

# PrioriFI: Efficient Fault Injection for Edge Neural Networks

Olivia Weng, Andres Meza, Nhan Tran, Ryan Kastner

FastML'25  
September 2<sup>nd</sup>, 2025

UC San Diego  Fermilab

# What is **fault injection (FI)**?

- Injecting **faults** into an application to understand its **fault tolerance**

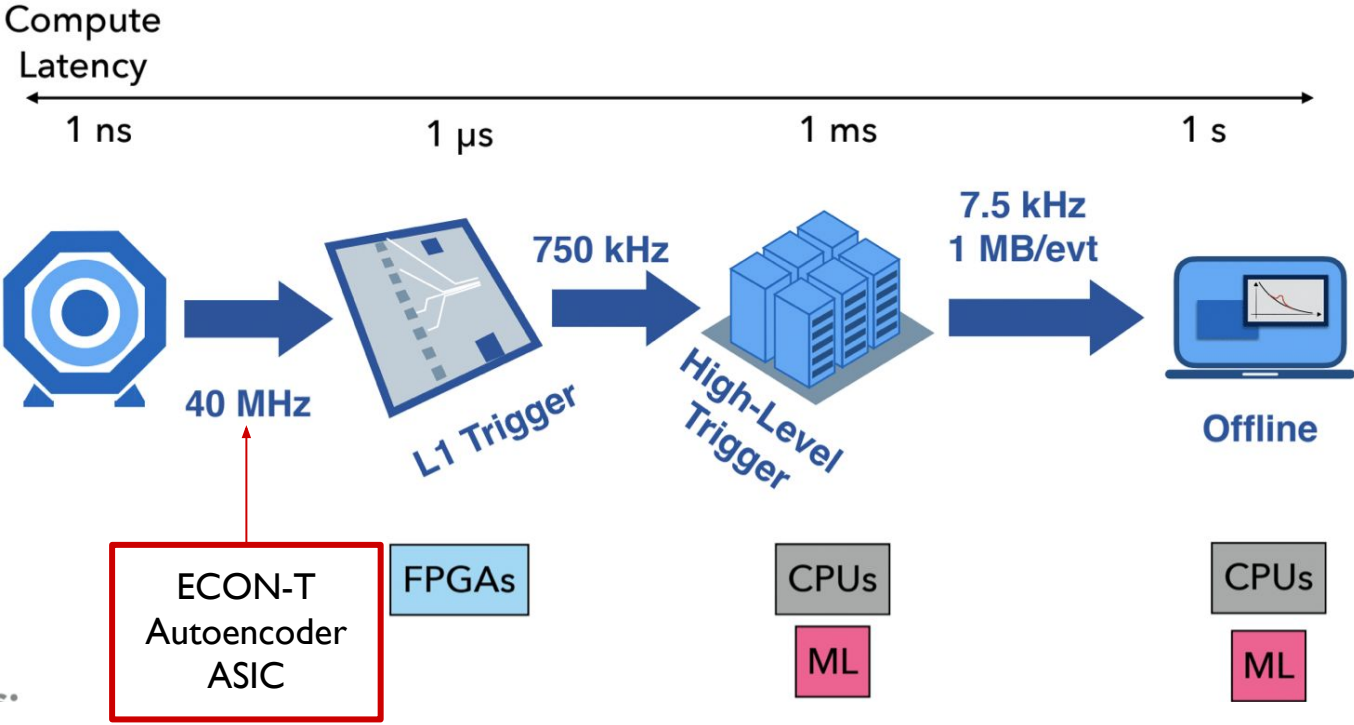
# What is **fault injection (FI)**?

- Injecting **faults** into an application to understand its **fault tolerance**
- Example FI: **Flip bit** in neural network (NN) weight (or bias)

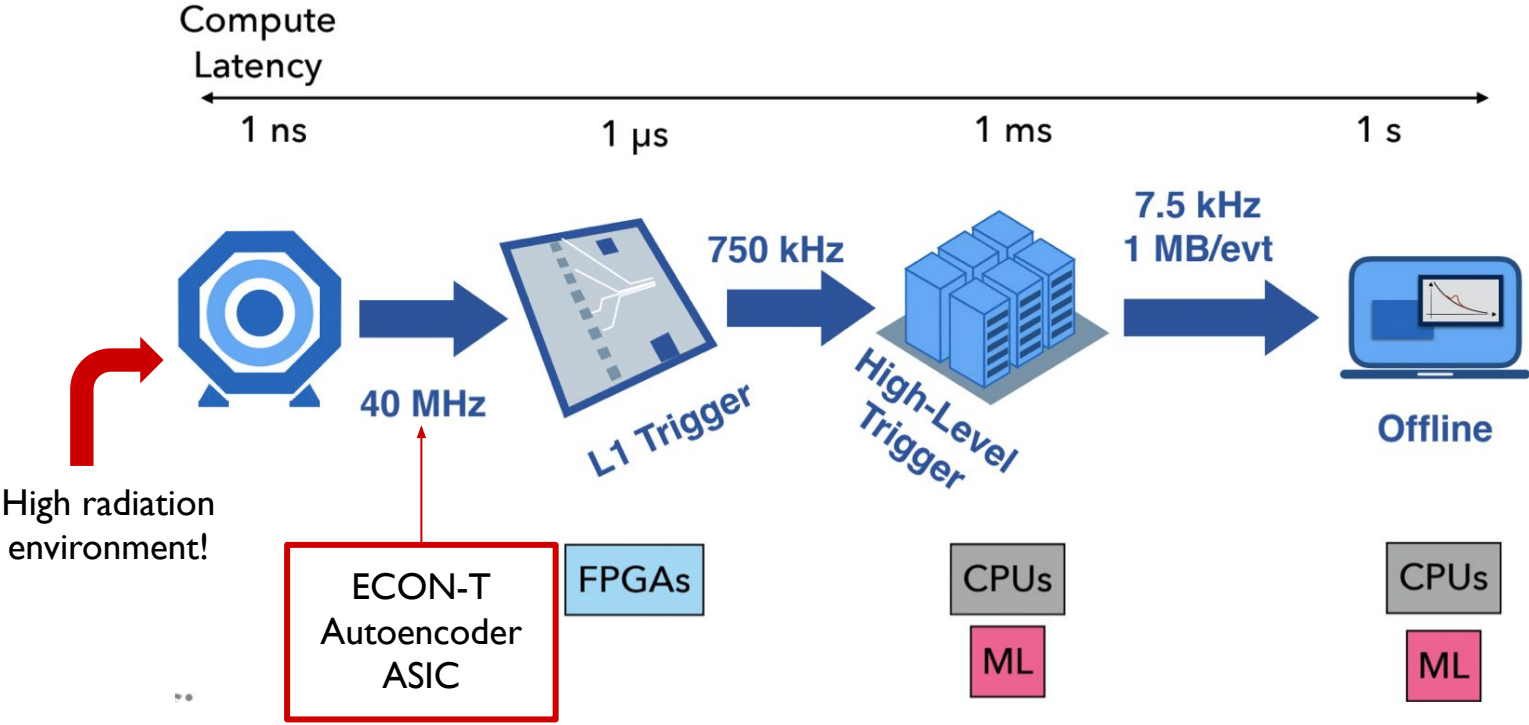
# What is **fault injection (FI)**?

- Injecting **faults** into an application to understand its **fault tolerance**
- Example FI: **Flip bit** in neural network (NN) weight (or bias)
- Why?
  - Simulate NN's performance under harsh conditions (e.g., high radiation at the LHC)

# Example: LHC's CMS Data Processing Pipeline



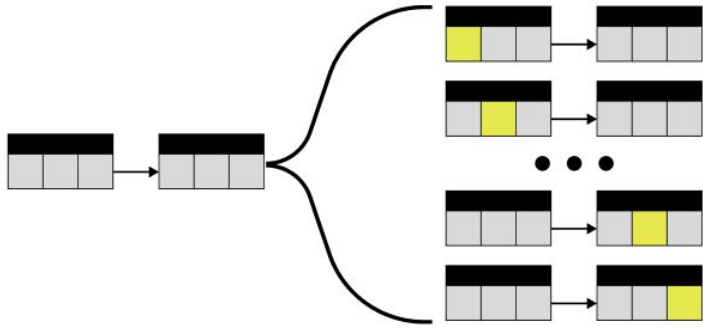
# Example: LHC's CMS Data Processing Pipeline



Ref: [https://indico.fnal.gov/event/46746/contributions/210450/attachments/141293/177902/hirschauer\\_AE\\_CPAD\\_19mar2020.pdf](https://indico.fnal.gov/event/46746/contributions/210450/attachments/141293/177902/hirschauer_AE_CPAD_19mar2020.pdf)

# Fault Injection Campaign

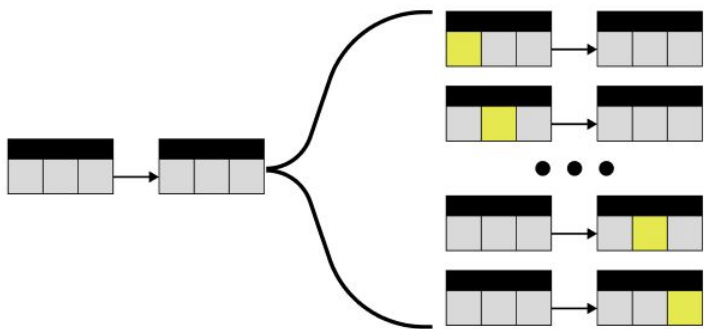
- Conceptually:



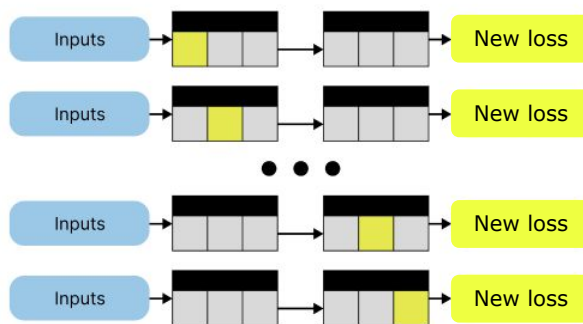
1. Generates X "faulty" variants by flipping a single weight\*bit

# Fault Injection Campaign

- Conceptually:



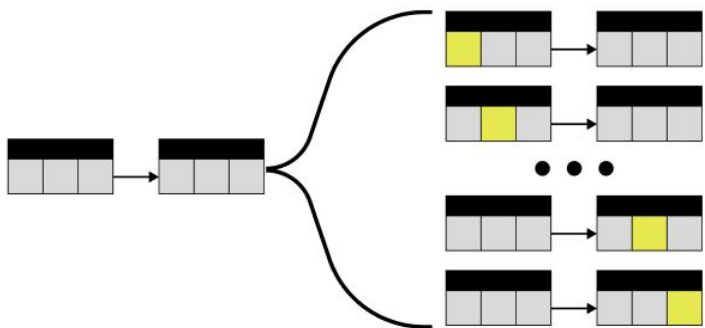
1. Generates X "faulty" variants by flipping a single weight bit



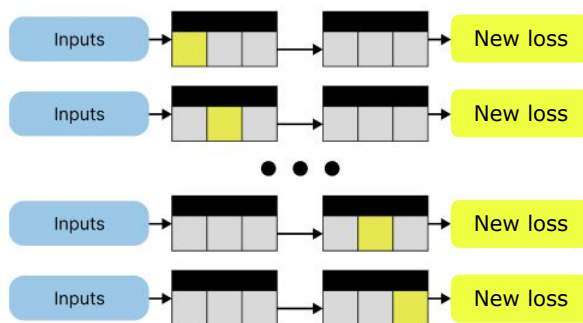
2. Measures the new loss on a set of test inputs

# Fault Injection Campaign

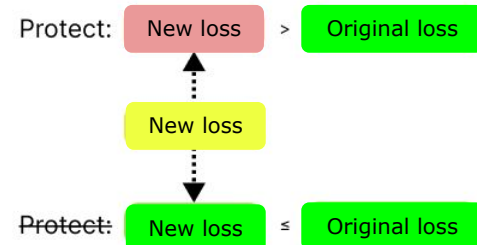
- Conceptually:



1. Generates X "faulty" variants by flipping a single weight bit



2. Measures the new loss on a set of test inputs



3. Determines the weight bits to protect

# Fault Injection Campaign on ECON-T

- Previously, we've conducted FI campaigns on 3 ECON-T models

**Small Pareto** (Total Weight Bits: 10,240)



DENSE → DENSE

**Medium Pareto** (Total Weight Bits: 12,720)



CONV-2D → DENSE

**Large Pareto** (Total Weight Bits: 61,344)



CONV-2D → DENSE

# Fault Injection Campaign on ECON-T

- Previously, we've conducted FI campaigns on 3 ECON-T models

**Small Pareto** (Total Weight Bits: 10,240)



**Medium Pareto** (Total Weight Bits: 12,720)



**Large Pareto** (Total Weight Bits: 61,344)



DENSE

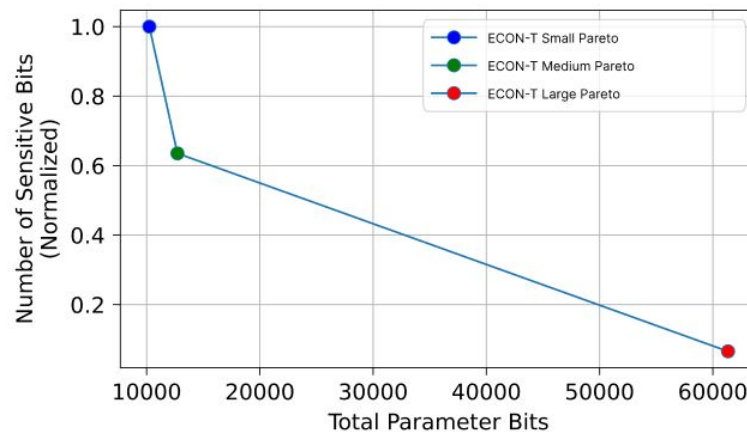
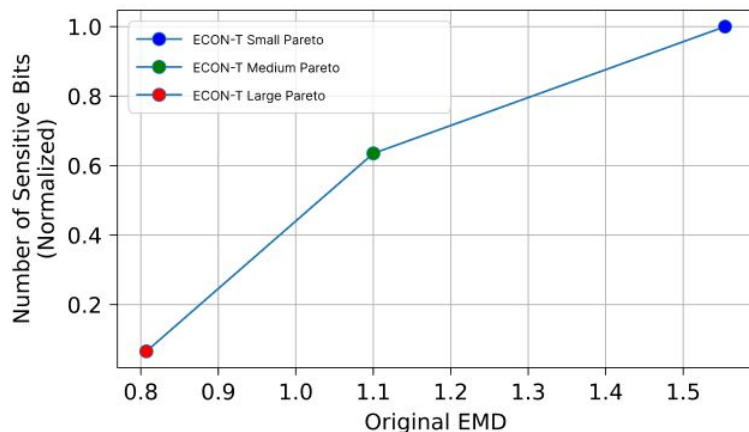
DENSE

CONV-2D

DENSE

CONV-2D

DENSE



In what **order** should we flip the NN bits?

In what **order** should we flip the NN bits?

- Randomly?

# In what **order** should we flip the NN bits?

- Randomly? **Slow**

## In what **order** should we flip the NN bits?

- Randomly? **Slow**
- From highest sensitivity to lowest sensitivity?

## In what **order** should we flip the NN bits?

- Randomly? **Slow**
- From highest sensitivity to lowest sensitivity? **How?**

# Flip from highest to lowest sensitivity

- Idea: From **most** significant bits (MSBs) to **least** significant bits (LSBs)

# Flip from highest to lowest sensitivity

- Idea: From **most** significant bits (MSBs) to **least** significant bits (LSBs)
- Flipping **MSBs** result in **larger changes in magnitude** than lesser significant bits

# Flip from highest to lowest sensitivity

- Idea: From **most** significant bits (MSBs) to **least** significant bits (LSBs)
- Flipping **MSBs** result in **larger changes in magnitude** than lesser significant bits
- Known as bit-level monotonicity
  - Many prior works rely on bit-level monotonicity to speed up FI campaigns

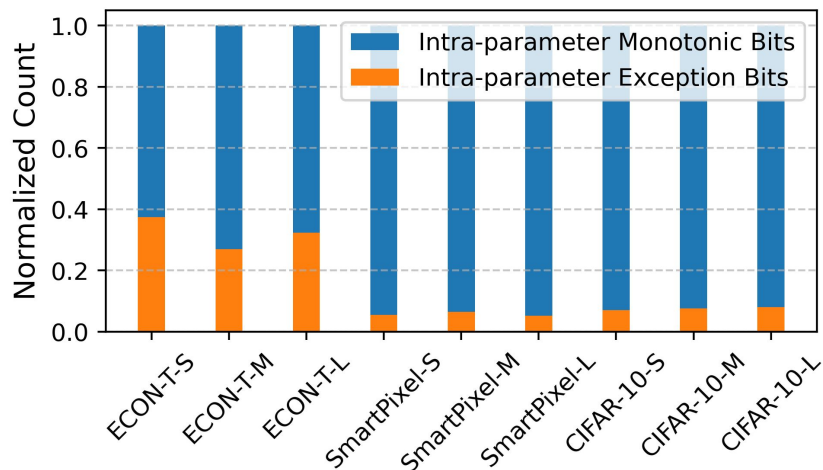
Does bit-level monotonicity correlate with sensitivity?

# Does bit-level monotonicity correlate with sensitivity?

- Two kinds of bit-level monotonicity
  - Intra-parameter (within a parameter)
  - Inter-parameter (between parameters)

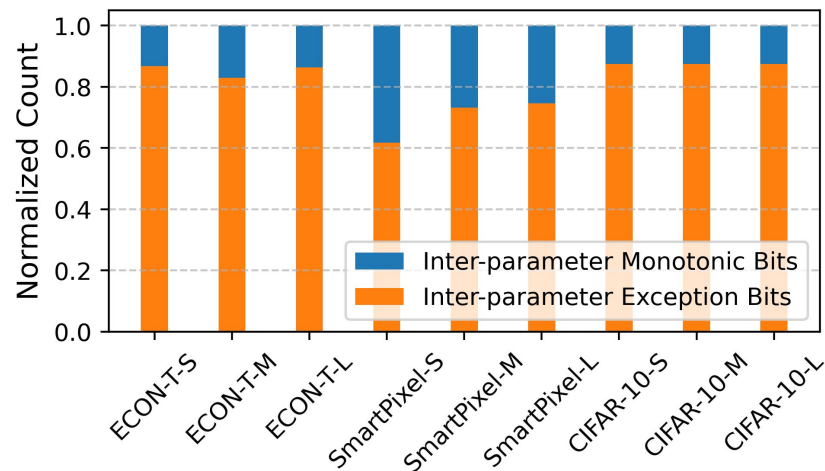
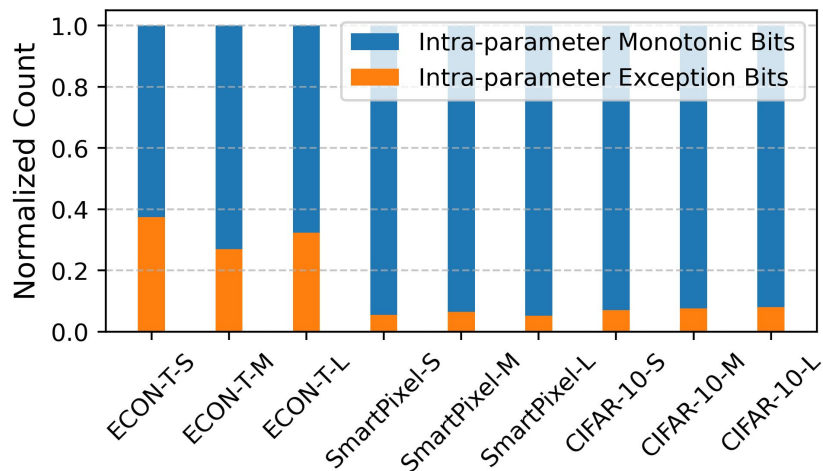
# Does bit-level monotonicity correlate with sensitivity?

- Two kinds of bit-level monotonicity
  - Intra-parameter (within a parameter)
  - Inter-parameter (between parameters)



# Does bit-level monotonicity correlate with sensitivity?

- Two kinds of bit-level monotonicity
  - Intra-parameter (within a parameter)
  - Inter-parameter (between parameters)



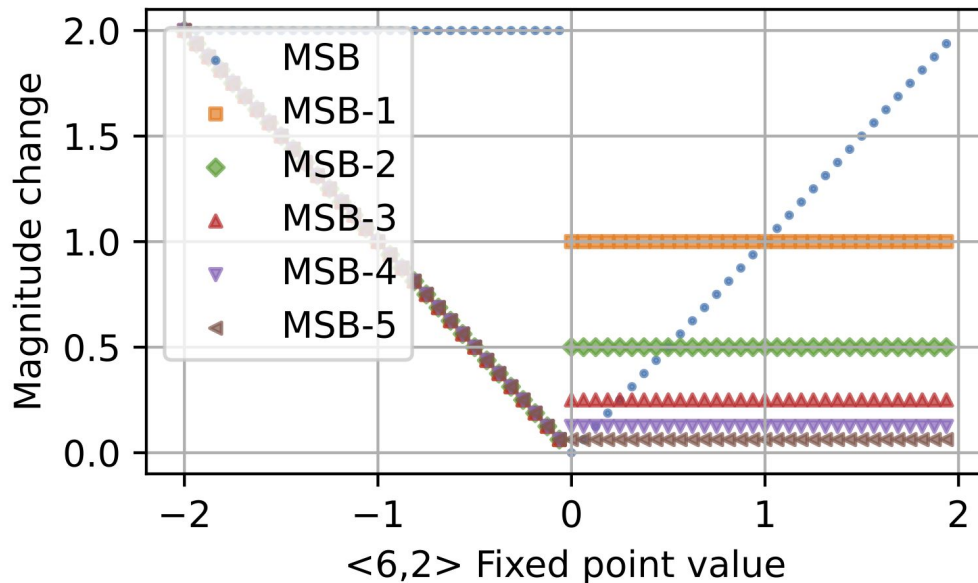
There are **many exceptions** to bit-level monotonicity!

# Why do exceptions to bit-level monotonicity exist?

- Faults get masked by operations downstream in the NN, e.g., ReLU

# Why do exceptions to bit-level monotonicity exist?

- Faults get masked by operations downstream in the NN, e.g., ReLU
  - Ex: flip a bit then apply ReLU, plotting the magnitude change



How to **account** for the **exceptions** to bit-level monotonicity?

# PrioriFI

- Idea: Use information gained from **previous FIs** to **guide** the FI campaign **on the fly**
  - Previous FIs provide a good heuristic for bit sensitivity

# PrioriFI

- Idea: Use information gained from **previous FIs** to **guide** the FI campaign **on the fly**
  - Previous FIs provide a good heuristic for bit sensitivity
  - **Prioritize** the bits most likely to be sensitive based on previous FIs
    - Accounts for the exceptions to bit-level monotonicity

# PrioriFI Algorithm

1. Use the Hessian to rank the parameters by sensitivity

# PrioriFI Algorithm

1. Use the Hessian to rank the parameters by sensitivity
2. Rank MSBs by Hessian, then MSB-Is by Hessian, etc. for the initial ranking

# PrioriFI Algorithm

1. Use the Hessian to rank the parameters by sensitivity
2. Rank MSBs by Hessian, then MSB-Is by Hessian, etc. for the initial ranking
3. Prioritize the next bit to flip

# PrioriFI

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1_s : B_{H_1}^1 \mid B_{H_2}^1 \mid \dots \mid B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2_s : B_{H_1}^2 \mid B_{H_2}^2 \mid \dots \mid B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m_s : B_{H_1}^m \mid B_{H_2}^m \mid \dots \mid B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

$t = 0$

# PrioriFI

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

## 2. Flip first bit per list

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}]$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 0$

# PrioriFI

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

## 2. Flip first bit per list

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}]$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 0$

## 3. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 2$$

$t = 1$

# PrioriFI

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1_s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2_s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m_s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

## 2. Flip first bit per list

Bit Lists	$\Delta C_s$
$B^1_s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2_s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}]$
$\vdots$	$\vdots$
$B^m_s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 0$

## 3. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 2$$

## 4. Flip next prioritized bit

Bit Lists	$\Delta C_s$
$B^1_s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2_s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}, \Delta C_{B_{H_2}^2}]$
$\vdots$	$\vdots$
$B^m_s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 1$

# PrioriFI

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

## 2. Flip first bit per list

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}]$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 0$

## 3. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 2$$

## 4. Flip next prioritized bit

Bit Lists	$\Delta C_s$
$B^1 s : B_{H_1}^1 B_{H_2}^1 \dots B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s : B_{H_1}^2 B_{H_2}^2 \dots B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}, \Delta C_{B_{H_2}^2}]$
$\vdots$	$\vdots$
$B^m s : B_{H_1}^m B_{H_2}^m \dots B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 1$

## 5. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 1$$

$t = 2$

# PrioriFl

## 1. Initial Hessian Bit Rankings

Bit Lists	$\Delta C_s$
$B^1 s$ : $B_{H_1}^1$ $B_{H_2}^1$ ... $B_{H_n}^1$	$\Delta C_{B^1} : \emptyset$
$B^2 s$ : $B_{H_1}^2$ $B_{H_2}^2$ ... $B_{H_n}^2$	$\Delta C_{B^2} : \emptyset$
$\vdots$	$\vdots$
$B^m s$ : $B_{H_1}^m$ $B_{H_2}^m$ ... $B_{H_n}^m$	$\Delta C_{B^m} : \emptyset$

## 2. Flip first bit per list

Bit Lists	$\Delta C_s$
$B^1 s$ : $B_{H_1}^1$ $B_{H_2}^1$ ... $B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s$ : $B_{H_1}^2$ $B_{H_2}^2$ ... $B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}]$
$\vdots$	$\vdots$
$B^m s$ : $B_{H_1}^m$ $B_{H_2}^m$ ... $B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 0$

## 3. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 2$$

## 4. Flip next prioritized bit

Bit Lists	$\Delta C_s$
$B^1 s$ : $B_{H_1}^1$ $B_{H_2}^1$ ... $B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}]$
$B^2 s$ : $B_{H_1}^2$ $B_{H_2}^2$ ... $B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}, \Delta C_{B_{H_2}^2}]$
$\vdots$	$\vdots$
$B^m s$ : $B_{H_1}^m$ $B_{H_2}^m$ ... $B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 1$

## 5. Prioritize next bit flip

$$ms = \text{medianLastK}(\Delta C_s, k)$$

$$//ms = [\Delta \tilde{C}_{B^1}, \Delta \tilde{C}_{B^2}, \dots, \Delta \tilde{C}_{B^m}]$$

$$\text{argmax}(ms) = 1$$

## 6. Flip next prioritized bit

Bit Lists	$\Delta C_s$
$B^1 s$ : $B_{H_1}^1$ $B_{H_2}^1$ ... $B_{H_n}^1$	$\Delta C_{B^1} : [\Delta C_{B_{H_1}^1}, \Delta C_{B_{H_2}^1}]$
$B^2 s$ : $B_{H_1}^2$ $B_{H_2}^2$ ... $B_{H_n}^2$	$\Delta C_{B^2} : [\Delta C_{B_{H_1}^2}, \Delta C_{B_{H_2}^2}]$
$\vdots$	$\vdots$
$B^m s$ : $B_{H_1}^m$ $B_{H_2}^m$ ... $B_{H_n}^m$	$\Delta C_{B^m} : [\Delta C_{B_{H_1}^m}]$

$t = 2$

How does PrioriFl compare with a perfect ranking?

# How does PrioriFI compare with a perfect ranking?

- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta\text{EMD}$ 
  - $\Delta\text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$

# How does PrioriFl compare with a perfect ranking?

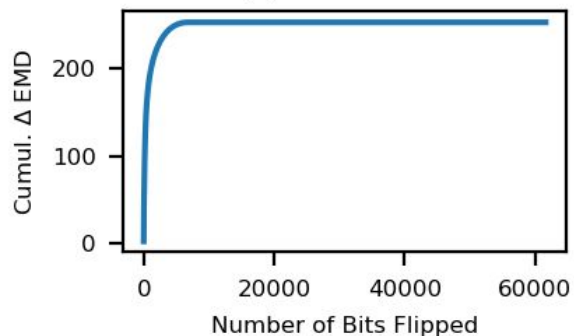
- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta\text{EMD}$ 
  - $\Delta\text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$
  - Measures how much error we accumulate as we flip more bits

# How does PrioriFl compare with a perfect ranking?

- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta \text{EMD}$ 
  - $\Delta \text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$
  - Measures how much error we accumulate as we flip more bits

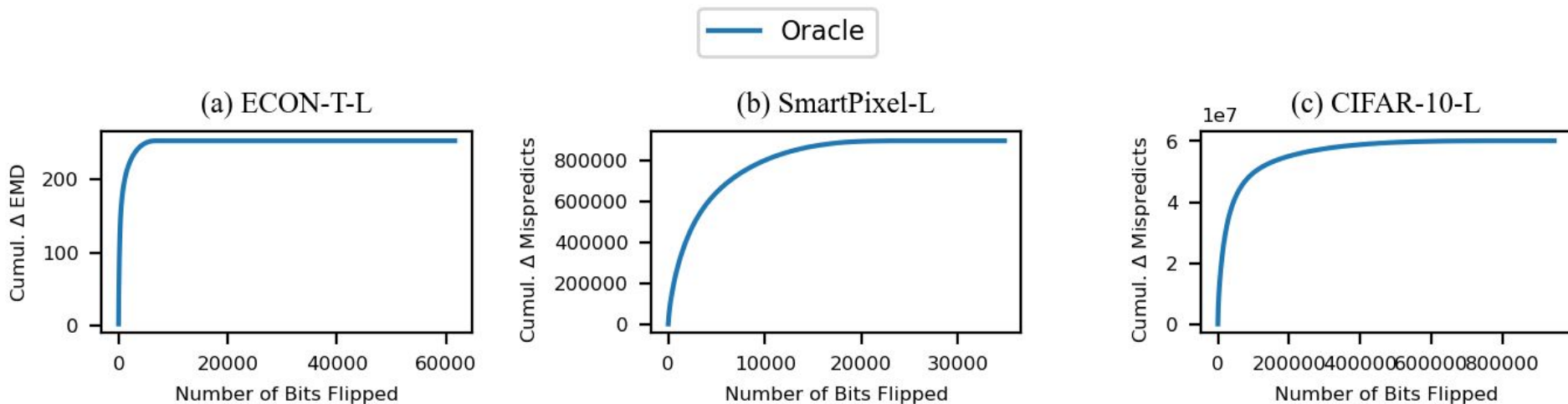
— Oracle

(a) ECON-T-L



# How does PrioriFl compare with a perfect ranking?

- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta EMD$ 
  - $\Delta EMD = \max(\overline{EMD}_{faulty} - \overline{EMD}_{faultless}, 0)$
  - Measures how much error we accumulate as we flip more bits

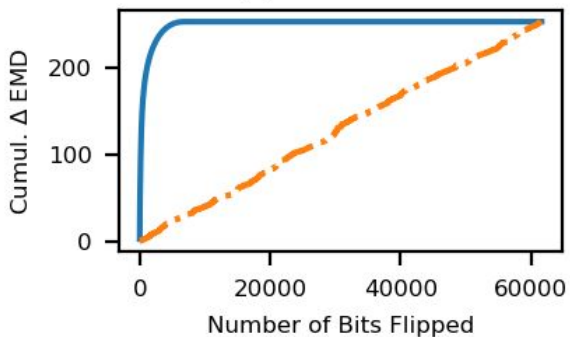


# How does PrioriFl compare with a perfect ranking?

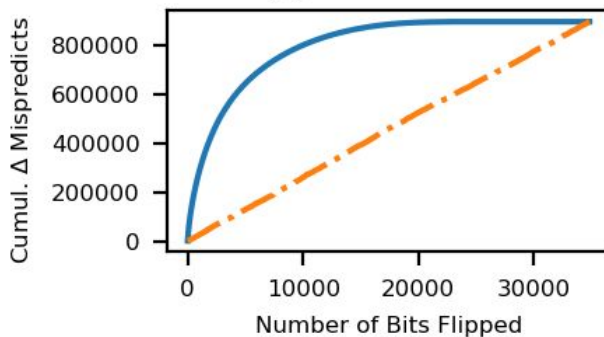
- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta \text{EMD}$ 
  - $\Delta \text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$
  - Measures how much error we accumulate as we flip more bits



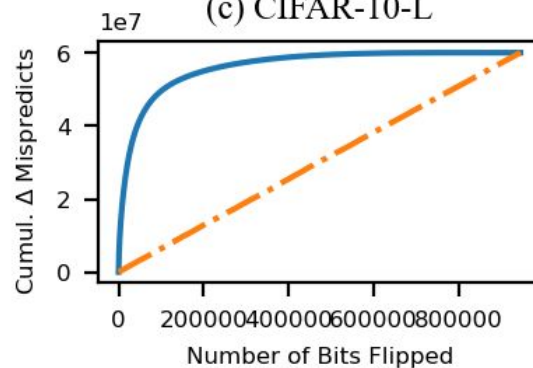
(a) ECON-T-L



(b) SmartPixel-L



(c) CIFAR-10-L

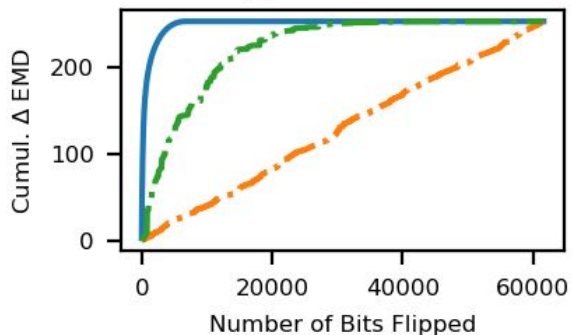


# How does PrioriFl compare with a perfect ranking?

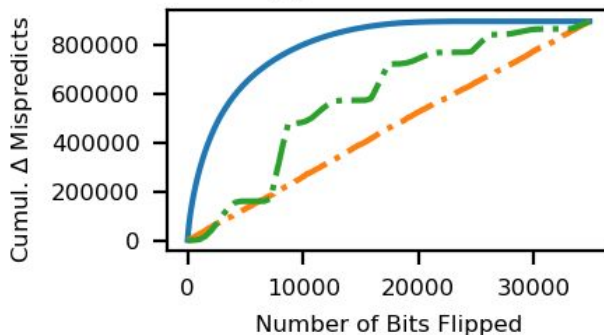
- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta EMD$ 
  - $\Delta EMD = \max(\overline{EMD}_{faulty} - \overline{EMD}_{faultless}, 0)$
  - Measures how much error we accumulate as we flip more bits



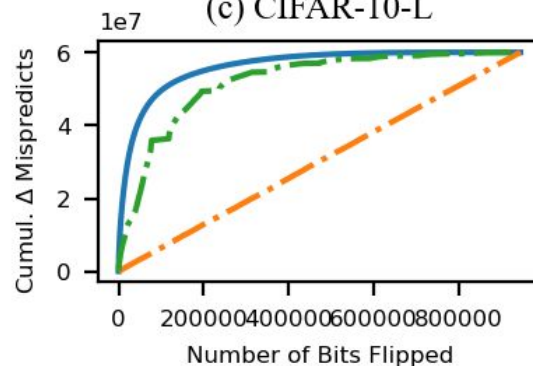
(a) ECON-T-L



(b) SmartPixel-L



(c) CIFAR-10-L

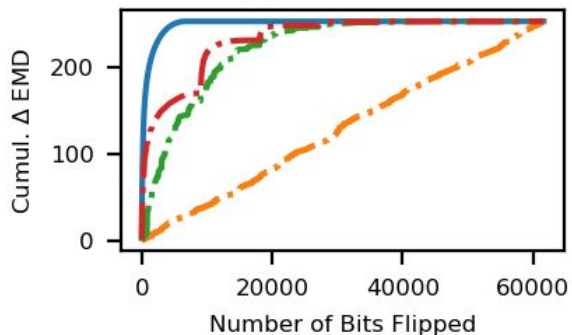


# How does PrioriFl compare with a perfect ranking?

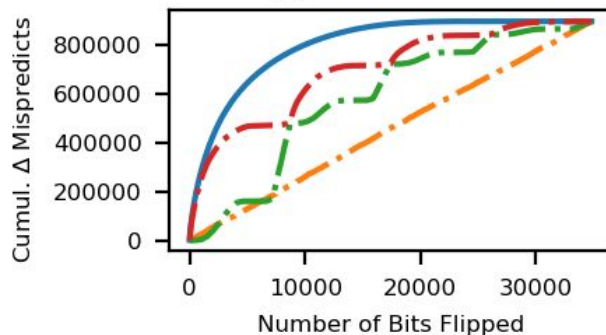
- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta \text{EMD}$ 
  - $\Delta \text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$
  - Measures how much error we accumulate as we flip more bits



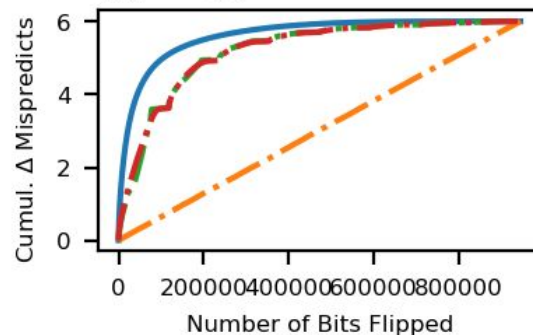
(a) ECON-T-L



(b) SmartPixel-L



(c) CIFAR-10-L

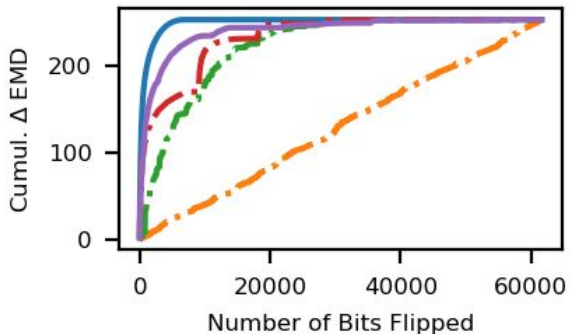


# How does PrioriFI compare with a perfect ranking?

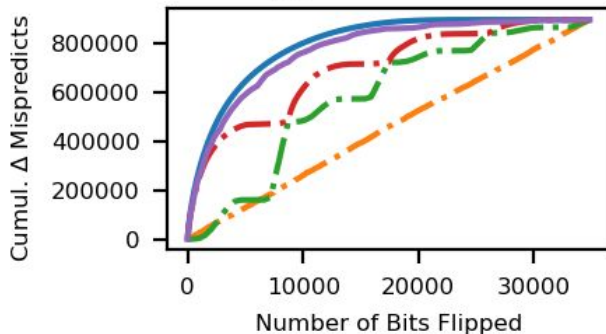
- Cumulative Delta EMD =  $\sum_{i=0}^{\text{num\_bits\_flipped}} \Delta \text{EMD}$ 
  - $\Delta \text{EMD} = \max(\overline{\text{EMD}}_{\text{faulty}} - \overline{\text{EMD}}_{\text{faultless}}, 0)$
  - Measures how much error we accumulate as we flip more bits



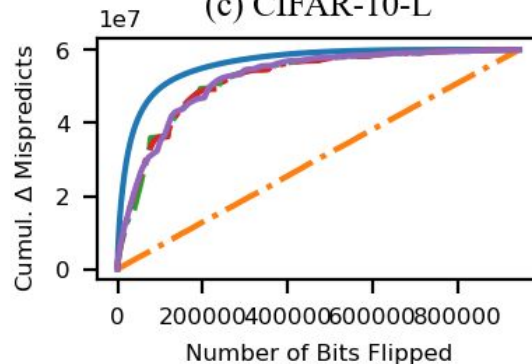
(a) ECON-T-L



(b) SmartPixel-L



(c) CIFAR-10-L



Some models are more monotonic than others

# Some models are more monotonic than others

- We define a **model monoscore**
  - $+1$  = perfectly monotonic
  - $0$  = equally monotonic & non-monotonic
  - $-1$  = perfectly non-monotonic

# Some models are more monotonic than others

- We define a **model monoscore**
  - +1 = perfectly monotonic
  - 0 = equally monotonic & non-monotonic
  - -1 = perfectly non-monotonic

Model	Model monoscore
ECON-T-L	0.32
SmartPixel-L	0.25
CIFAR-10-L	0.40

# Some models are more monotonic than others

- We define a **model monoscore**
  - +1 = perfectly monotonic
  - 0 = equally monotonic & non-monotonic
  - -1 = perfectly non-monotonic

Model	Model monoscore
ECON-T-L	0.32
SmartPixel-L	0.25
CIFAR-10-L	0.40

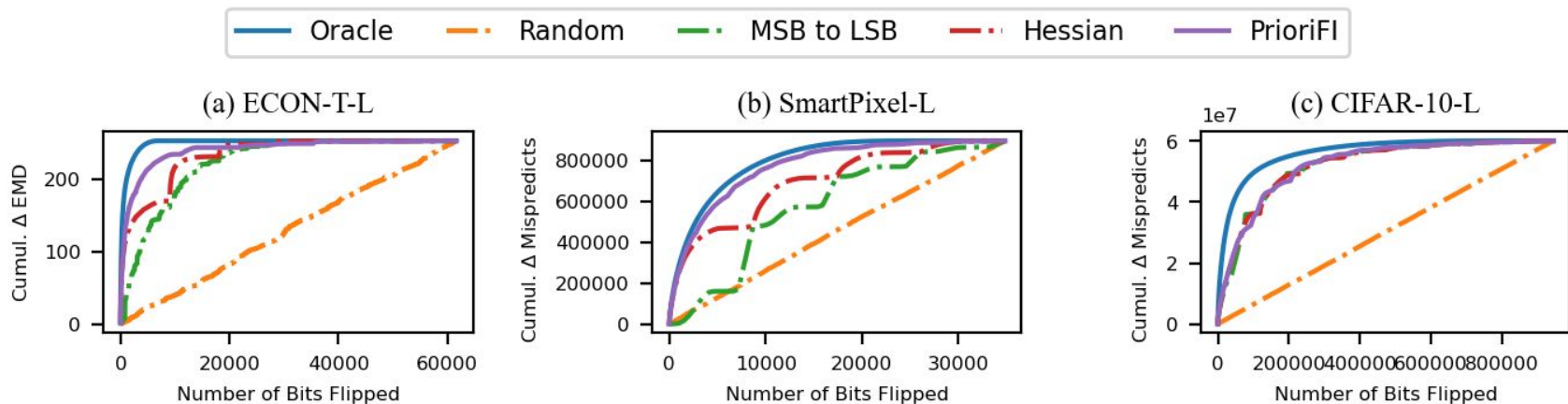
Most monotonic

# Some models are more monotonic than others

- We define a **model monoscore**
  - +1 = perfectly monotonic
  - 0 = equally monotonic & non-monotonic
  - -1 = perfectly non-monotonic

Model	Model monoscore
ECON-T-L	0.32
SmartPixel-L	0.25
CIFAR-10-L	0.40

Most monotonic



# PrioriFI FI Speedup

- PrioriFI finds sensitive bits faster or on par with prior work
  - Ex: Finding 50% of the sensitive bits compared with Random

Model	Hessian Speedup	PrioriFI Speedup
ECON-T-L	22.7 X	<b>37.5 X</b>
SmartPixel-L	4.2 X	<b>6.0 X</b>
CIFAR-10-L	<b>7.0 X</b>	6.7 X

# PrioriFI

- **Prioritizes** the bits most likely to be sensitive based on previous FIs
- Uses **previous FIs** as a heuristic for future FIs
- Helps designers **quickly evaluate** the sensitivity of edge NNs

# Backup

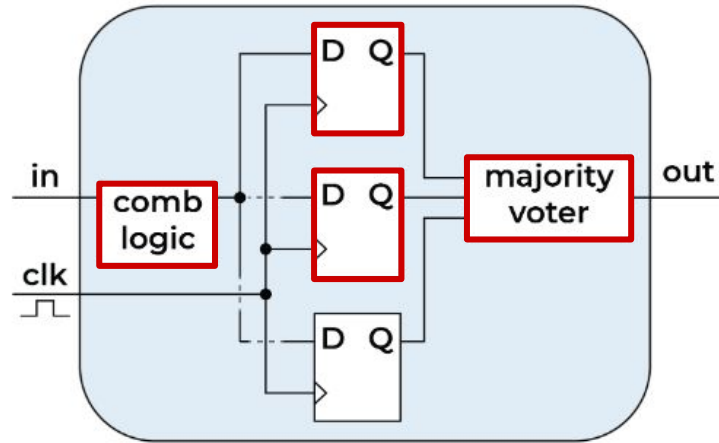
How **fault tolerant** is the ECON-T NN?

# ECON-T Radiation Tolerance

- ECON-T employs **triple modular redundancy (TMR)** to its registers

# ECON-T Radiation Tolerance

- ECON-T employs **triple modular redundancy (TMR)** to its registers



**TMR incurs  $\geq 200\%$  area overhead!**

How can we **reduce** radiation tolerance **costs**?

Observation: Tolerance only applied to **hardware**

Observation: Tolerance only applied to **hardware**

What about **software**?

# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models
- Current features:
  - Fault injection
  - **Bit-level sensitivity metrics**

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits
  - **High** sensitivity:  $\text{EMD}_{\text{flipped}} \gg \text{Original EMD}$

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits
  - **High** sensitivity:  $\text{EMD\_flipped} \gg \text{Original EMD}$
  - **Low** sensitivity:  $\text{EMD\_flipped} \leq \text{Original EMD}$

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - Gradient
  - Hessian

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - Gradient
  - Hessian

How do these metrics **compare** with a perfect ranking?

# Some models are more monotonic than others

- We define a **model monoscore**
  - +1 = perfectly monotonic
  - 0 = equally monotonic & non-monotonic
  - -1 = perfectly non-monotonic

Model	Model monoscore	AUC (norm. to oracle)
ECON-T-L	0.32	0.96
SmartPixel-L	0.25	0.97
CIFAR-10-L	0.40	0.93

Most monotonic

