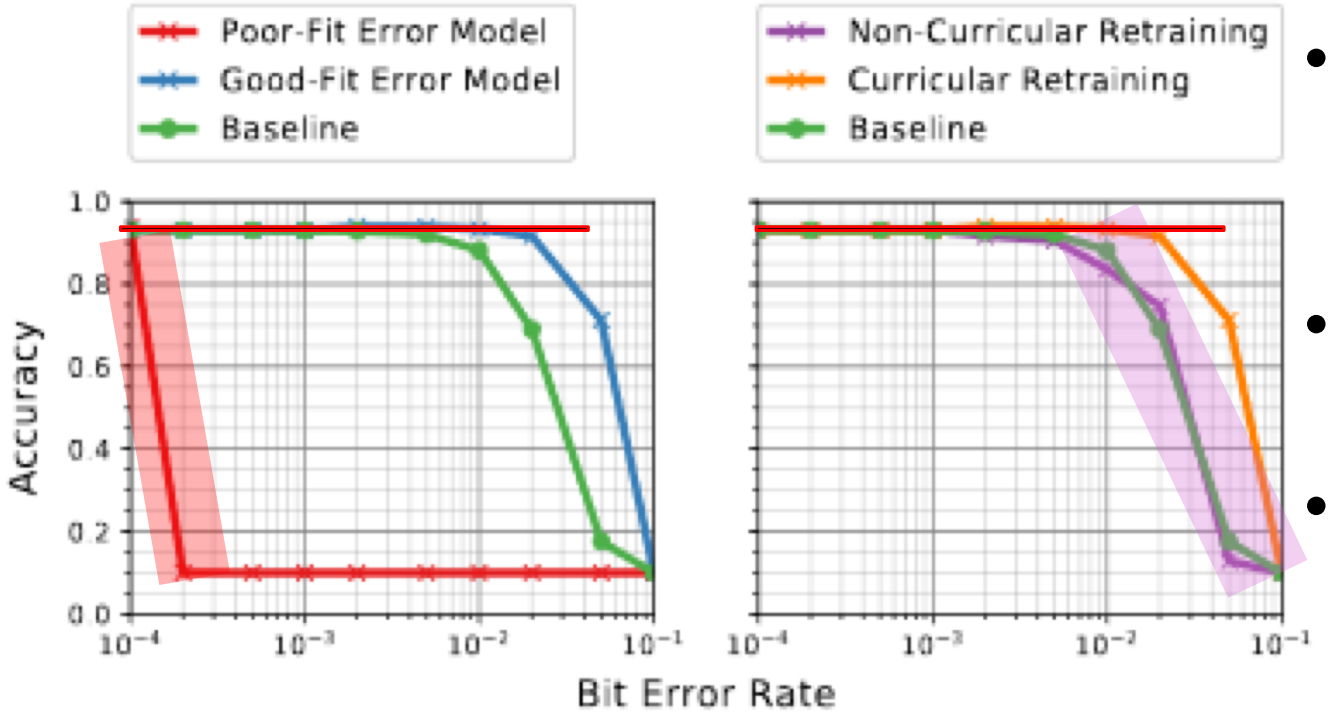
Performance with EDEN



- Boosted curve (dark red) extends far more to the left before accuracy collapse occurs
- Clear improvement!

DNN: LeNet

Accuracy Collapse in Retraining



DNN: ResNet101

- Left graph: Poor-fit error model causes accuracy collapse
- Right graph: Regular retraining has no effect
- Both graphs: well used EDEN boosts error tolerance

Coarse Grained Profiling & Mapping

	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
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

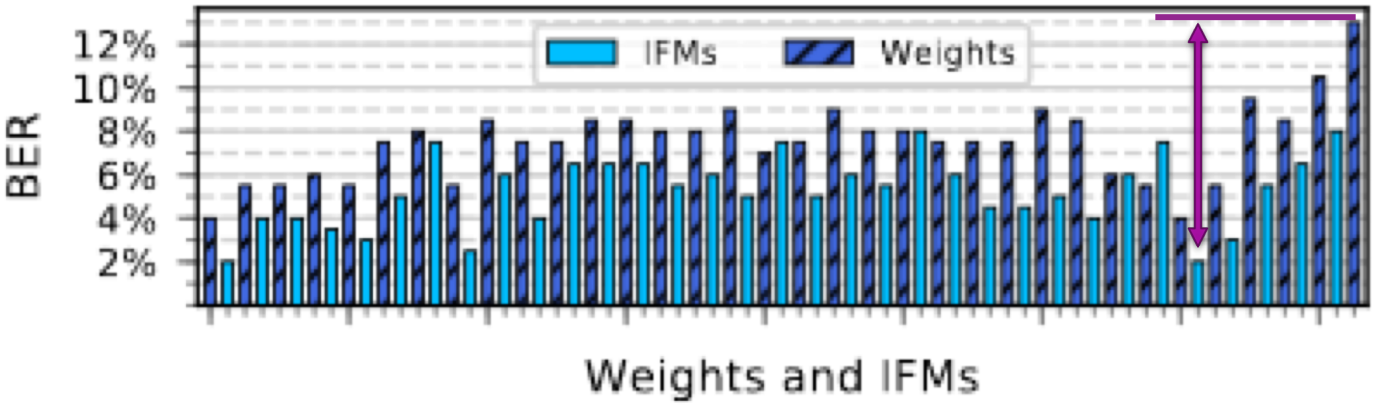
<1% accuracy degradation

Nominal values:

V_{DD}=1.35V, tRCD=12.5ns

- Same DNN, different quantization: (almost) the same values
- Different DNN, same quantization: high variation

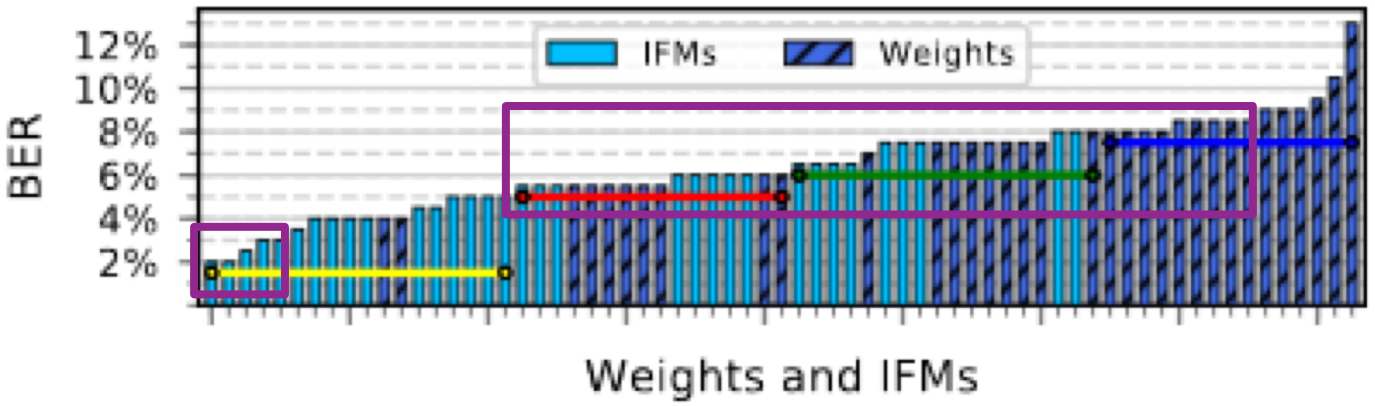
Fine Grained Profiling



DNN: ResNet101
Deeper levels to the right
<1% accuracy degradation

- Weights (dark blue) generally more tolerant than IFMs
- High variance in tolerance between data types (2% - 13% BER)

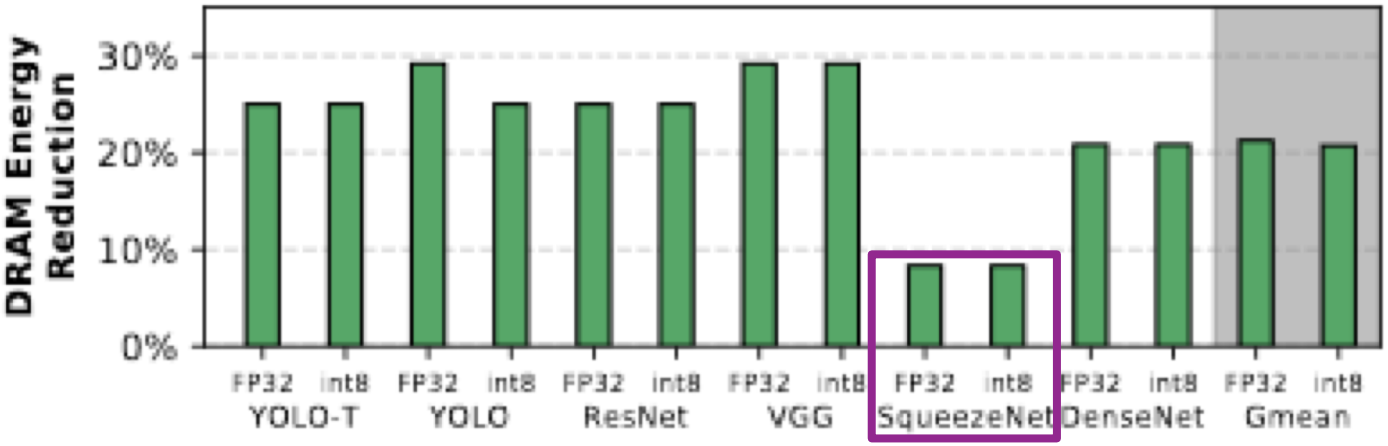
Fine Grained Mapping



DNN: ResNet101
<1% accuracy degradation

- Only 4 partitions (horizontal lines), but ~3/4 of all data types get nearly ideal assignment!

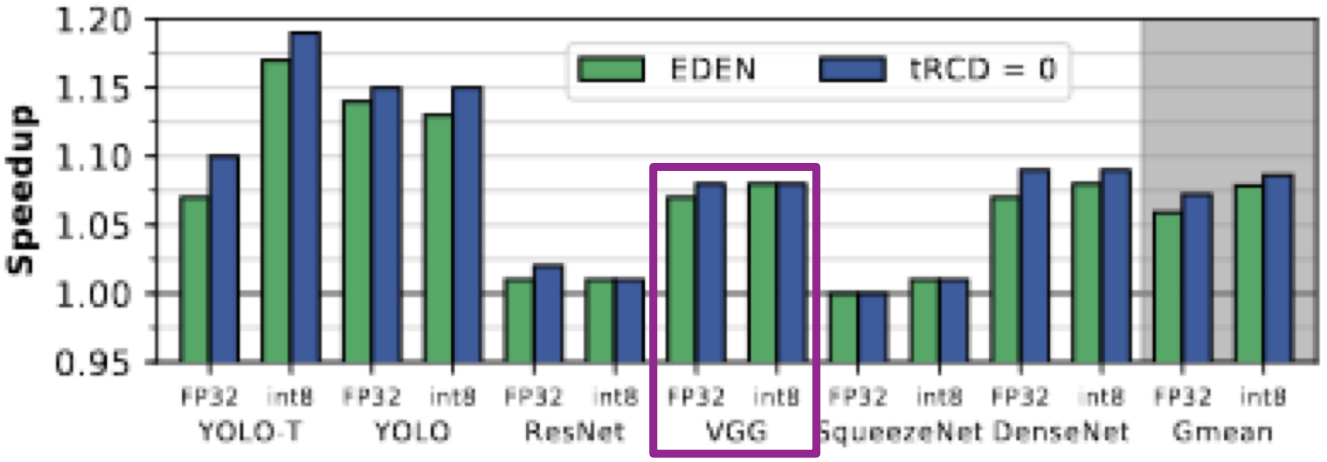
CPU Inference: Energy Reduction



- Average: 21%
- Max: 29%
- Not as much gain for energy-optimized DNN (used in embedded systems)

<1% accuracy degradation
Coarse grained mapping

CPU Inference: Speedup



<1% accuracy degradation
Coarse grained mapping

- Average: 8% EDEN (10% ideal tRCD)
- Max: 17% EDEN
- EDEN gets extremely close to ideal, equal in some cases!

Maintain Accuracy of Baseline DNN

Average energy saving: 16%
Maximum energy savings: 18%

Average speedup: 4%
Maximum speedup: 7%

Accelerators for Inference

- <1% accuracy degradation
- GPU
  - Higher energy savings than CPU (32.6% to 41.7%)
  - Lower speedup (0% to 5.5%)
- Eyeriss/TPU
  - Higher energy savings than CPU (31% to 34%)
  - No speedup in any tested case

Conclusion

- EDEN can efficiently boost a DNN to tolerate high BERs in an approximate DRAM device
- The boosted DNN enables high energy/latency reductions compared to the baseline DNN
- The possible reductions are influenced by both the specific DNN and the system it is run on
- EDEN does not care about the exact cause of bit errors (tested: voltage & tRCD scaling), and can be used to reduce other parameters as well

CRITIQUE

Strengths

- Innovative approach to known problem
- Needs only small adjustments to run on (almost) any conventional system
  - Other approaches (e.g. processing-in-memory) much more invasive
- Framework is adaptable to improve other DRAM parameters than energy/latency who trade off with reliability (example: refresh rate)
- Framework is adaptable for other memory systems which trade off some parameters with reliability

Subjective Strength

I know some of the authors!

They’re all nice people^^

Weaknesses

- Needs extensive knowledge about BER of (partitions of) changing DRAM device
- Specialized for a single device, need to retrain again for any new device
- Speedup only if DNN is memory latency-bound on the used system
- Fine grained mapping not tested in inference

THOUGHTS & TAKEAWAYS

Thoughts

- Experiments using fine grained characterization/mapping?
- Compare fine grained to coarse grained?
- Change other/multiple timing parameters? (tRAS must have changed along with tRCD, not mentioned)
- What if I have another DRAM device with very similar error patterns?
  Start retraining from the boosted DNN?
  Better results starting from scratch?

Takeaways

- DRAM (and other hardware components) trade performance for reliability
- DNNs can greatly benefit from approximate DRAM
- Correcting implausible values / accelerators: specialized hardware to speed up software important for both ‘big’ and ‘small’ issues!

DISCUSSION

Questions

Other Applications

What other applications could benefit from using approximate DRAM?

- Internet telephone: sound quality not too important, use approximate DRAM to reduce energy needed by TCP/UDP buffer?
- Same for video streaming?

Using DRAM Adjustment Techniques Differently

DRAM parameters can handle some adjustment before any errors occur: what about a system that dynamically adapts them to the current reliable optimum?

- How often does re-characterization have to occur?
- How can we test reliability at runtime?
- How conservative do we still have to be?

Other Solutions

When might other methods to solve the DNN memory intensity problem (e.g., processing-in-memory) be more efficient (for some variable) than EDEN?

- Processing-in-memory if we use many different DNNs that are run on the same system, updating system cheaper than updating all DNNs? (TETRIS [3])
- Exploit latency differences between different DRAM cells, maybe additionally? (FLY-DRAM [4])

THANK YOU FOR YOUR ATTENTION

SEE YOU IN THE MOODLE DISCUSSION :)

[1] SmartShuttle

J. Li *et al.*, "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2018, pp. 343-348

SmartShuttle: Optimizing Off-Chip Memory Accesses for Deep Learning Accelerators

Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, Xiaowei Li
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
{lijiajun, yan, luwenyan, jiangshuhao, gongshijun, wujingya, lxw}@ict.ac.cn

https://ieeexplore.ieee.org/document/8342033

[2] Voltron

K. K. Chang et al. Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1, Article 9 (June 2017), 42 pages

Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms

KEVIN K. CHANG, A. GİRAY YAĞLIKÇI, and SAUGATA GHOSE, Carnegie Mellon University
ADITYA AGRAWAL and NILADRISH CHATTERJEE, NVIDIA
ABHIJITH KASHYAP, Carnegie Mellon University
DONGHYUK LEE and MIKE O'CONNOR*, NVIDIA
HASAN HASSAN and ONUR MUTLU†, ETH Zürich

https://dl.acm.org/doi/pdf/10.1145/3084447?download=true

[3] TETRIS

M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," ASPLOS, 2017

TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory

Mingyu Gao Jing Pu Xuan Yang Mark Horowitz Christos Kozyrakis

Stanford University

{mgao12,jingpu,xuany,horowitz,kozyraki}@stanford.edu

https://web.stanford.edu/~mgao12/pubs/tetris.asplos17.pdf

[4] FLY-DRAM

K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhi-menko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in SIGMETRICS, 2016

Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization

Kevin K. Chang[1] Abhijith Kashyap[1] Hasan Hassan[1,2]
Saugata Ghose[1] Kevin Hsieh[1] Donghyuk Lee[1] Tianshi Li[1,3]
Gennady Pekhimenko[1] Samira Khan[4] Onur Mutlu[5,1]
[1]Carnegie Mellon University [2]TOBB ETÜ [3]Peking University [4]University of Virginia [5]ETH Zürich

https://dl.acm.org/doi/pdf/10.1145/2964791.2901453?download=true

Tested DNNs

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* [89]	CIFAR10 [4]	1.65MB	2.30MB

* we use this small model in some evaluations where the experimental setup does not support large models.

Baseline Accuracy

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.

CPU System Configuration

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

ZSim & Ramulator for simulation of core/DRAM

GPU System Configuration

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

GPGPU-Sim for Simulation

Accelerator System Configuration

	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

Scale-Sim for Simulation