Slaying Old Dragons: Error-Resilient Machine Learning for Safety-Critical Applications

Karthik Pattabiraman, Guanpeng (Justin) Li, Zitao Chen

Siva Hari, Michael Sullivan, Tim Tsai, Joel Emer, Steve Keckler

Nathan DeBardeleben

Machine Learning

- Machine Learning (ML) Systems work around humans

Home Care

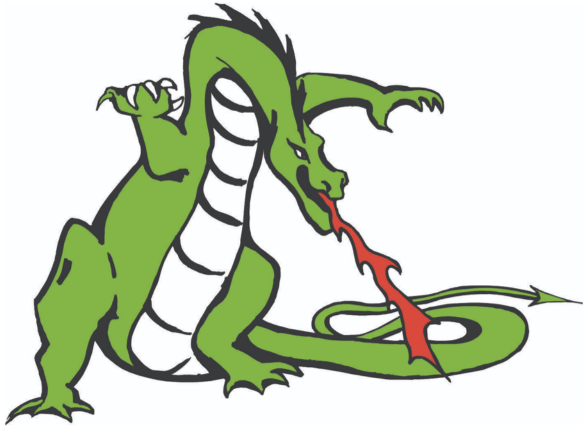
Policing

Self-Driving Cars

Machine-learning is increasingly used in safety-critical systems

Dragons of Machine Learning (ML)

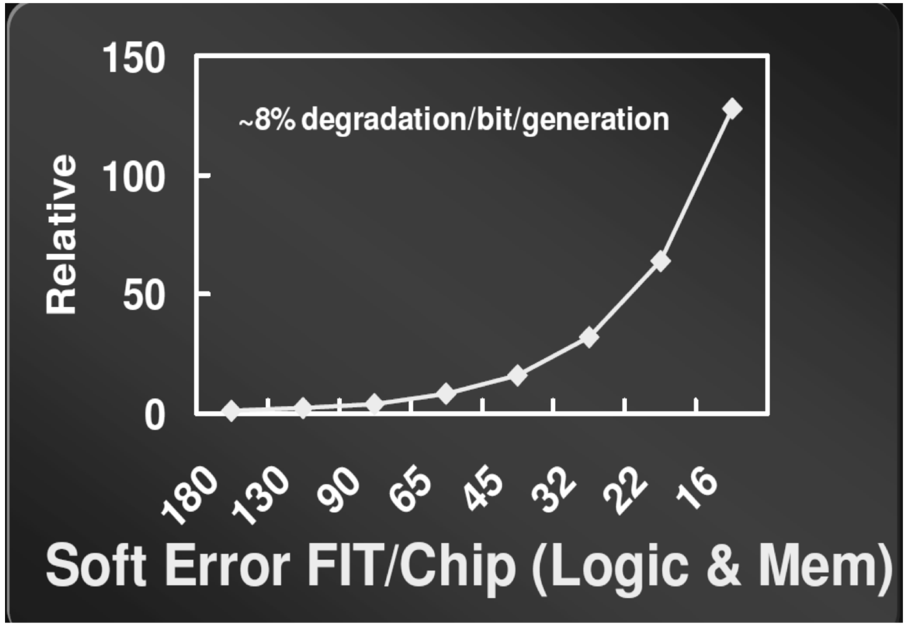
- **New dragons**
  - Adversarial Inputs (Security)
  - Edge cases (Reliability)
  - Confidentiality/Privacy



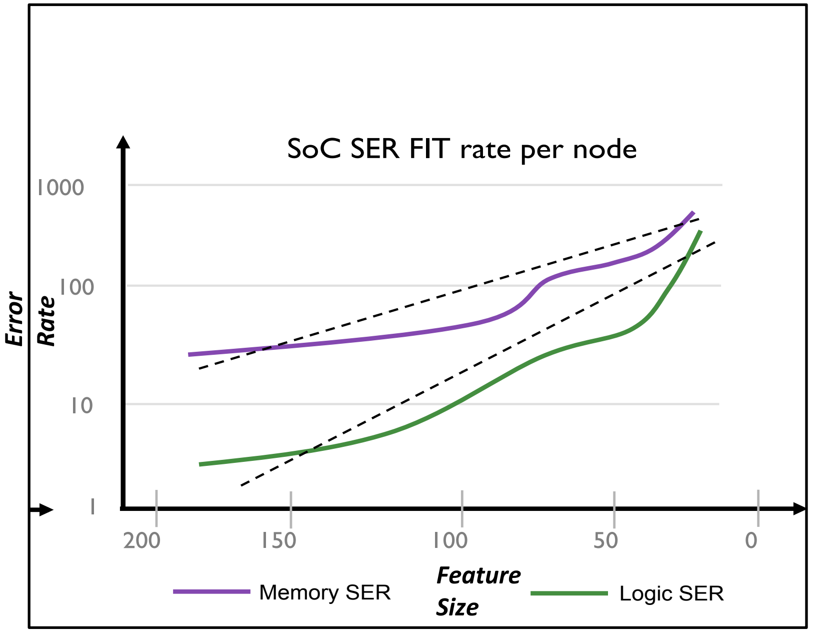
- **Old dragons**
  - Soft errors
  - Permanent faults
  - Logical errors
  - Implementation faults (bugs)

Soft Error Problem

Soft errors are increasing in computer systems

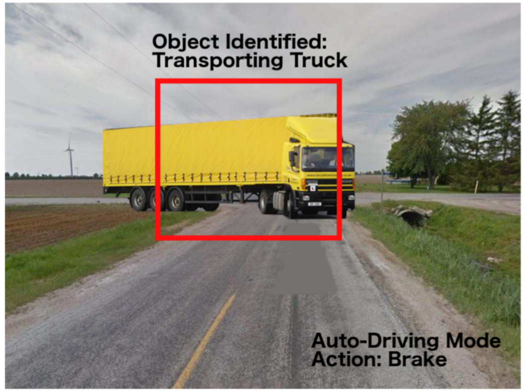


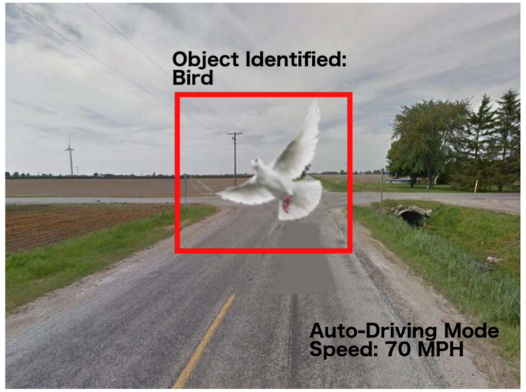
Source: Shekar Borkar (Intel) - 2005



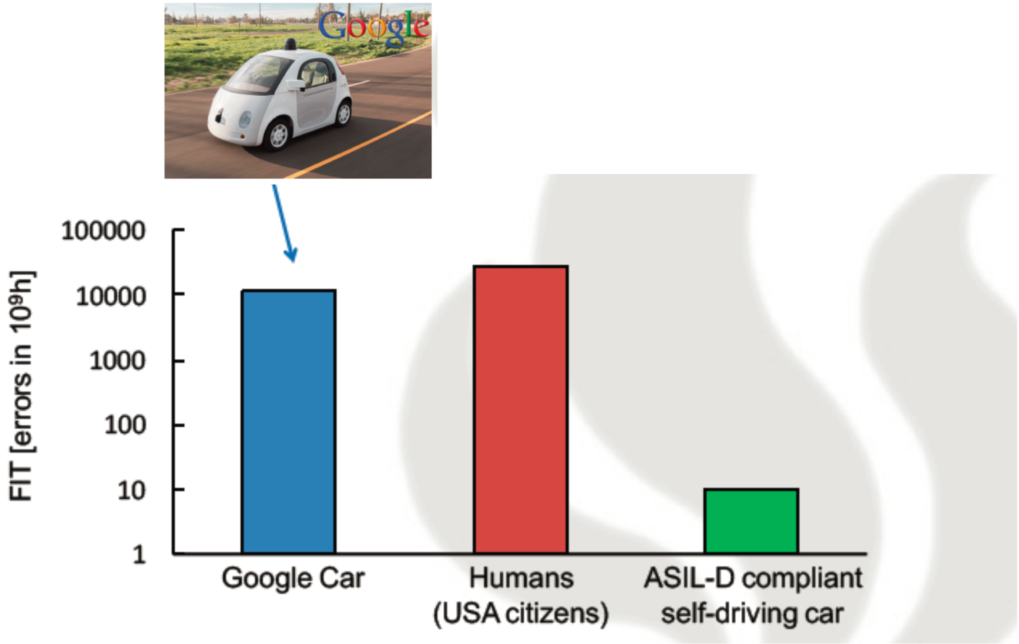
[Chandra et. al., DATE'14]

Why Soft Errors?





[Our work - SC 2017]



[Saxena'16]

- Safety standard – Automotive Safety Integrity Level (ASIL-D)
  - Error rate <10 FIT (per 1 billion hours) – ISO 26262
  - DNN systems do not meet the requirement without protection

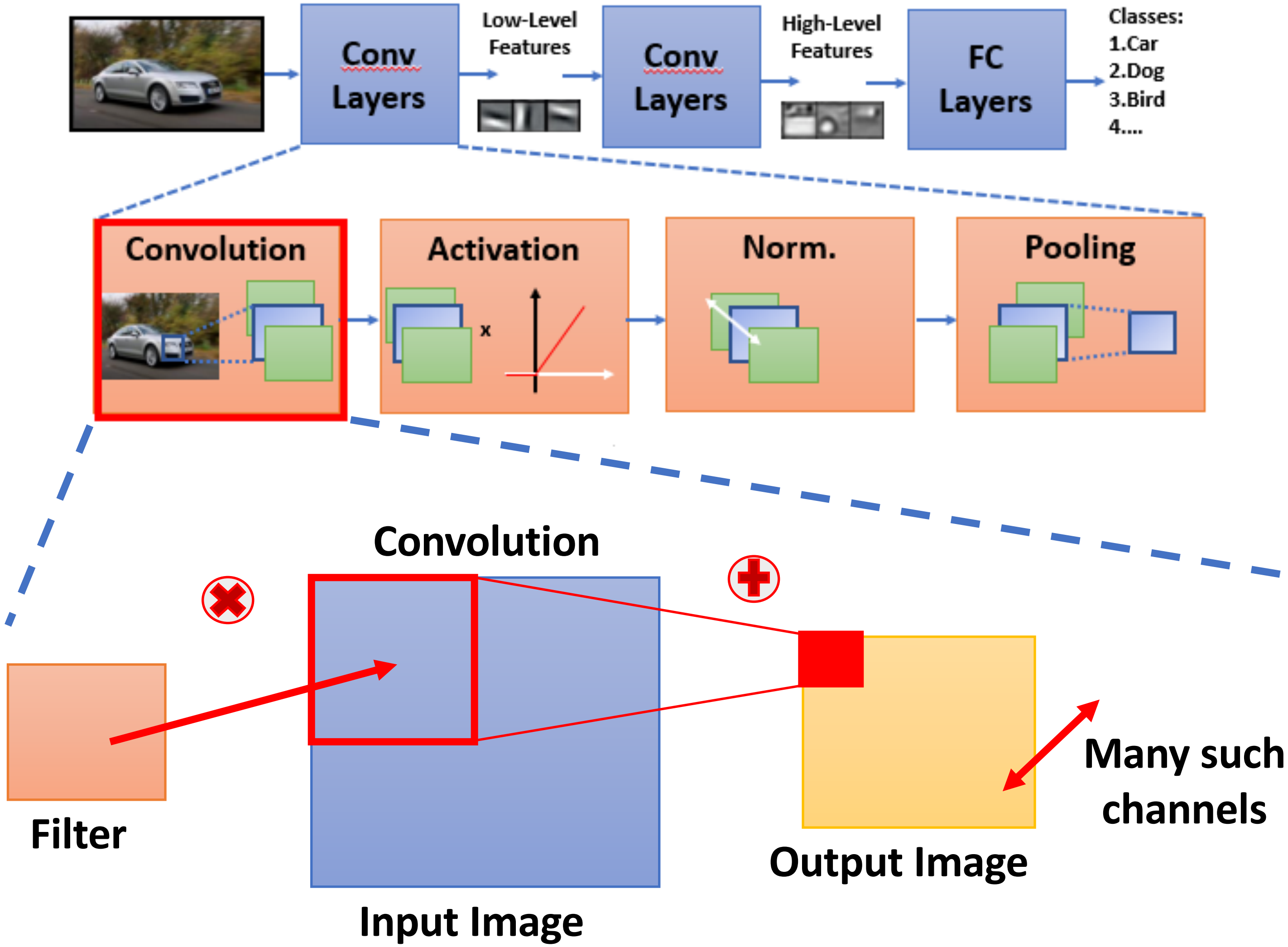
Traditional Solutions

- **Full redundancy**
  - **Expensive for cost-sensitive domains such as automotive sector**
    - Profit margin of mid-class sedan: 8-10%
    - Efficiency regarding per-unit-price
  - **High overheads in performance and energy**
    - Significant reduction of processing frame rate which is critical in high-speed self-driving
    - Significant energy and cooling costs

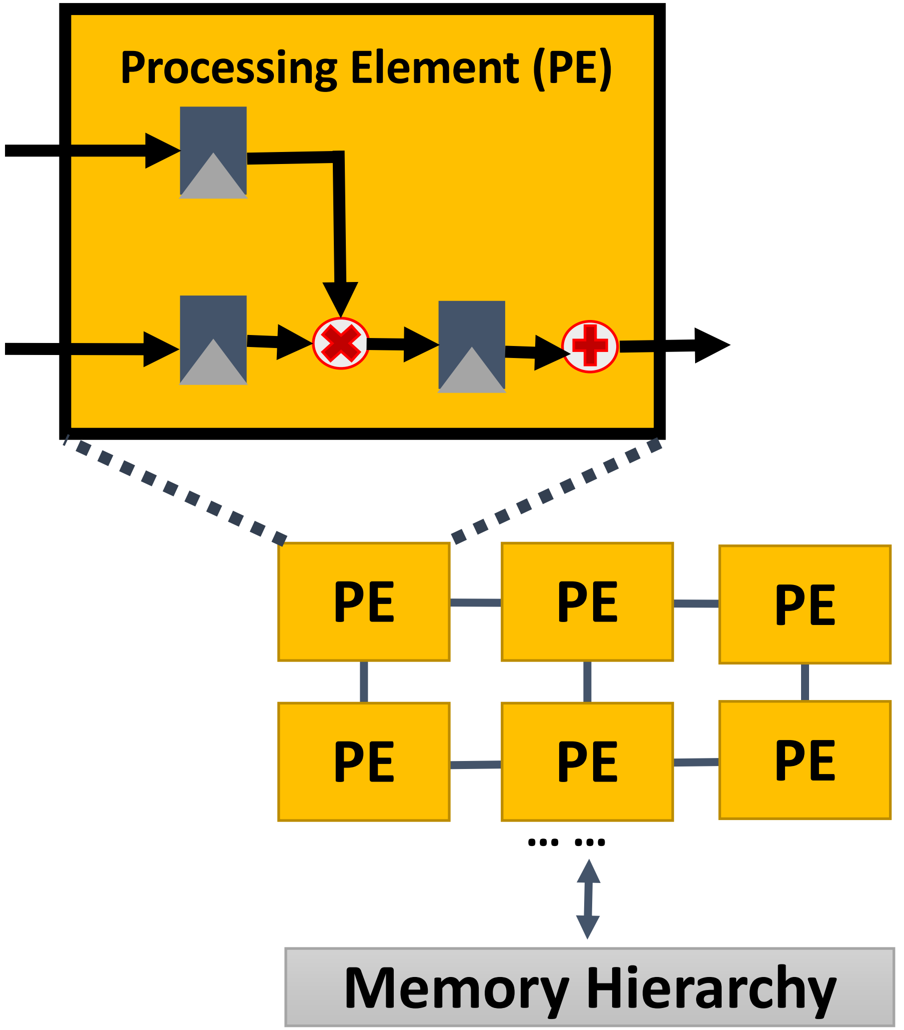
Outline

- Motivation and Goals
- Fault-Injection into Deep Neural Networks [SC’17]
- BinFI: Efficient Fault Injector for ML systems [SC’19 - to appear]
- Ongoing Work and Conclusions

DNN and Accelerators

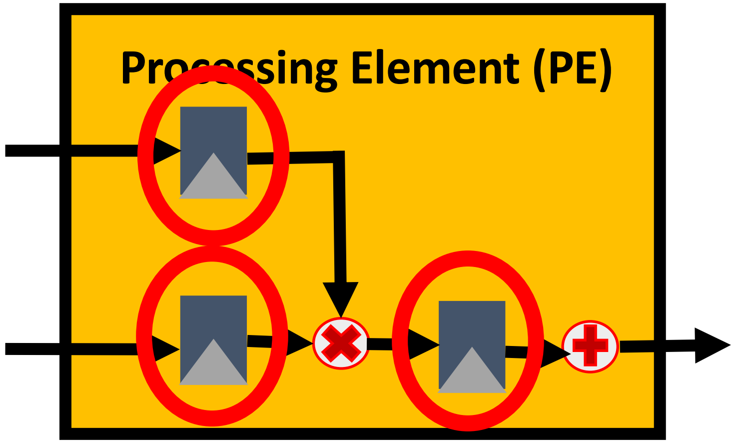


Spatial Architecture



Fault Model and Fault Injection

- Inject one random single bit-flip fault per one inferencing (input)
- 3,000 trials per each latch (less than 1% error bars)
- Silent Data Corruption (SDC): Mismatch with the winner of fault-free execution



Experimental Setup

- **Goal:**
  - Investigate error sensitivity of different neural networks , data types, bit positions, positions and types of layers, as well as values
  - Design cost-effective mitigation techniques
- **Neural Networks:**
  - AlexNet, CaffeNet, NiN, ConvNet
- **Data Types:**
  - 4 Integers and 3 Floating Points

SDC Types

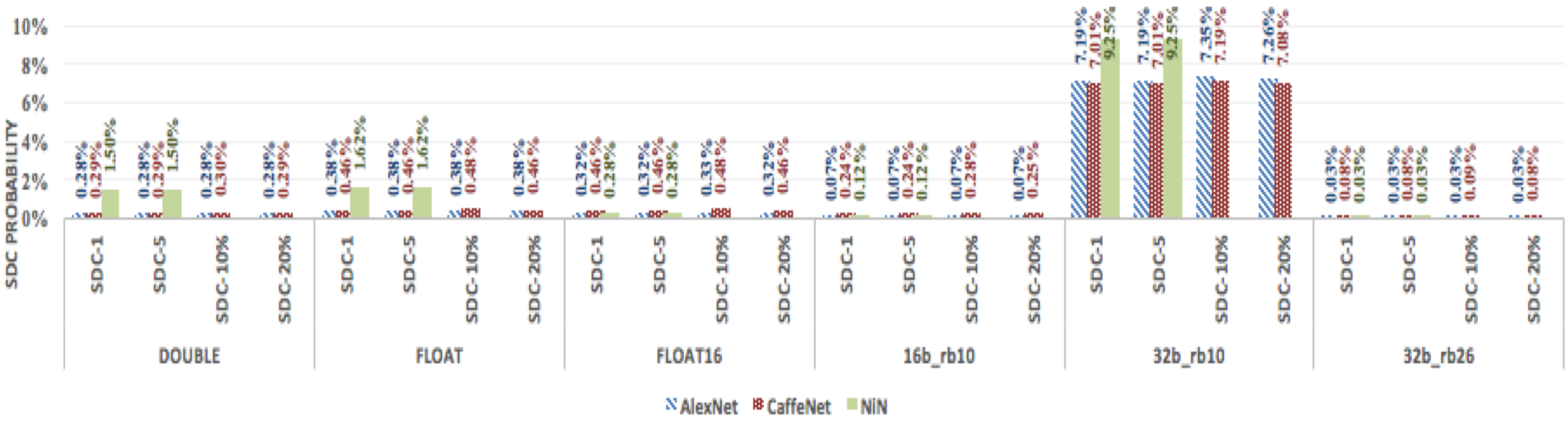
SDC1:
Mismatch between winners from faulty and fault-free execution.

SDC5:
Winner is not in top 5 predictions in the faulty execution.

SDC10%:
The confidence of the winner drops more than 10%.

SDC20%:
The confidence of the winner drops more than 20%.

RQ1: SDC in DNNs



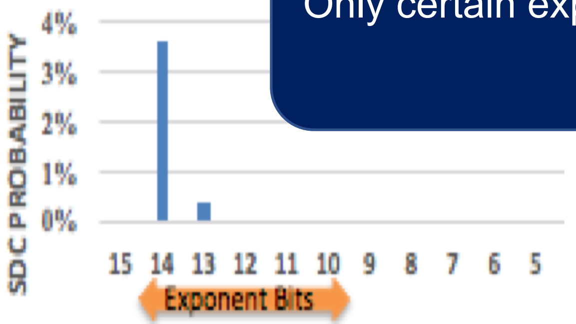
1.All SDCs defined have similar SDC probabilities

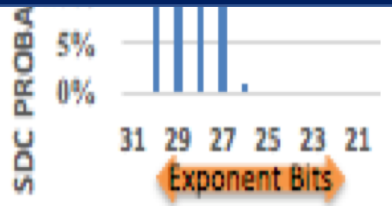
2.SDC probabilities are different in different DNNs

3.SDC probabilities vary a lot using different data types

RQ2: Bit Sensitivity

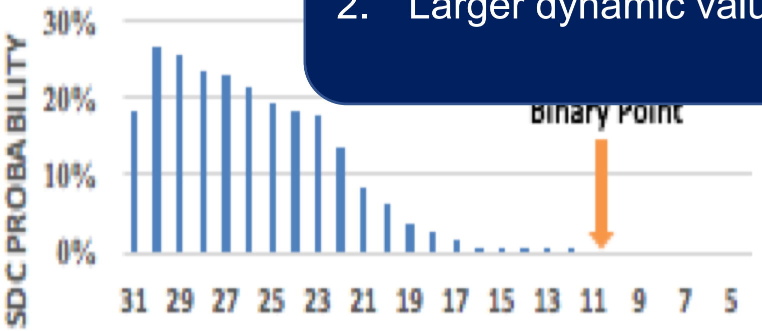
FP data types:

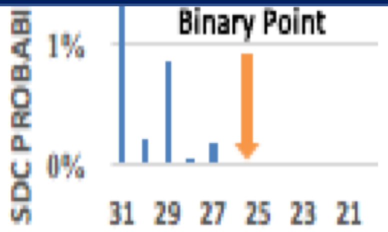




Only certain exponent bits are vulnerable to SDCs

FxP data types:

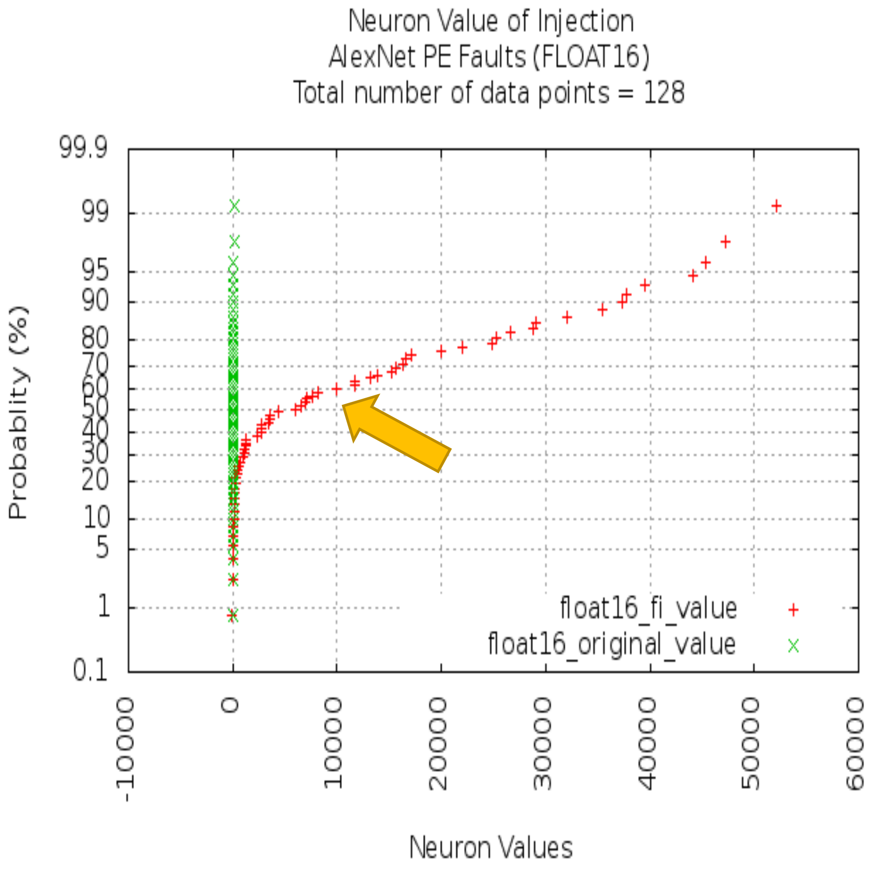




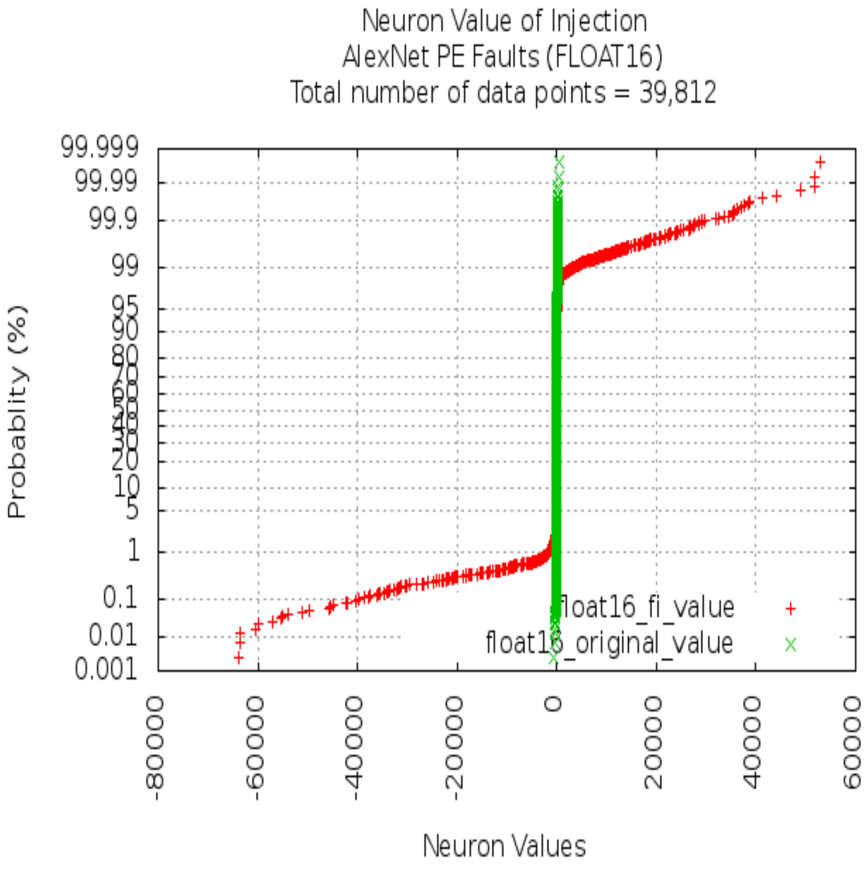
1. High-order bits are vulnerable
2. Larger dynamic value range allows more vulnerable bits

RQ3: Value Changes

AlexNet, PE Errors, Float16



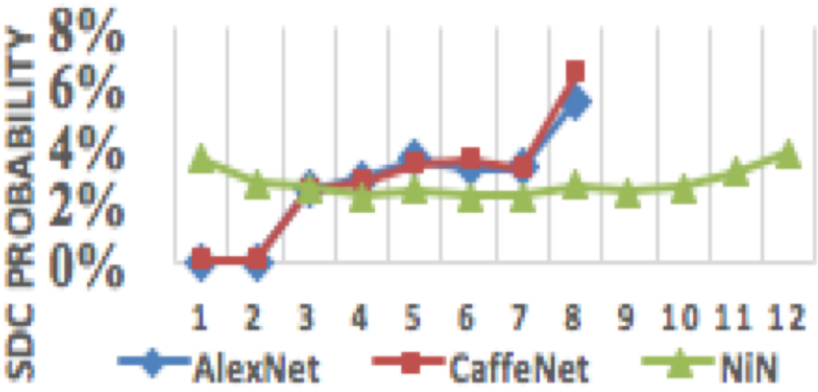
SDC

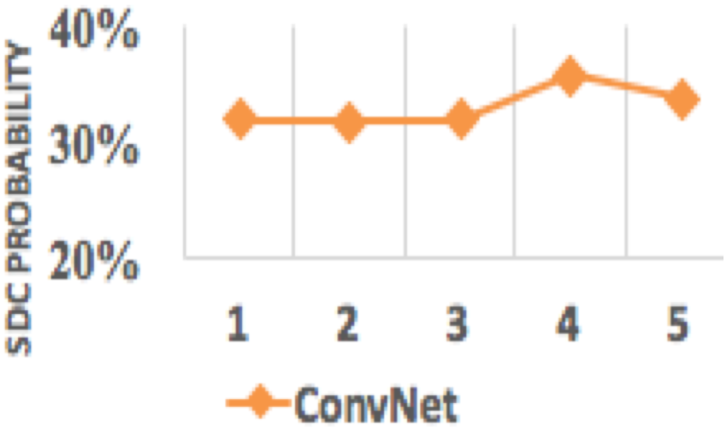


Benign

If a neuron value is changed to be a large value under a fault, it likely causes SDC

RQ4: SDC in Different Layers

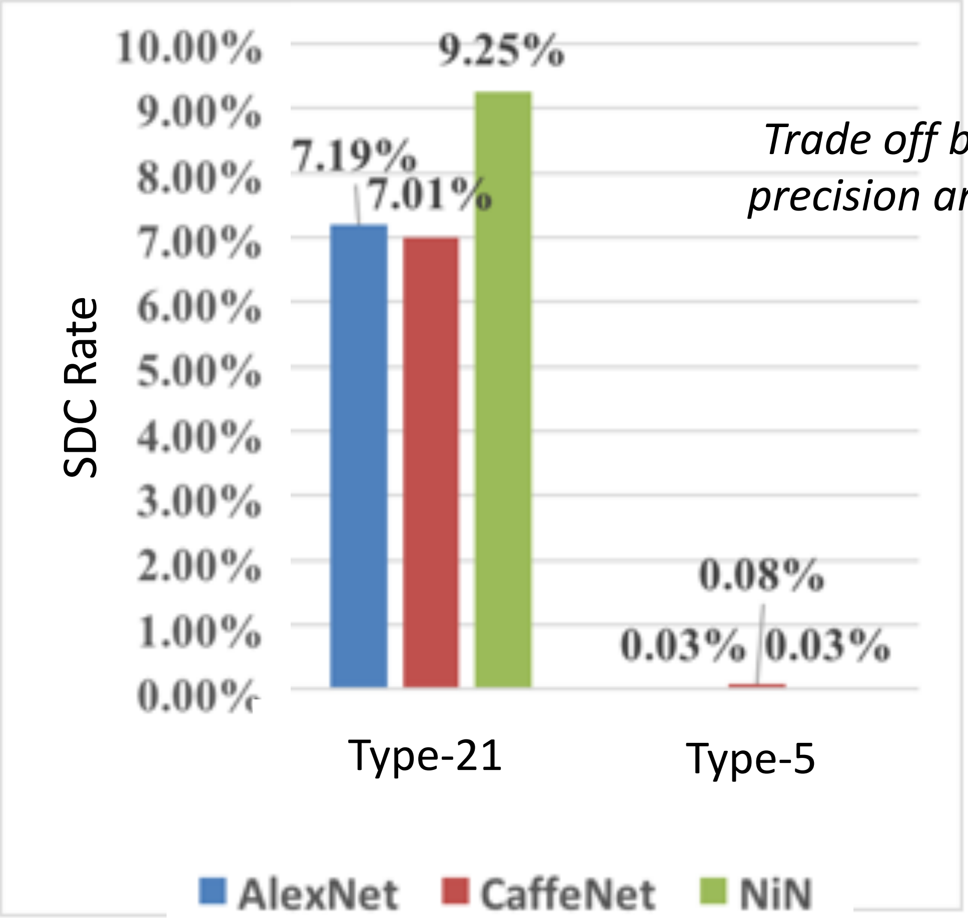




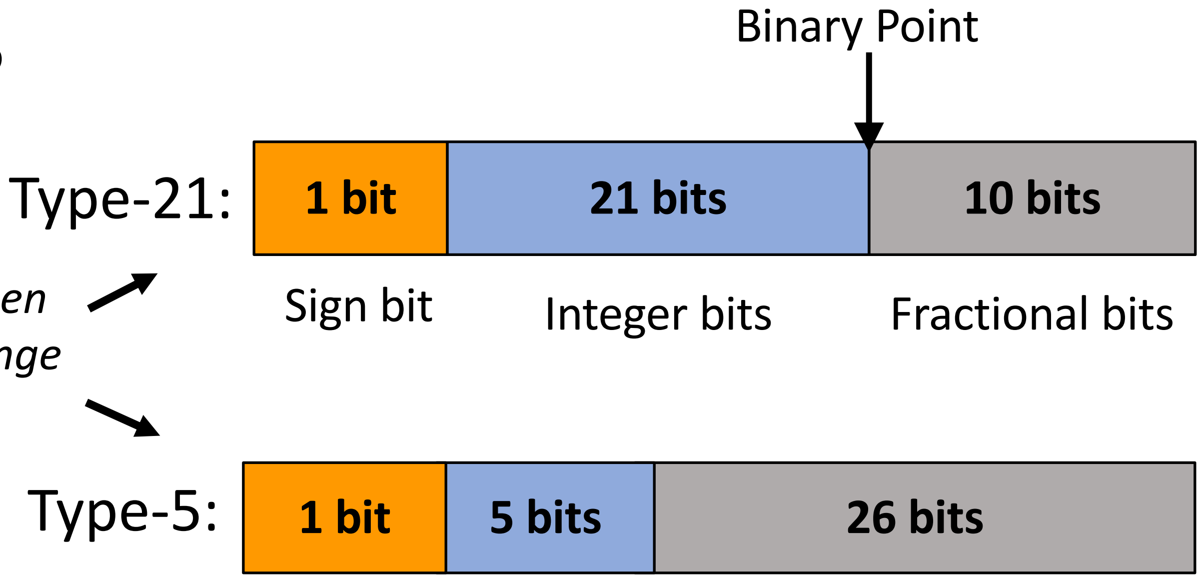
1.Layers 1&2 have lower SDC probabilities in AlexNet and CaffeNet

2.SDC probability increases as layer numbers increase

Mitigation: Data Types



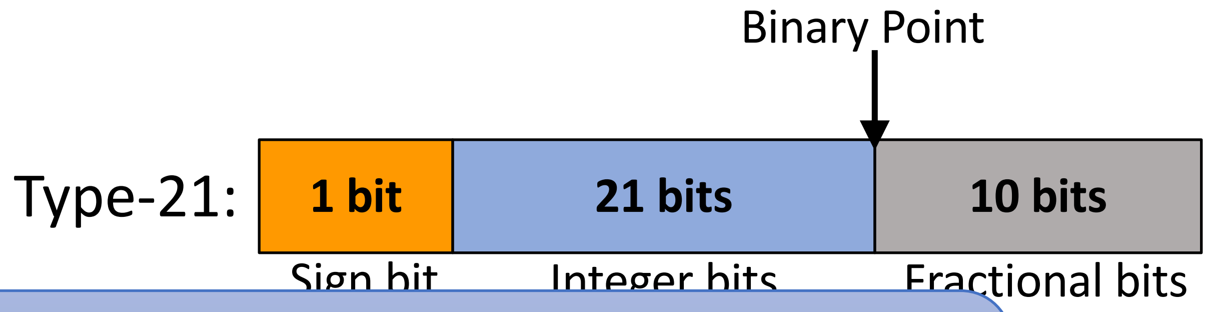
Trade off between precision and range



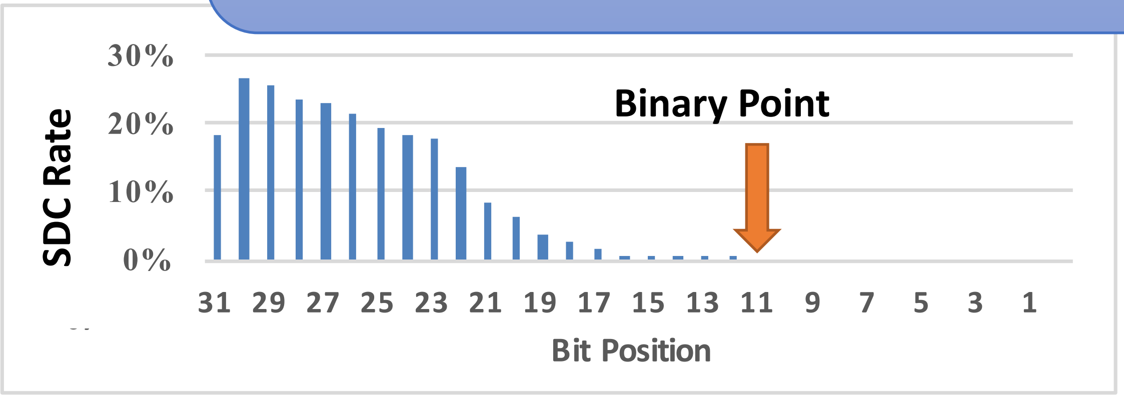
32-Bit Fixed-Point data types as examples

Restraining dynamic value range suppresses SDCs

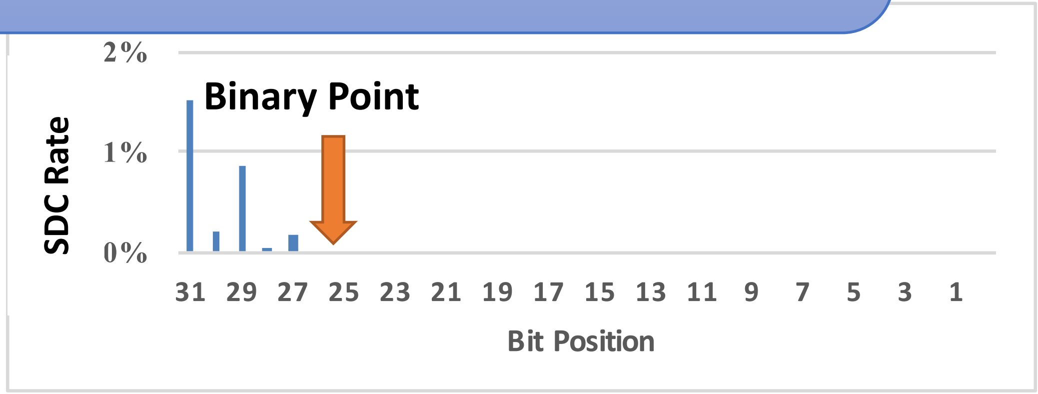
Mitigation: Hardware



1. Only high-order bits are vulnerable
2. Sliding binary points controls the amount of vulnerable bits



Type-21



Type-5

Selective latch hardening:
Can reduce FIT rate by 100X with 20-25% area overhead

Takeaways

- **DNNs are not as resilient as one may think**
  - Single bit-flips can lead to safety-critical outcomes
  - Accelerator platforms exacerbate the situation

- **Key findings from fault injection study**
  - Restricted range improves resilience
  - Higher-order bits are more sensitive
  - Errors that occur in later layers are more impactful

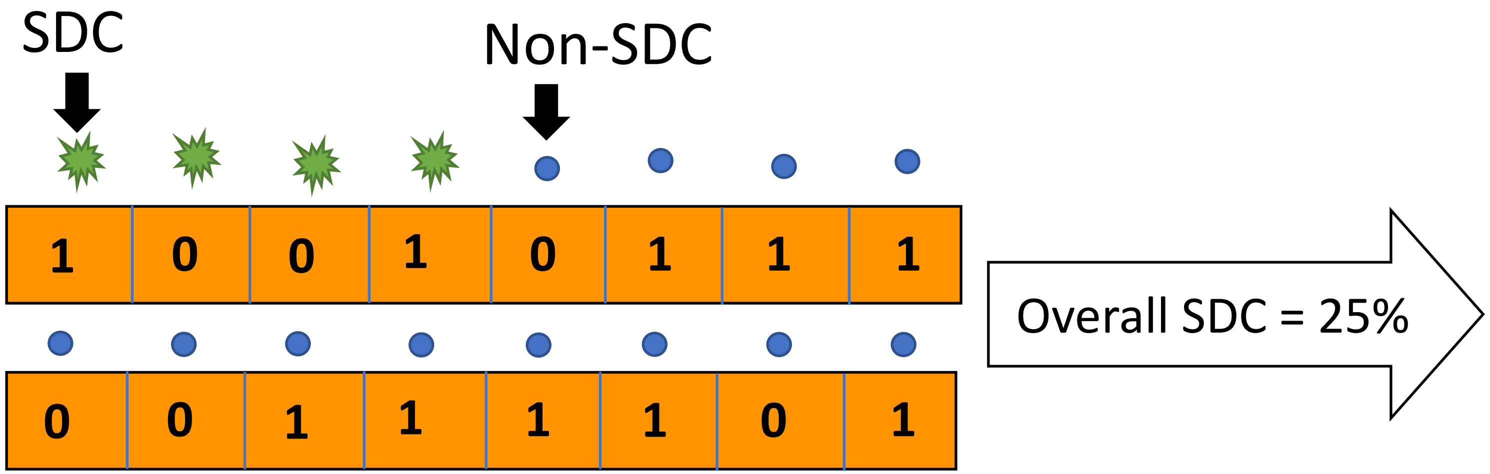
https://github.com/DependableSystemsLab/DNNFI

Outline

- Motivation and Goals
- Fault-Injection into Deep Neural Networks [SC'17]
- BinFI: Efficient Fault Injector for ML systems [SC'19 - to appear]
- Ongoing Work and Conclusions

Motivation

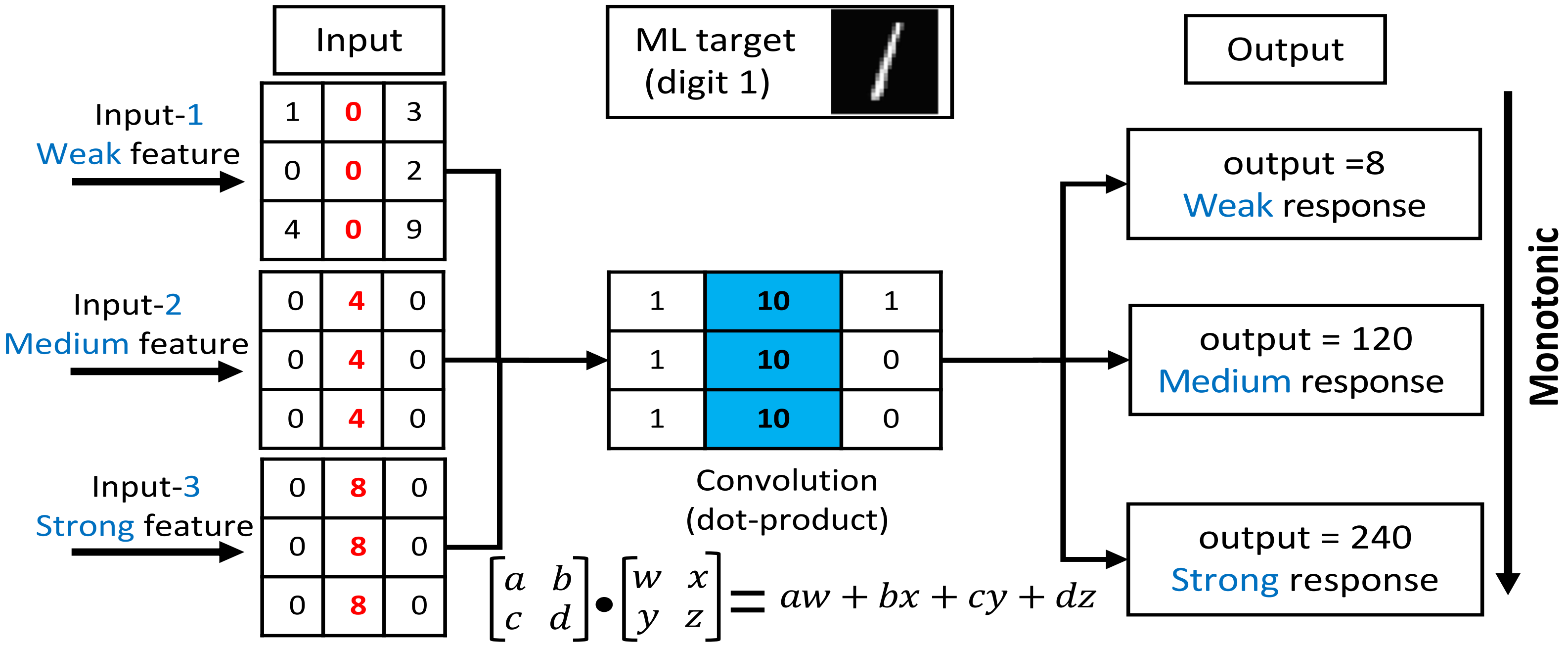
- Existing approaches – fault injection (FI)
  - Exhaustive FI: Ground truth, high overhead (impractical)
  - Random FI: Statistically significant results, low overhead (not good enough)



Which part is more prone to SDCs

Key Insight

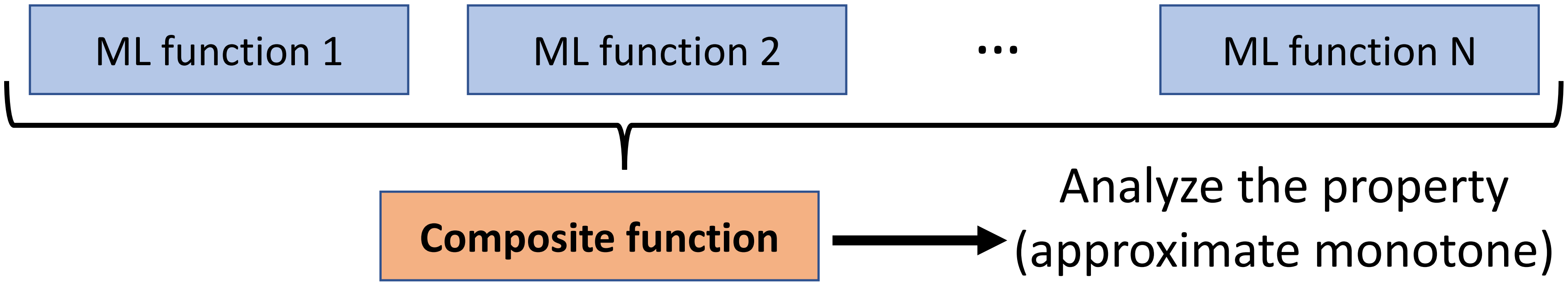
- In ML, a fault *only* results in numerical changes in the output
- Output by ML is usually determined by numerical magnitude



- **Larger deviation** in the **Output** → **higher** probability of SDCs

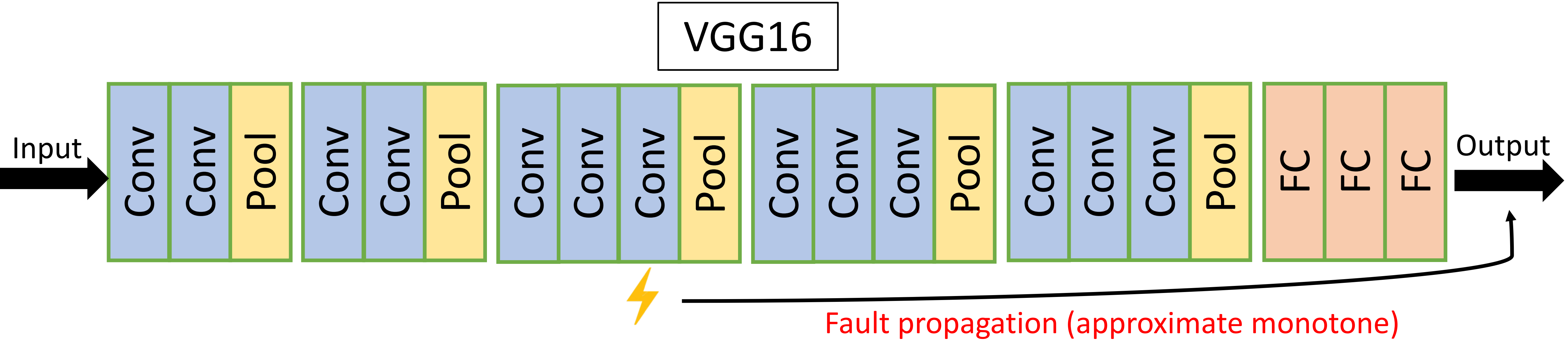
ML computations and Monotonicity property

- *Approximate monotonicity*: A function being non-strictly monotonic in a non-trivial interval
  - E.g., $f(x) = 100 * max(x-1, 0) - max(x, 0)$, is *approximate monotonic* when $x > 0$
- **Approach**: Analyze the property of the ML functions, which propagate the fault from fault site to the output



Our Insight

- **Approximate** the fault propagation behavior as an *approximate monotonic* function.

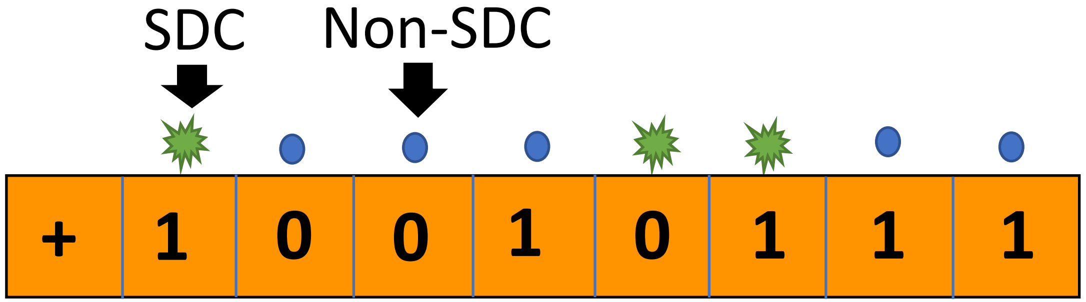


- Implication: *Larger input deviation (fault at higher-order bit) generates larger deviation at the output, thus more likely to cause SDCs*

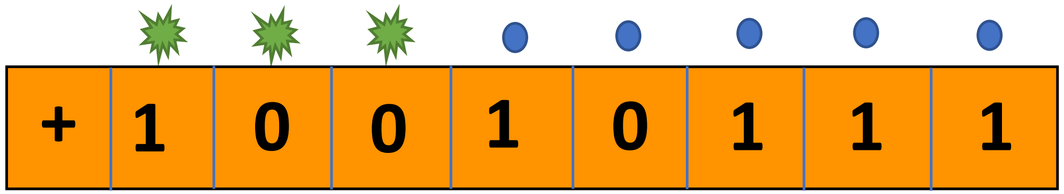
Our Approach: Binary fault injection (BinFI)

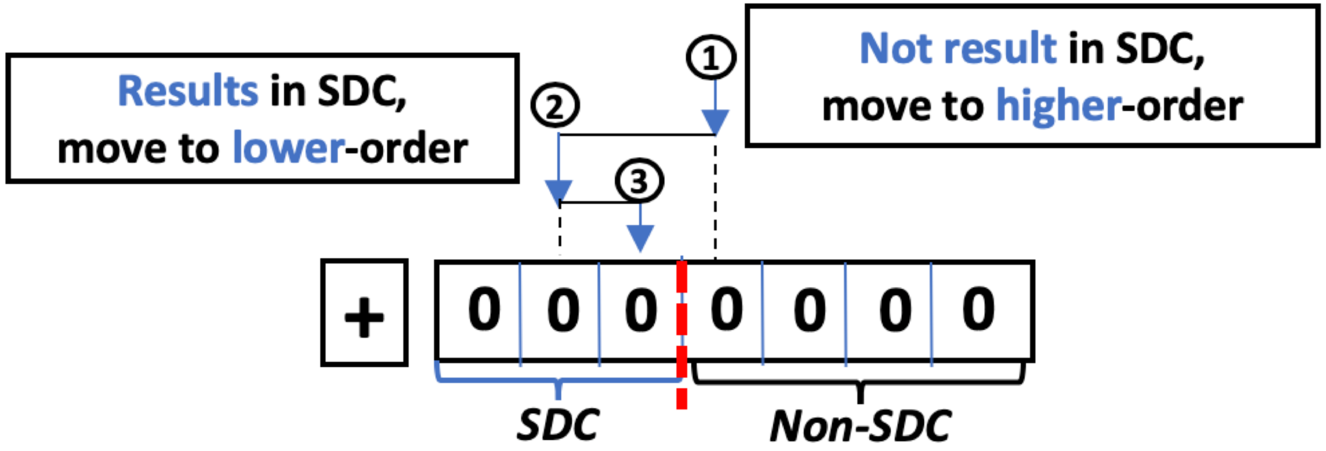
- Identify SDC boundary: *faults at higher-order bits would lead to SDCs and faults from lower-order bits would be masked.*

Our prior work



New observation: analyzing the montonic property





An example – kNN (k=1)

Fault propagation

```
negPixel = tf.negative(testImg)
relativeDistance = tf.add(neighbors, negPixel)
absDistance = tf.abs(relativeDistance)
distance = tf.reduce_sum(absDistance, reduction_indices=1)
nearestNeighbor = tf.arg_min(distance, 0)
```

- Each neighbor ($|N|$ in total) has a distance to the input image ($dis_n, n \in |N|$)
- For the *nearest neighbor* , we have $dis_i < dis_j, \forall j \in |N|, i \neq j$.
- *Fault propagation* (FP): Fault site → tf.abs() → distance (output)
  - Mapping from fault site to output: $FP(bitFlip) = \pm abs(bitFlip)$, + for positive value deviation; − for negative value deviation.
- SDC occurs if the *nearest neighbor* has changed (i.e., $dis_i > dis_j, \exists j \in |n|$), due to bit-flip.

An example (cont.)

$$FP(bitFlip) = \pm abs(bitFlip)$$

```
negPixel = tf.negative(testImg)
relativeDistance = tf.add(neighbors, negPixel)
absDistance = tf.abs(relativeDistance)
distance = tf.reduce_sum(absDistance, reduction_indices=1)
nearestNeighbor = tf.arg_min(distance, 0)
```

Assume fault affects the dis_i, and dis_j remains unchanged:

- $dis'_i = dis_i + abs(bitFlip_m)$, bit-flip occurs at m_{th} bit.

  $dis''_i = dis_i + abs(bitFlip_n), m > n$, i.e., m_{th} *bit is the high-order bit.*
- We have: $abs(bitFlip_m) > abs(bitFlip_n)$, thus $dis'_i > dis''_i$.
- **If fault at m_{th} bit does not lead to SDC (by FI), fault at n_{th} bit (lower-order) will not lead to SDC, without actual FI, because $dis''_i < dis''_i < dis_j$.**

Analyzing ML computations

- Common ML computations in modern DNNs:
  - E.g., AlexNet, VGGNet, InceptionNet, Dave steering model, etc.

Basic	Conv; MatMul; Add (BiasAdd)
Activation	ReLu; ELu;
Pooling	Max-pool; Average-pool
Normalization	Batch normalization (BN); Local Response Normalization (LRN)
Data transformation	Reshape; Concatenate; Dropout
Others	SoftMax; Residual function

- *Monotonic* property of ML computation:
  - Conv computation: $\vec{X} \cdot \vec{W} = \sum x_i w_i, x_i \in \vec{X}, w_i \in \vec{W}$
  - Assume two faults $x_1 > x_2 > 0$ at same location (i.e., different bits).
  - We have: $|x_1 w_i| \geq |x_2 w_i|$, we call Conv is *monotonic*.

- Apply to most of (not all, e.g., LRN) the other computations, e.g., Pooling, ReLu.

Evaluation

- Compare different FI approaches on:
  1. Identifying the critical bits.
  2. Measuring the overall resilience.
  3. Overhead.
- FI tool: TensorFI [1]

Dataset	Dataset Description	ML models
MNIST [9]	Hand-written digits	2-layer NN LeNet-4 [47]
Survive [13]	Prediction of patient survival	kNN
Cifar-10 [4]	General images	AlexNet [46]
ImageNet [24]	General images	VGG16 [71]
German traffic sign [38]	Traffic sign images	VGG11 [71]
Driving [6]	Driving video frames	Nvidia Dave [19] Comma.ai [5]

[1] *https://github.com/DependableSystemsLab/TensorFI*

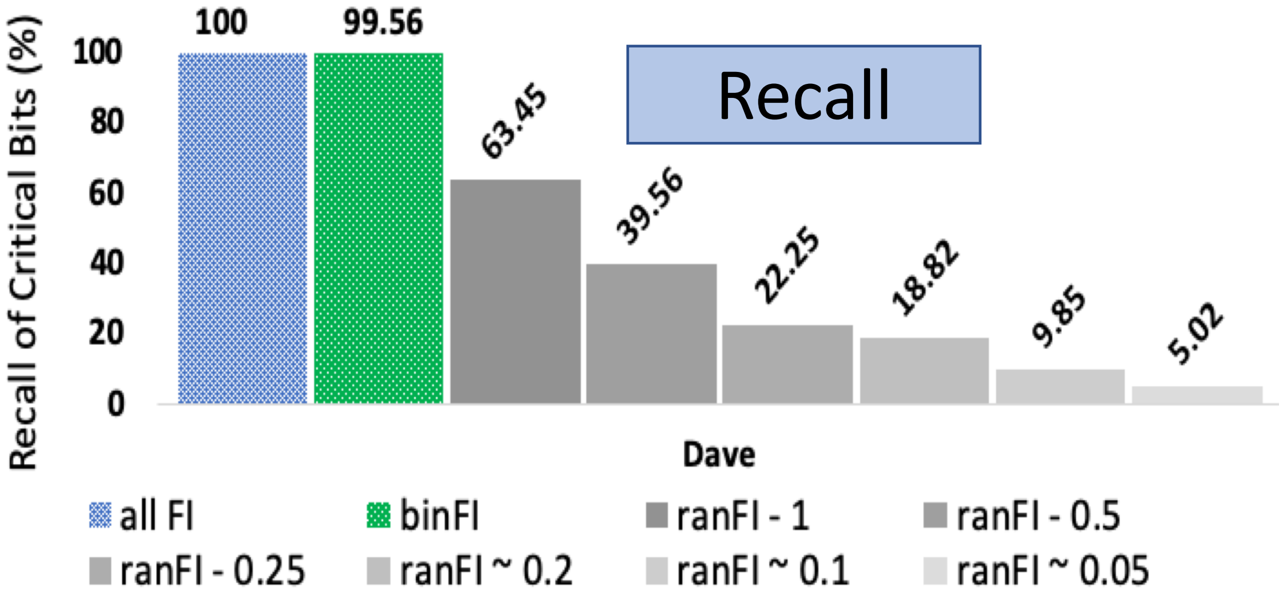
Results

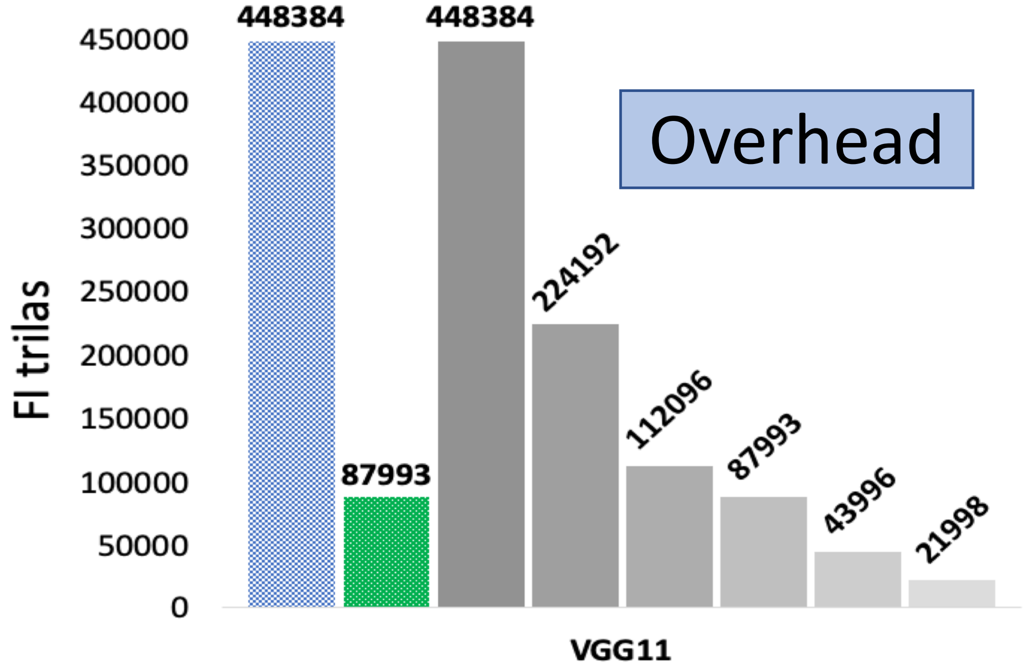
1. BinFI: recall **99+%** of critical bits with **99+%** precision.

   Random FI: recall **less than 65%** with **4x overhead more than binFI**

2. Overall resilience measurement: Random FI ≈ BinFI

3. Overhead: ~20% of that by exhaustive FI (binary search).





Takeaways

- Many common ML computations exhibit monotonicity
- The monotonicity property constrains the fault propagation
- Critical bits in ML program cluster around higher-order bits
- Can be efficiently found through a binary-search like approach

https://github.com/DependableSystemsLab/TensorFI

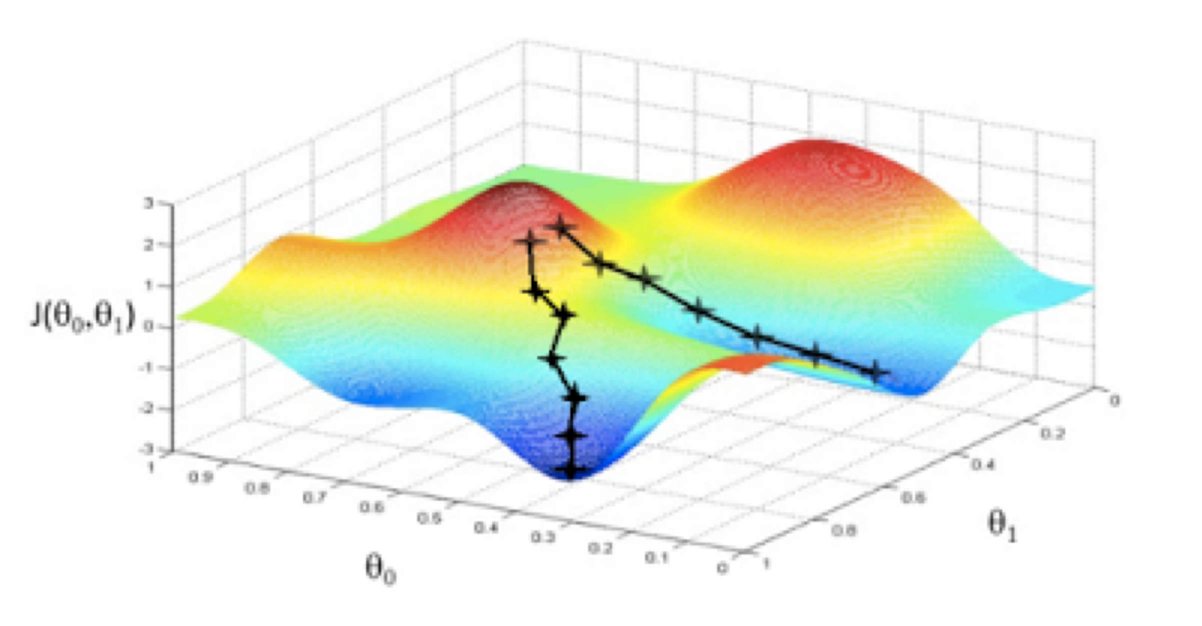
Outline

- Motivation and Goals
- Fault-Injection into Deep Neural Networks [SC’17]
- BinFI: an Efficient Fault Injector for ML systems [SC’19 - to appear]
- Ongoing Work and Conclusions

Ongoing Work: Resilient ML

Deriving ML algorithms resilient to perturbations

- Small changes → Similar outputs
- Convergence properties
- Differences in outputs - safety-critical



Conclusions

- **Machine learning reliability is an important problem**
  - Old problems like soft-errors are still an issue
  - Getting worse with scale and deployment
  - Violation of safety standards (e.g., ISO 26262)
- **Single bit-flip faults can lead to safety-critical outcomes**
  - Need both hardware and software-level protection techniques
- **BinFI: Efficient fault-injection for safety-critical ML systems**
  - Identified safety violations in a fraction of time as exhaustive injections

http://blogs.ubc.ca/karthik/