慶應義塾
Keio University

Fault-aware Hardware Scheduling of Computations in Deep Neural Networks

Shaswot Shresthamali, Yuan He, Masaaki Kondo

Kondo Laboratory
Keio University

SWoPP 2022
Shimonoseki

29-July-2022

Overview – tl;dr version

1. Deep Neural Network (DNN) accelerators have become very popular for efficient DNN implementation.
2. DNN accelerators may (will) introduce errors in computation
3. DNNs can tolerate some computational inexactness and degrade gracefully.
4. It is possible to cleverly schedule DNN computations in inexact hardware so that performance degradation is minimized
5. This work presents Hardware Agnostic Scheduler (HAS) - which can recover significant DNN accuracy (up to 30%) even when DNNs are implemented in compromised DNN accelerators
   - **Central Idea**: Shuffle rows during matrix multiplication to schedule non-important computations to compromised hardware
   - Use GA search to find best shuffling order

Introduction

Faulty Accelerators

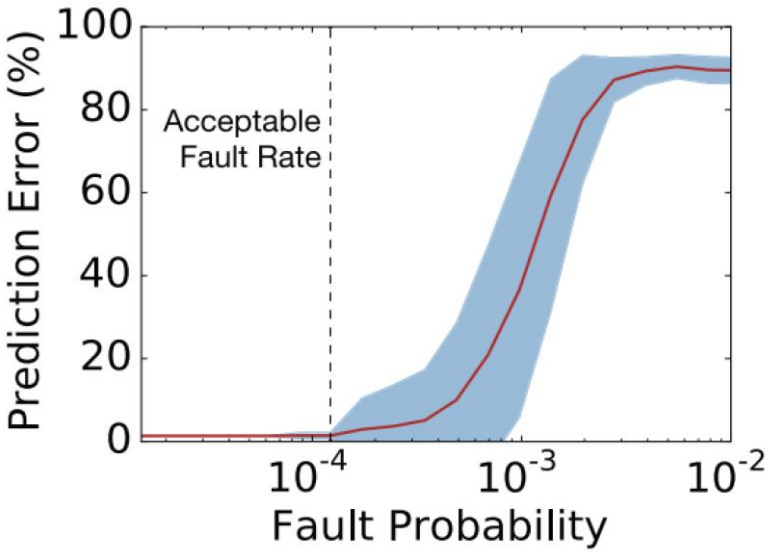
- Accelerators are faulty → errors in computations

- Uncontrollable Faults
  - Manufacturing defects
    - Yield and reliability decrease with smaller process technology and larger wafers
    - Testing is becoming very expensive (and unreliable)!
  - Degradation with age
    - More hardware PEs → shorter Mean Time Between Failures (MTBF)

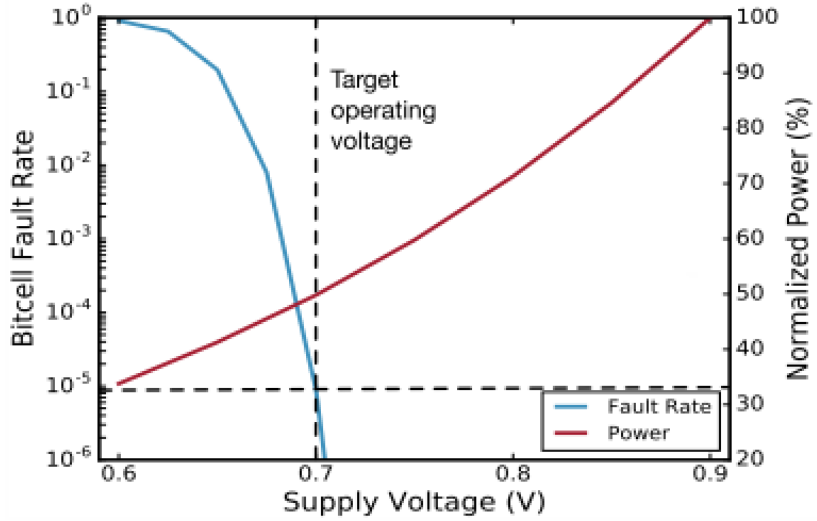
- Controllable Faults
  - Marginal operation
    - Lowering supply voltage for energy efficiency
  - Approximate computing
    - Approximate arithmetic – e.g. linear approximations
    - Approximate hardware – e.g. reduced precision

DNNs are resilient to errors

- DNNs have overprovisioned parameters
- Computations within a layer are
  - independent
  - distributed
  - parallel
- DNNs degrade gracefully with increasing error probabilities
- Some degradation of model accuracy is tolerable to extract
  - higher energy efficiency (lower Vdd)
  - lower latency (reduced precision)



Reagen et al. - 2016 - Minerva Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators



Related Works

- Approach: Leverage DNN resilience for
  - Power Savings
  - Lower Latency/ Higher Throughput

- *MINERVA, ARES*
  - Reduce SRAM voltage and increase Bit Error Rate (BER)
  - Detect and correct errors in PEs (Processing Elements)

- Approximate Computing (*AxNN, AxTrain, ApproxANN*)
  - Use inexact arithmetic for low priority neurons
  - Requires neuron sensitivity analysis and retraining

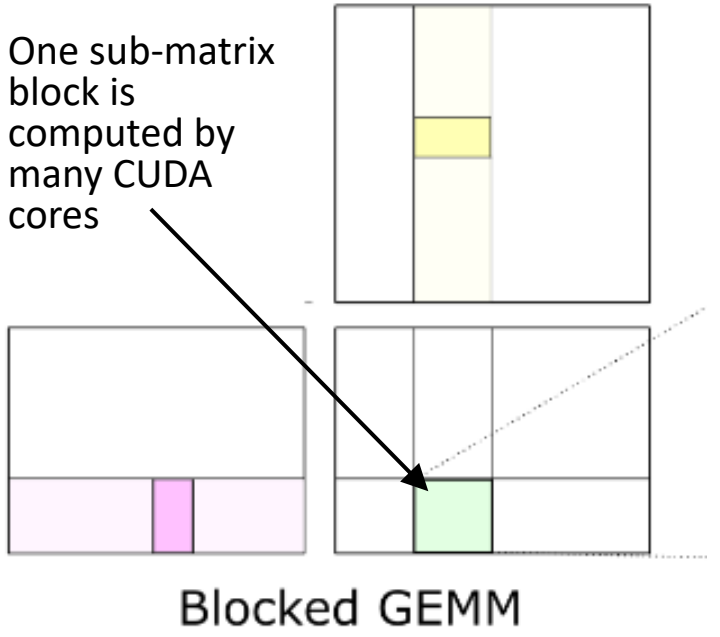
- Reduced/mixed precision computation (*HAQ, RAPiD*)
  - Use hardware-in-loop optimization method
  - Specialized accelerator architecture for mixed precision

Limitations of previous approaches

- Previous works focus on analyzing *neuron sensitivity*
  - Allocate non-critical neurons to compromised hardware

- In reality, neurons (computations) are spread out among many PEs during parallelization
  - One neuron is computed using multiple PEs
  - A PE maybe reused for computation of multiple neurons
  - One-to-one mapping between computations and PEs does not exist.
- A neuron is a *computational concept.*

- Unimportant **computations** need to be scheduled to compromised hardware
  - Without interfering with accelerator optimizations – cache misses/shared memory/banking conflicts etc..



https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/

Motivation

- Trained DNN models are deployed in various hardware platforms
  - Edge devices in IoT systems
  - Different accelerator types – GPUs, TPUs etc.
- Hardware platforms have different fault profiles
  - Fault profile – where do faults occur with what probability
- Recovering performance by retraining, reoptimizing is not practical
- We need a general method to reschedule computations (for performance recovery) that
  - treats DNN as a black box
  - is agnostic to the type of hardware platform

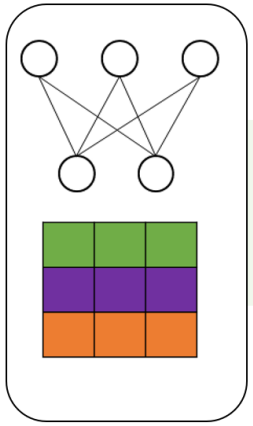
Hardware Agnostic Scheduler (*HAS*)

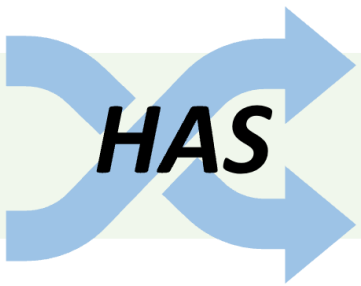
Proposed Solution: *HAS*

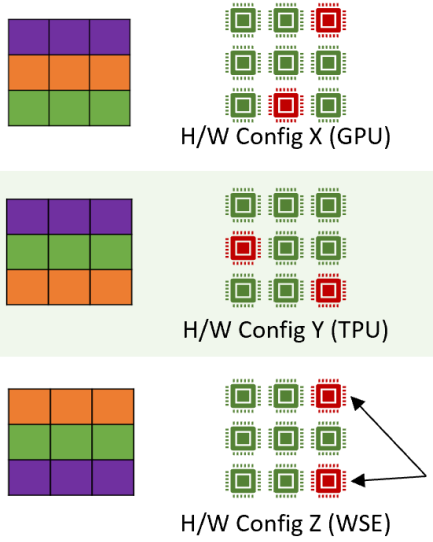
- Focus only on DNN inference in this work
- Schedule important ***computations*** in reliable hardware by shuffling the rows of the matrix
- n rows → $n!$ permutations [search space is too large]
  - Use GA (Genetic Algorithm) to find the best shuffling order

HAS shuffles the matrix rows.
The best shuffle order is found using *GA*.

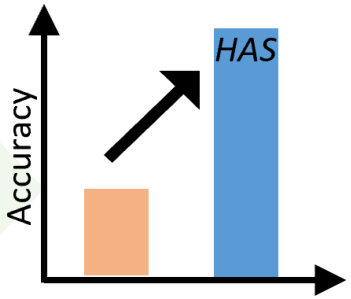
Inference involves many matrix multiplications







Naïve Scheduling in faulty hardware causes severe degradation.

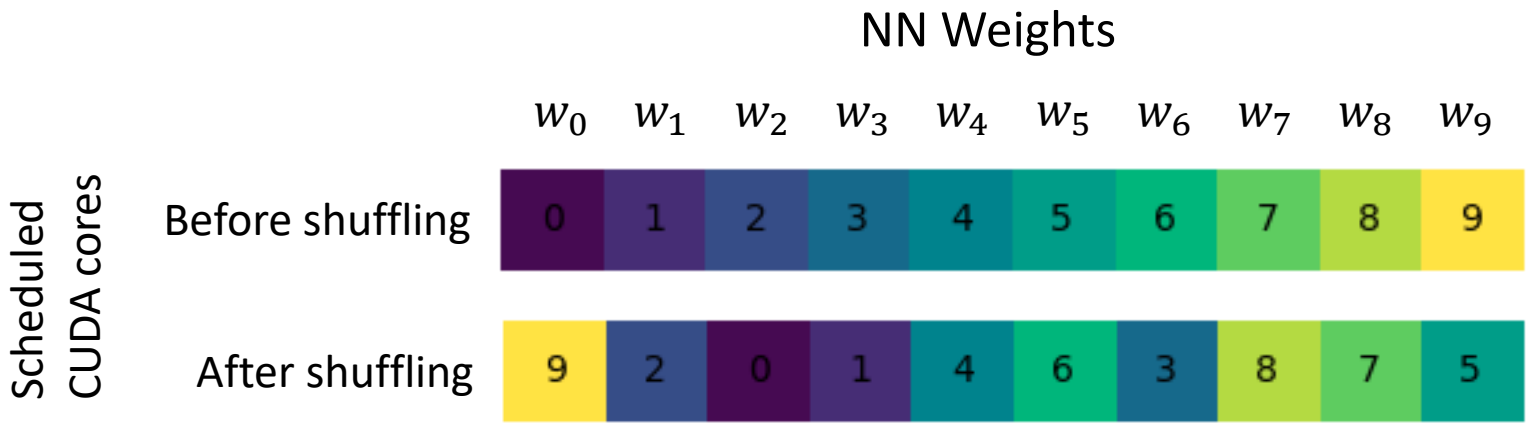


HAS can recover significant accuracy even with faulty hardware

GA Problem Statement

- Given a hardware fault-profile,
  - what is the best shuffling order for the rows of the weight matrix so that the performance degradation is minimized

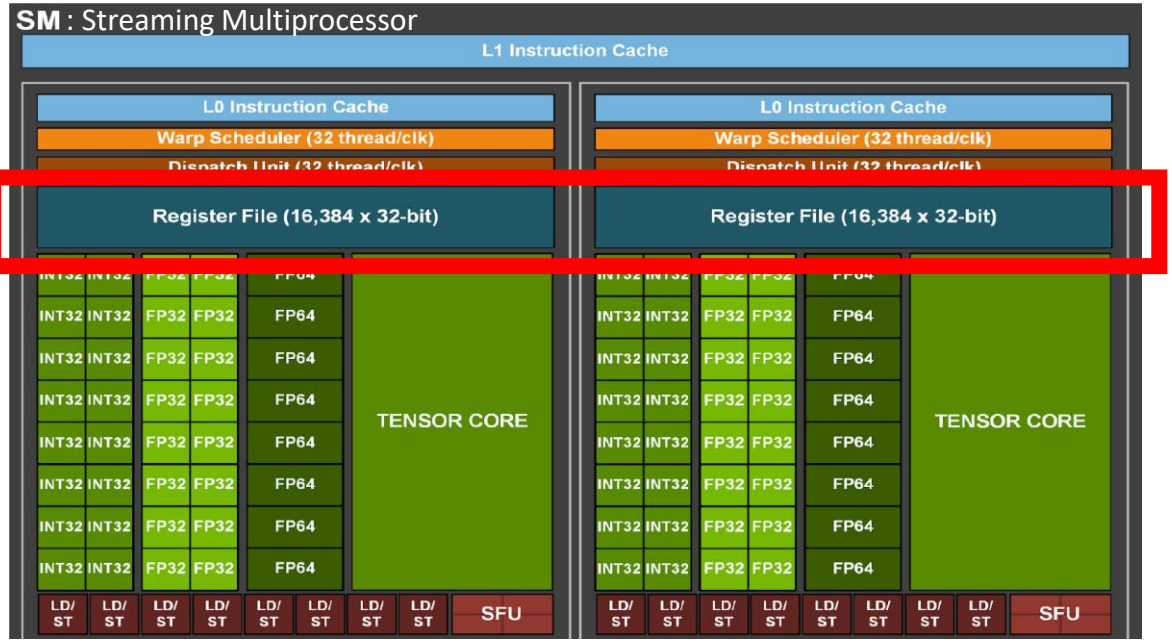
- GA method
  - Chromosome: represents the row shuffle order
  - Mutation/Crossover: different row shuffle orders
  - Fitness Function: top-1 accuracy over the test dataset



The Faults in our Hardware

- Assuming faults occur in SRAM of CUDA cores of a nVidia GPU

NVIDIA A100 Tensor Core GPU Architecture v1.0



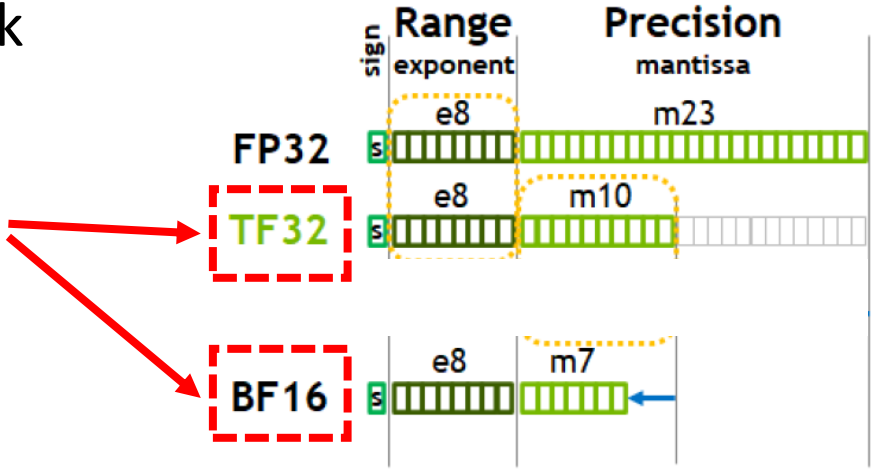
- **Permanent faults**
  - Due to irreversible physical damage
- **Intermittent faults**
  - Occurs for short period of time
  - Due to degradation with age or marginal operation
- **Reduced Precision**:
  - technically not a fault but modeled as one

- Fault profiles can be extracted via diagnostic tests
- Or GA can be run online

Fault Models

We simulate faults using bit-level error injection during inference computation

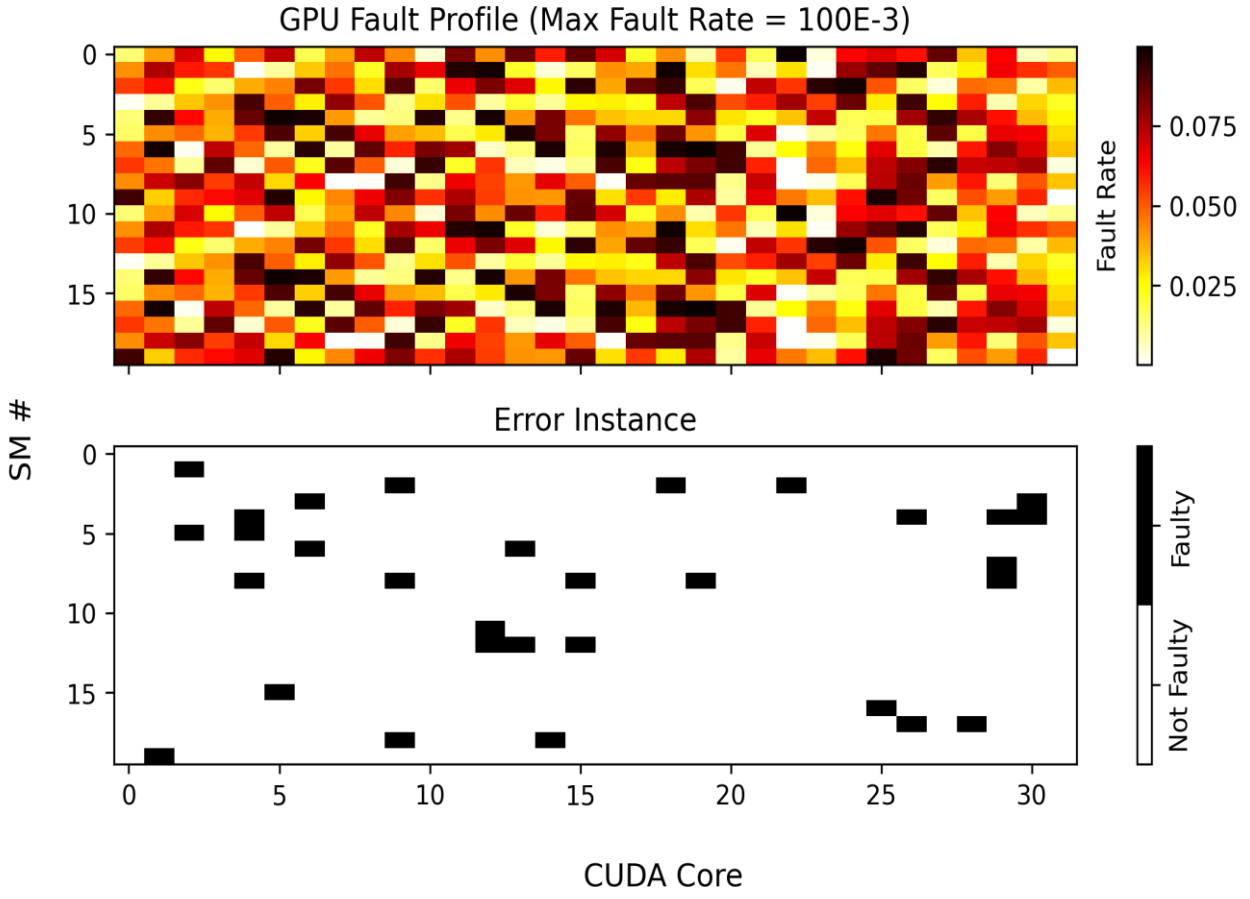
- **Flip-to-0/1:** a random exponent bit is stuck at either 0 or 1
  - Due to permanent damage e.g., shorts/opens
- **Bitflip:** the value of a random exponent bit is flipped.
  - Models intermittent faults due to timing delays, crosstalk
- **Reduced Precision**: some blocks of mantissa bit are zeroed out
  - Saves energy
  - increases throughput



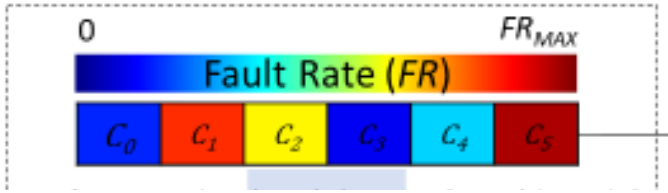
NVIDIA A100 Tensor Core GPU Architecture v1.0

Fault Injection

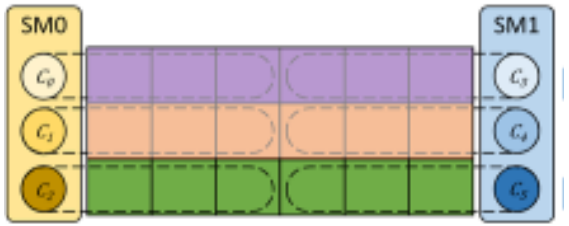
- Each CUDA core has a fixed **fault probability/rate (FR)**.
- A **fault profile** of a GPU determines the fault rate of each of its CUDA cores.
- A fault profile is characterized by the **maximum FR**.
- At each instant in time, a CUDA core is either faulty or not-faulty determined by binomial sampling from the fault rate distribution



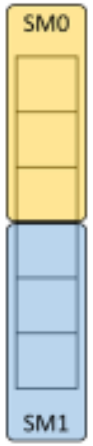
Putting it all together: *HAS* + GA + Fault Injection



The **Fault Profile** defines the fault rate for each PE.

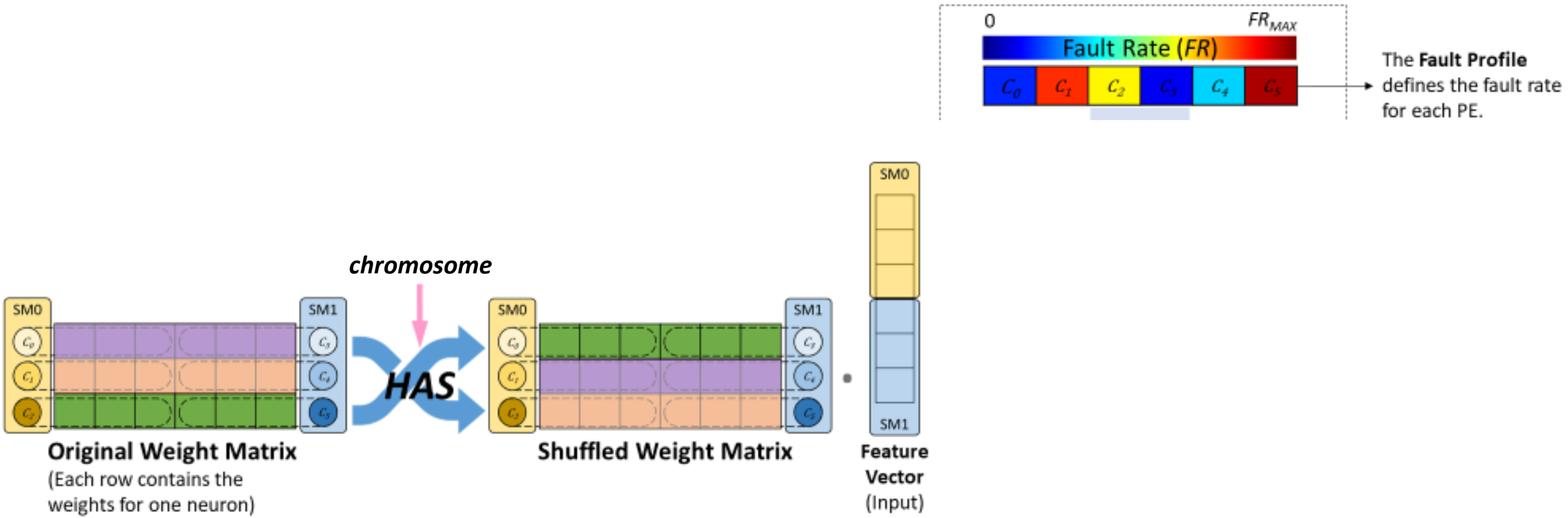


Original Weight Matrix
(Each row contains the weights for one neuron)

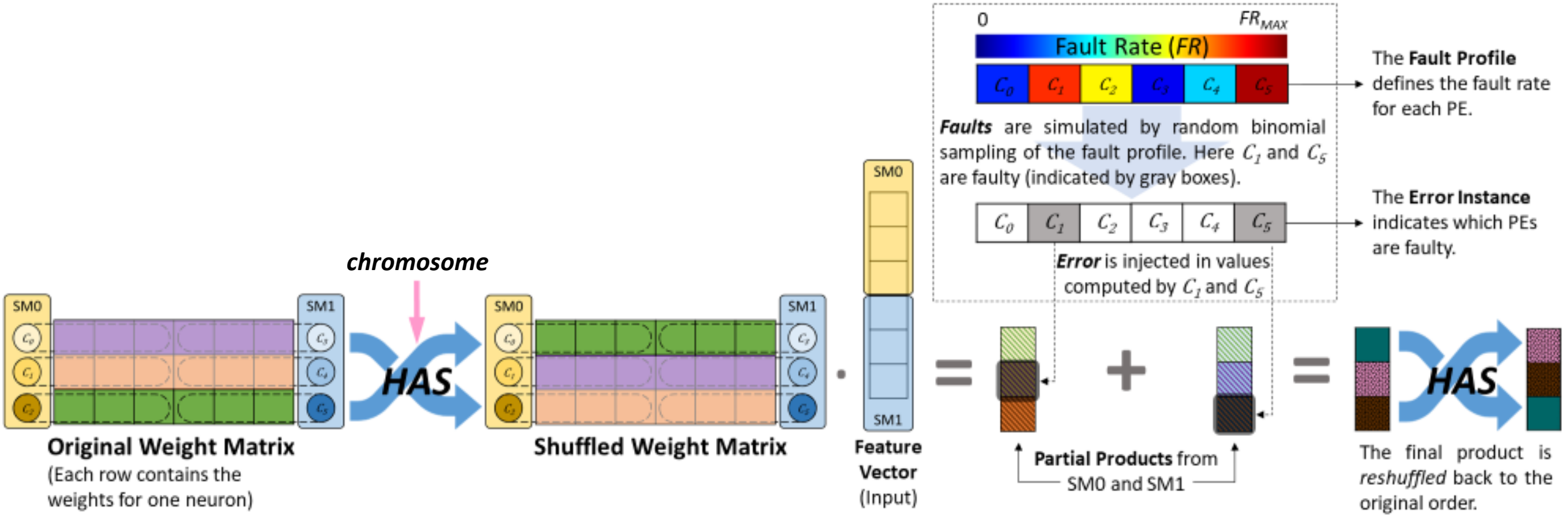


Feature Vector
(Input)

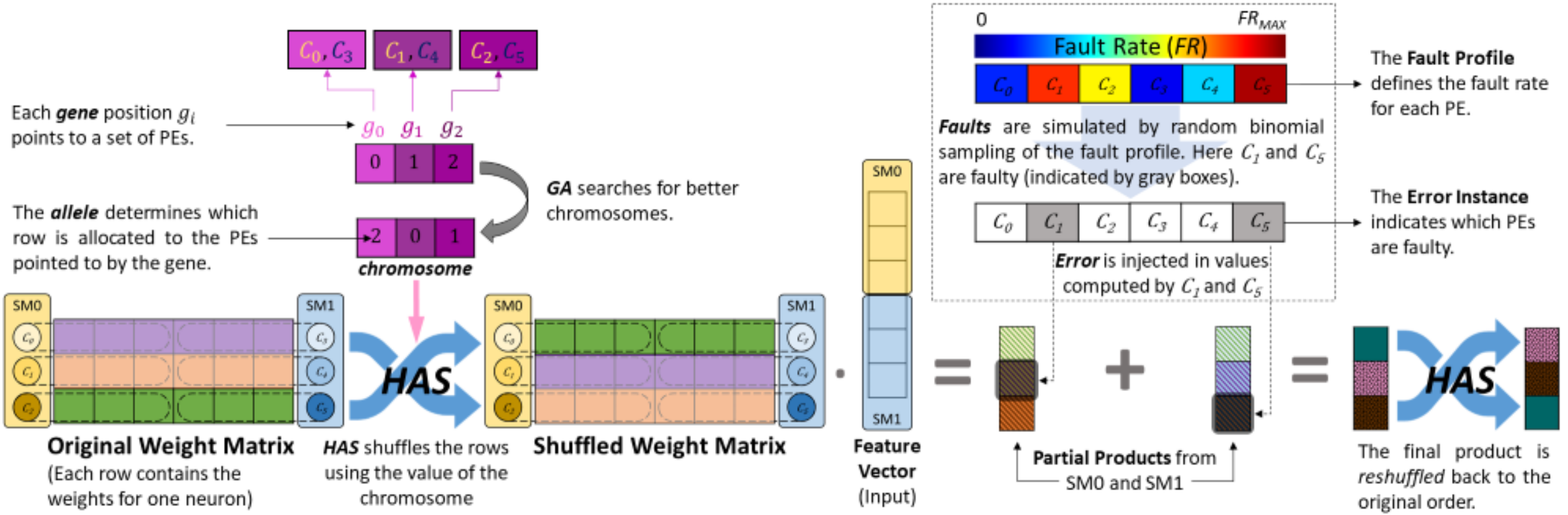
Putting it all together: *HAS* + GA + Fault Injection



Putting it all together: *HAS* + GA + Fault Injection



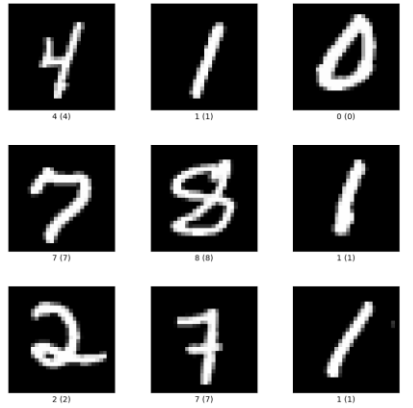
Putting it all together: *HAS* + GA + Fault Injection



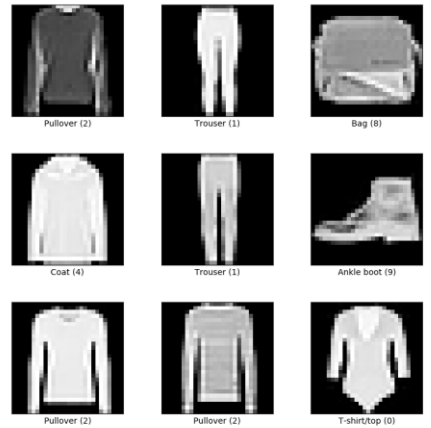
Experiments and Results

Experimental Setup

- DNN Models and Dataset
  - *mnist32-cnn* using MNIST Dataset
    - One conv layer followed by two fully-connected hidden layers
  - *fashion-cnn* using Fashion-MNIST Dataset
    - Two conv layers followed by one fully-connected hidden layer
- Assuming GPU has 20 SMs
- Fault profiles are artificially generated
- Max Fault Rates
  - 1E-3, 2E-3, 5E-3
  - 100E-3, 200E-3, 500E-3



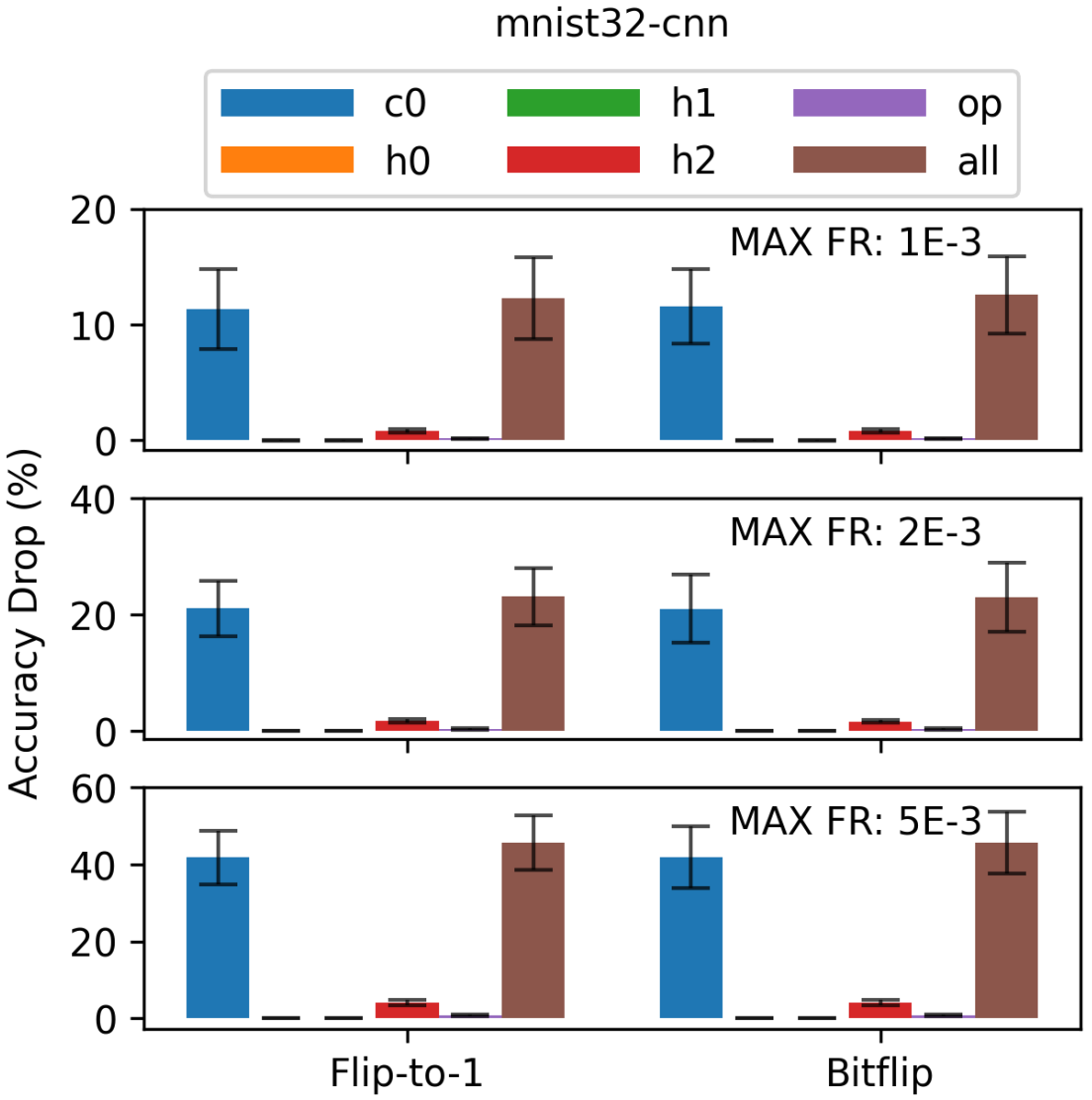
https://www.tensorflow.org/datasets/catalog/mnist



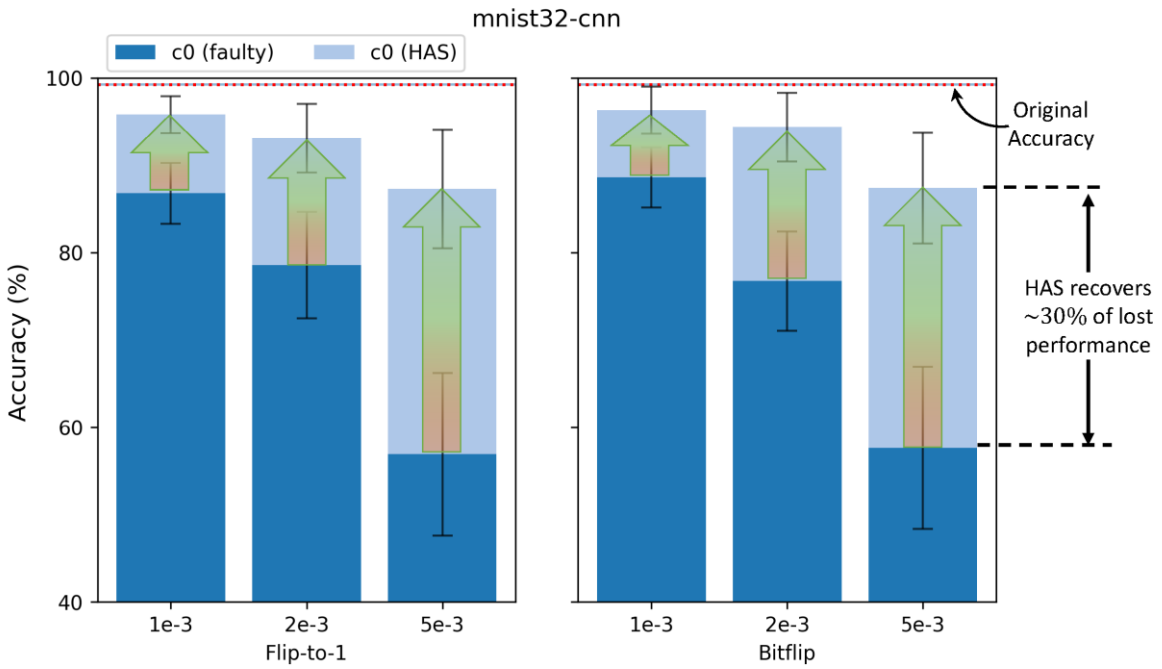
https://www.tensorflow.org/datasets/catalog/fashion_mnist

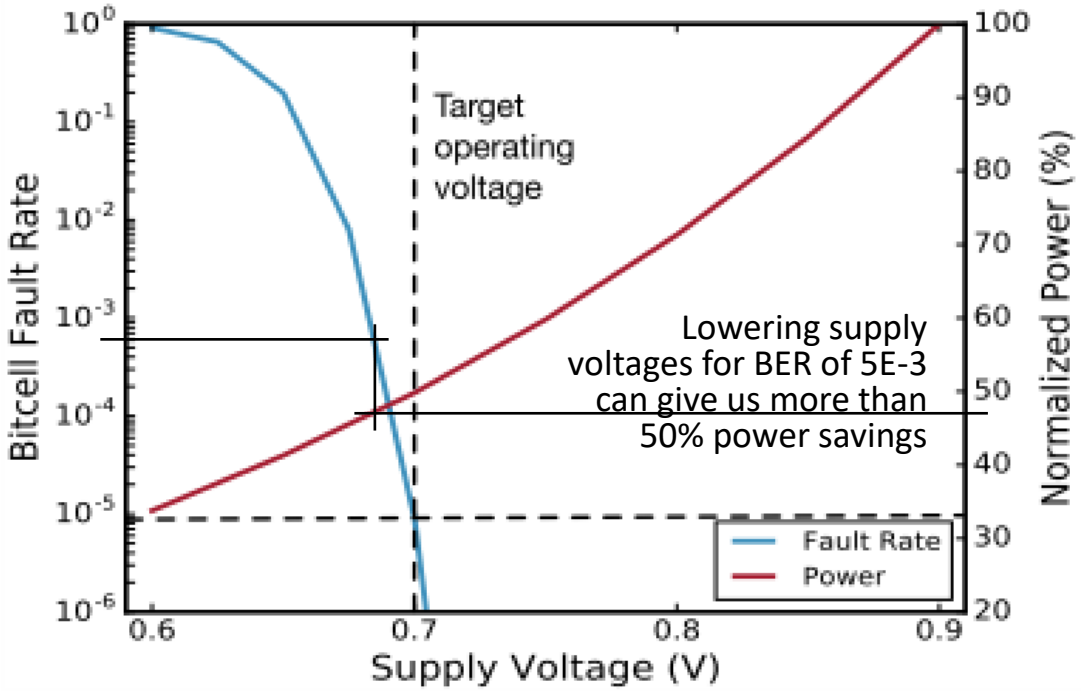
mnist32-cnn: Fault sensitivity (Bitwise Errors)

- Graceful degradation with increasing bit error rate
- conv layer *c0* is most sensitive
  - Due to high reuse
- Can we recover some performance using HAS on *c0*?



mnist32-cnn: Recovery with *HAS* (Bitwise Errors)





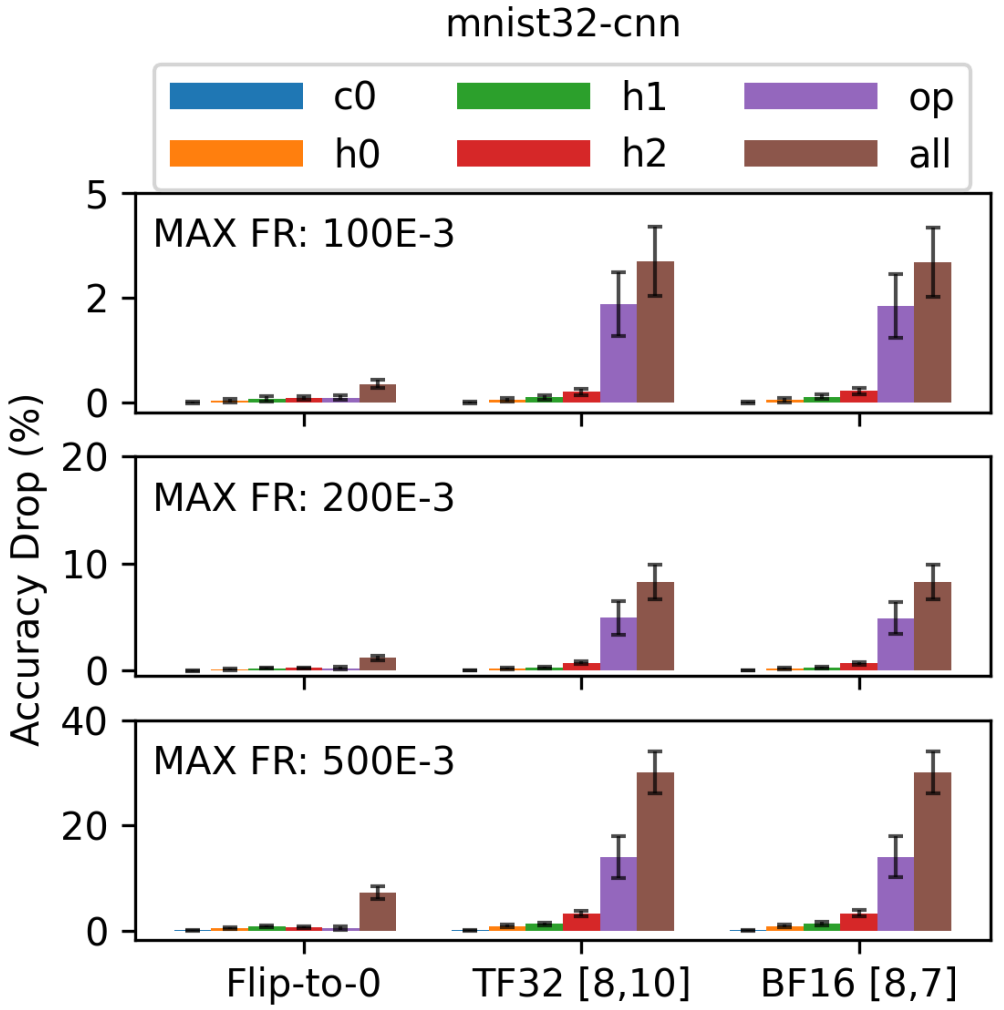
Yes *HAS* can!

- Upto 50% power savings!
- Recovery of up to 30% points!
- With almost no overhead!

Reagen et al. - 2016 - Minerva Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators

mnist32-cnn: Fault sensitivity (Mantissa Errors)

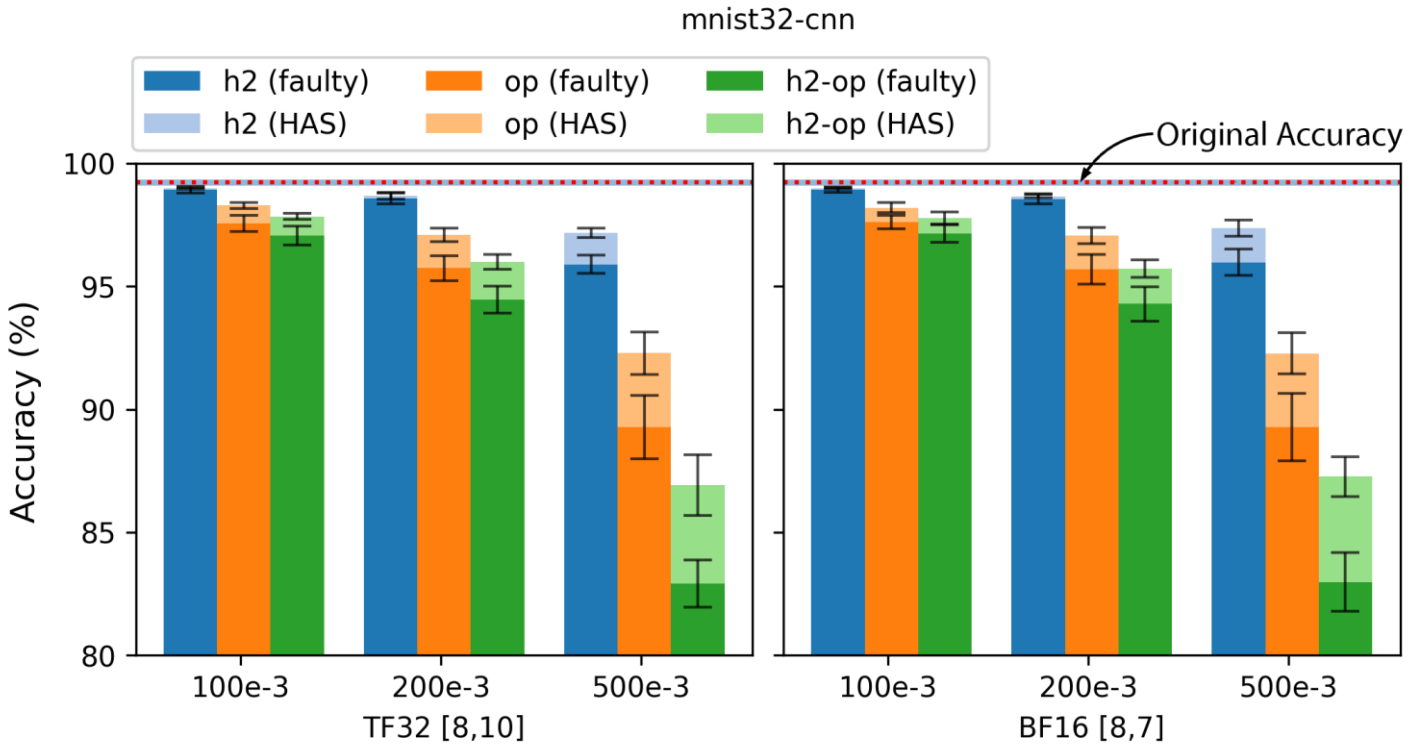
- Flip-to-zero is not that critical
  - Suppressed activations are mostly benign
- Precision can be reduced aggressively!
  - HAQ, RAPiD
- Output layer *op* is most sensitive to reduced precision
  - Softmax function requires high precision
- Degradation of each layer adds up quickly!
- Use *HAS* on *h2* and *op* layers



mnist32-cnn: Recovery with HAS (Mantissa Errors)

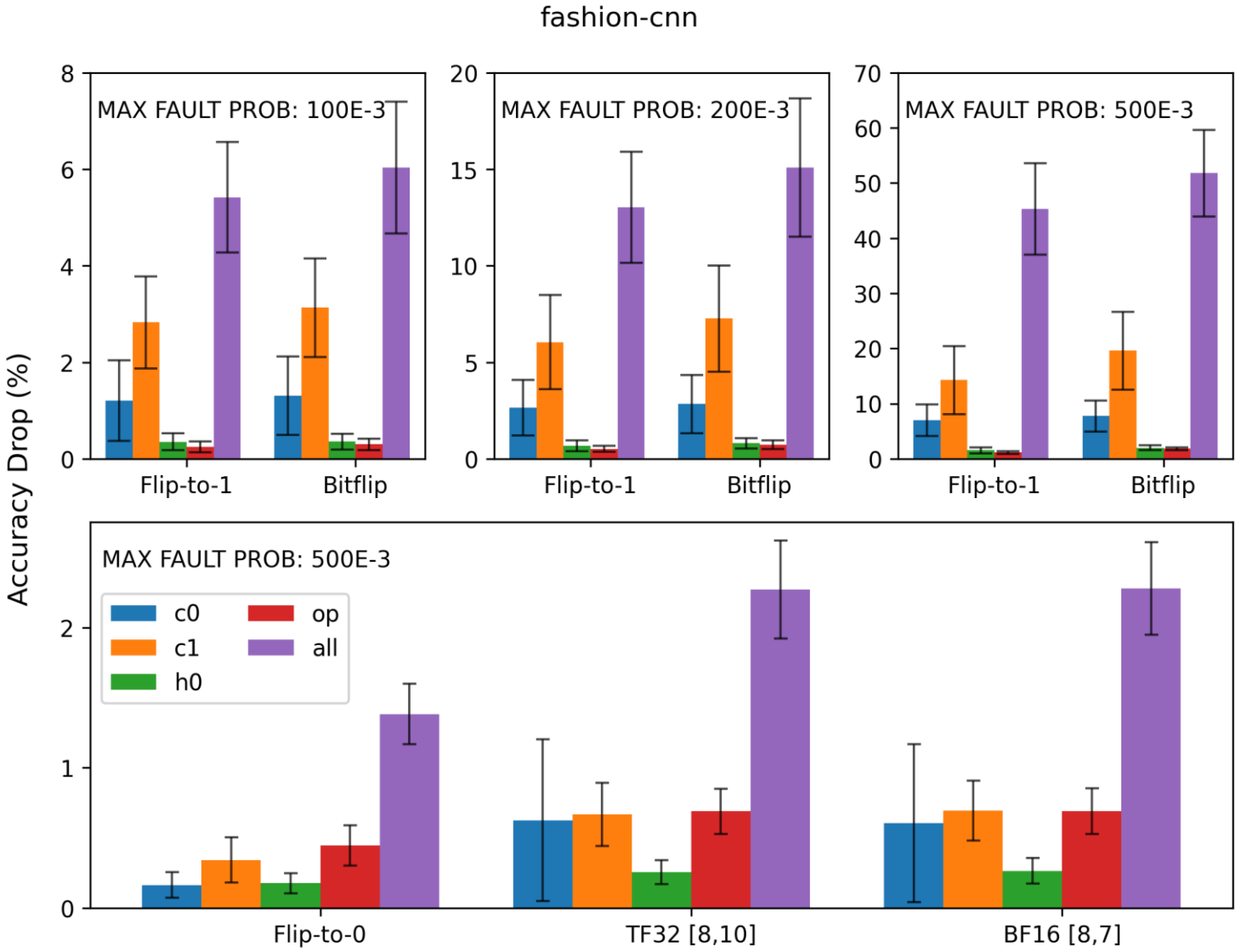
- Recovery of ~5% points
- Greater recovery in op layer

- In our simulation precision is reduced randomly
  - Harder optimization problem
- In reality
  - More deterministic and controlled mixed precision schemes



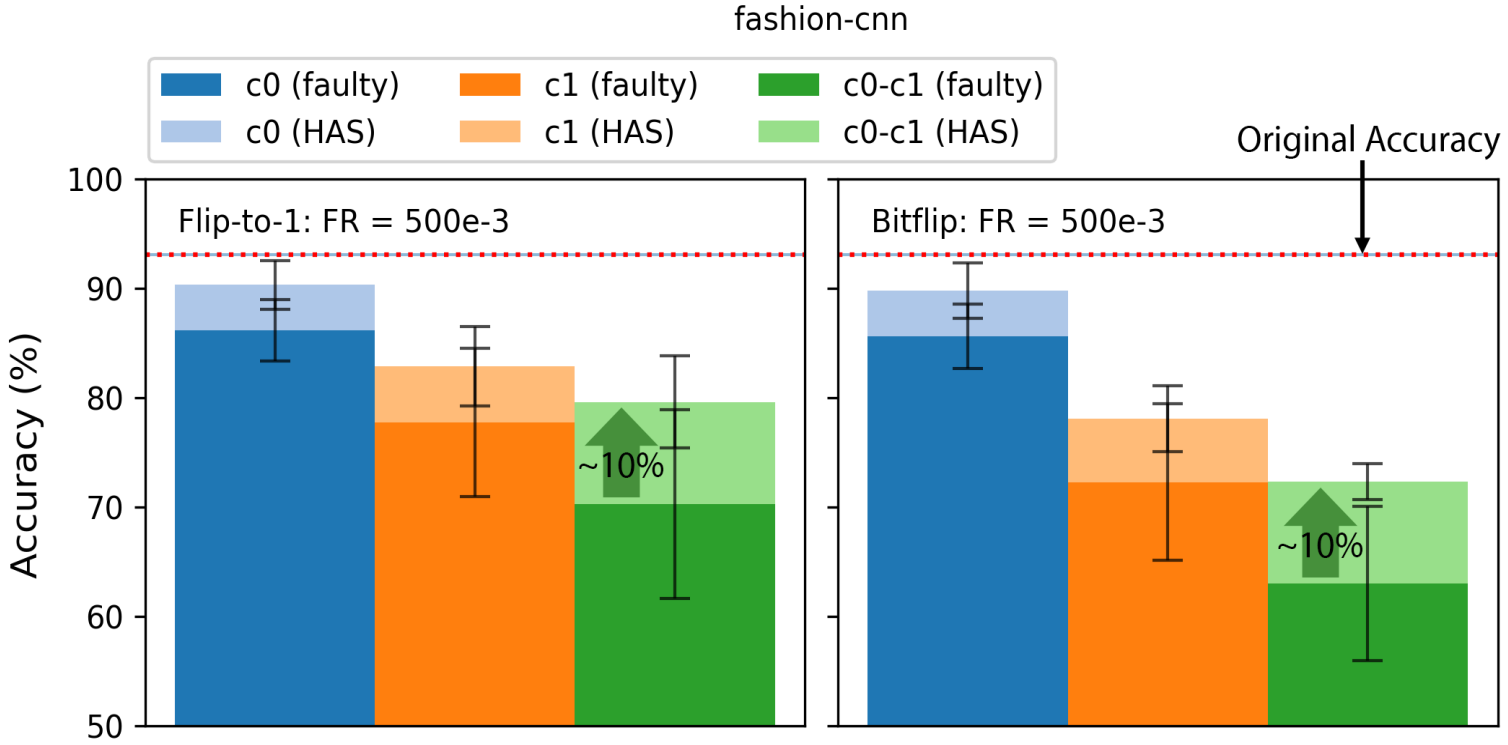
fashion-cnn: Fault sensitivity

- Trained with aggressive dropout so the model is more robust
- Conv layers *c0,c1* are most sensitive to flip-to-1/bitflip
- Model is almost unaffected by reduced precision
- Use HAS for *c0,c1*



fashion-cnn: Recovery with HAS

- Recovery of up to 10% points!
- Very large power savings ~60%



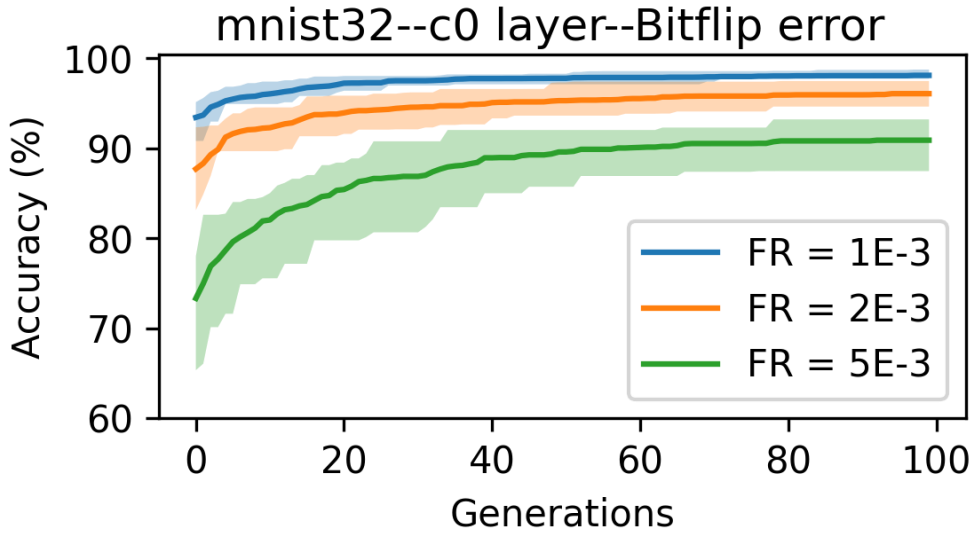
Conclusion

- Accelerators are faulty
- DNN can tolerate some computation inexactness
- We propose *HAS* so that unimportant computations are scheduled to compromised hardware
  - Achieved by shuffling rows
  - Best shuffling order is found using GA
- *HAS* can recover up to 30% points of performance
  - Corresponds to large power savings

- HAS is hardware-agnostic black-box optimization process – general, simple and cheap to implement

Future Work

- GA hyperparameters can be tuned for faster convergence
- Can we use faulty hardware for training?
- Integrate more sophisticated fault models



REFERENCES

- Reagen, Brandon, et al. "**Minerva**: Enabling low-power, highly-accurate deep neural network accelerators." *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016.
- Reagen, Brandon, et al. "**Ares**: A framework for quantifying the resilience of deep neural networks." *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018.

- Venkataramani, Swagath, et al. "**AxNN**: Energy-efficient neuromorphic systems using approximate computing." *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2014.
- He, Xin, et al. "**AxTrain**: Hardware-oriented neural network training for approximate inference." *Proceedings of the International Symposium on Low Power Electronics and Design*. 2018.
- Zhang, Qian, et al. "**ApproxANN**: An approximate computing framework for artificial neural network." *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015.

- Wang, Kuan, et al. "**Haq**: Hardware-aware automated quantization with mixed precision." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.
- Venkataramani, Swagath, et al. "**RaPiD**: AI accelerator for ultra-low precision training and inference." *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021.

Thank you

https://www.acsl.ics.keio.ac.jp/

https://shaswot.com/