# FKeras: A Sensitivity Analysis Tool for Edge Neural Networks

**Olivia Weng**, Andres Meza, Quinlan Bock, Benjamin Hawks, Javier Campos, Nhan Tran, Javier Duarte, Ryan Kastner

September 27th, 2023 | FastML'23

Fermilab UC San Diego hls4ml

# Why analyze the **sensitivity** of edge NNs?

# Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

# Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

- They are put through a lot:

# Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

- They are put through a lot:
  - Pruning
  - Quantization
  - Hardware faults

# Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

- They are put through a lot:
  - Pruning
  - Quantization
  - **Hardware faults**

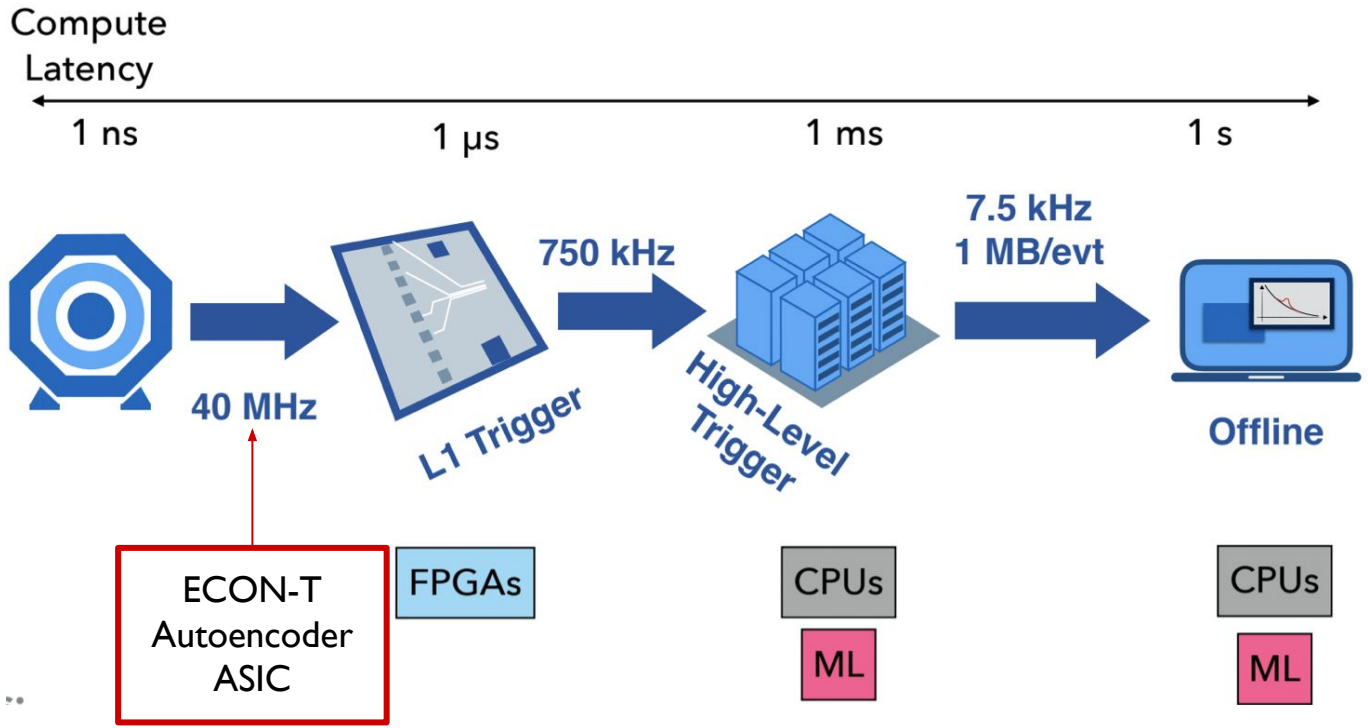
# Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

- They are put through a lot:
  - Pruning
  - Quantization
  - **Hardware faults**

When do **hardware faults** occur?

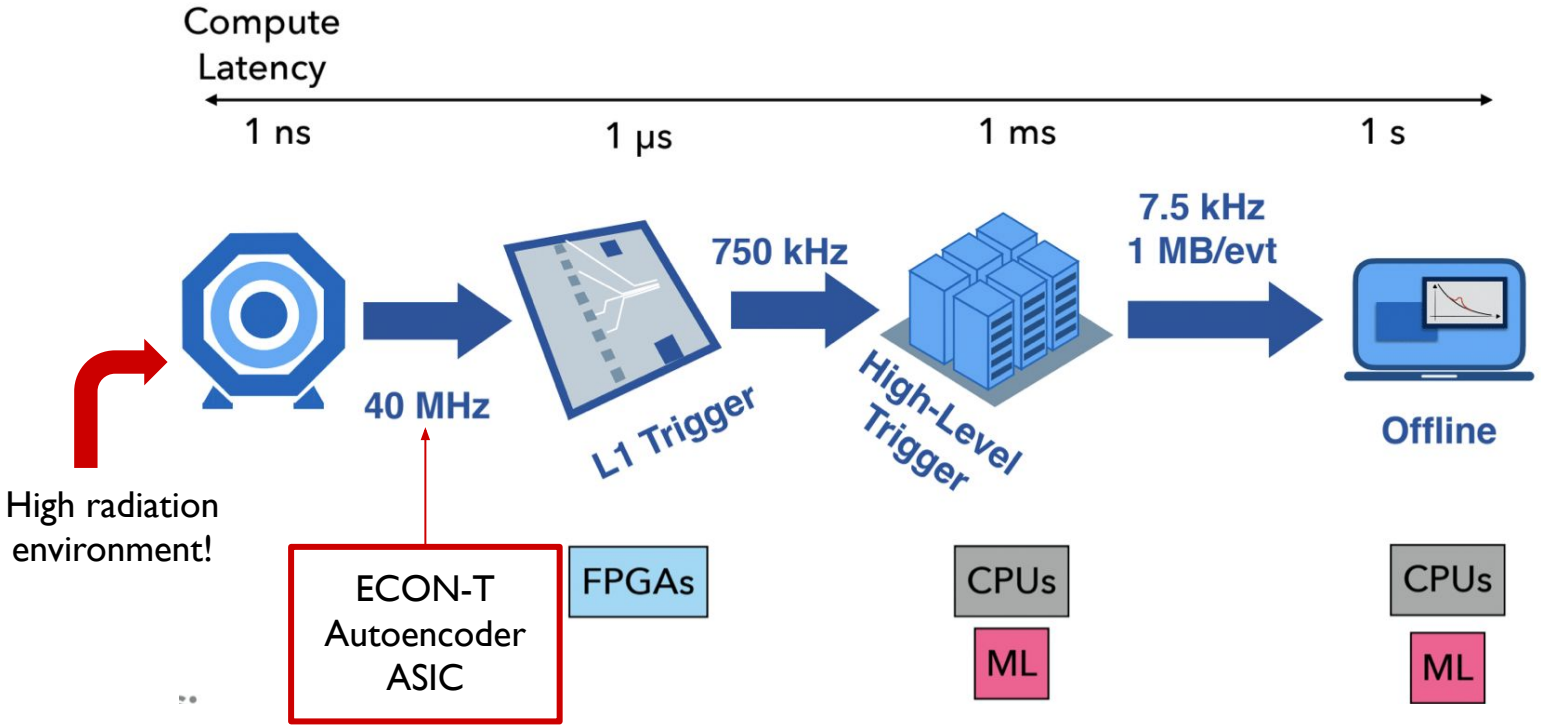
# Example: LHC's CMS Data Processing Pipeline






Ref: https://indico.fnal.gov/event/46746/contributions/210450/attachments/141293/177902/hirschauer_AE_CPAD_19mar2020.pdf

# Example: LHC's CMS Data Processing Pipeline




Ref: https://indico.fnal.gov/event/46746/contributions/210450/attachments/141293/177902/hirschauer_AE_CPAD_19mar2020.pdf

How does the ECON-T Autoencoder **tolerate** radiation?

# ECON-T Radiation Tolerance

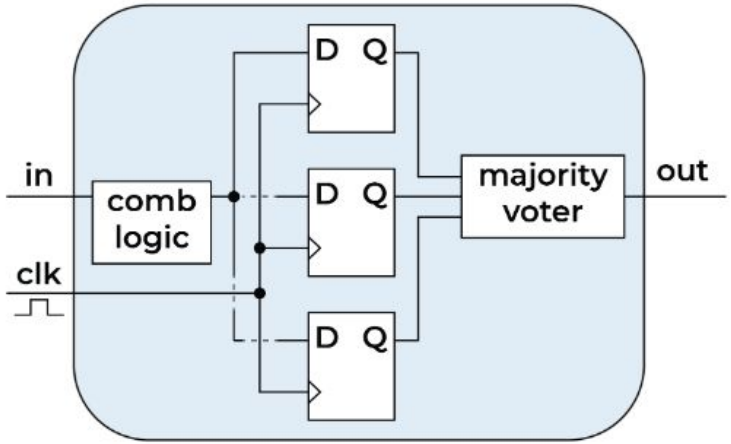
- ECON-T employs **triple modular redundancy (TMR)** to its registers



Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

# ECON-T Radiation Tolerance

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

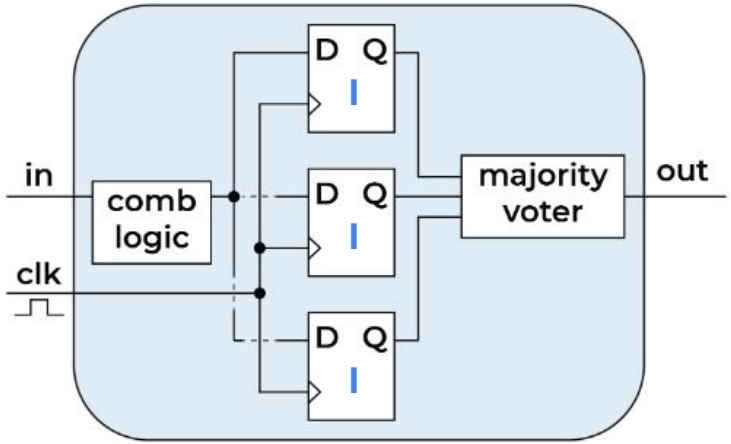





Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

# ECON-T Radiation Tolerance

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

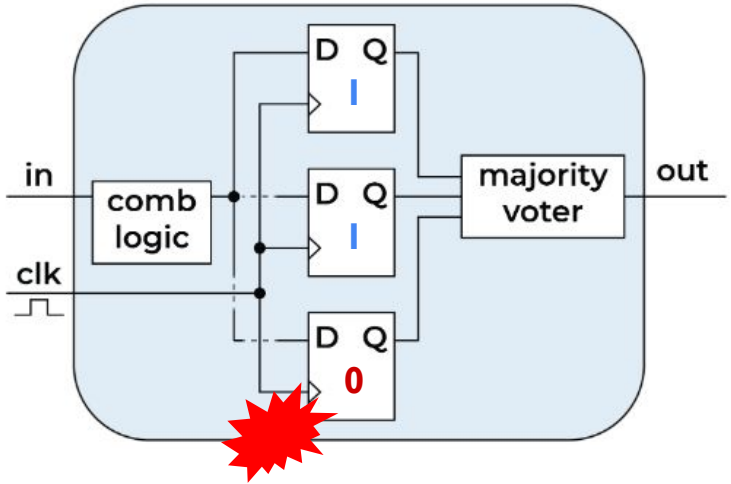





Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

# ECON-T Radiation Tolerance

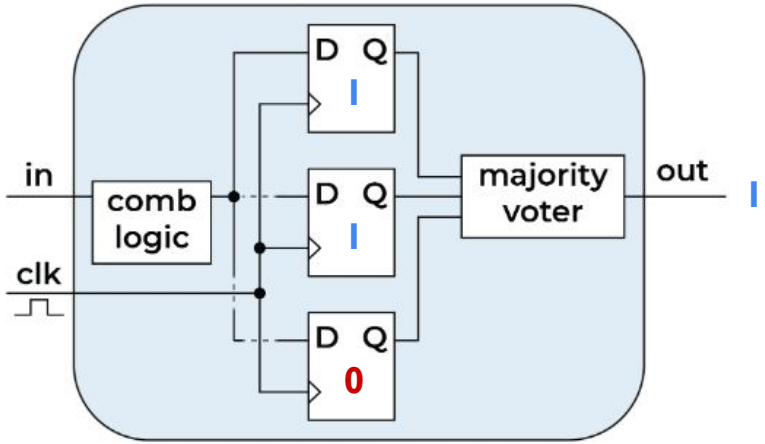
- ECON-T employs **triple modular redundancy (TMR)** to its registers




Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

# ECON-T Radiation Tolerance

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

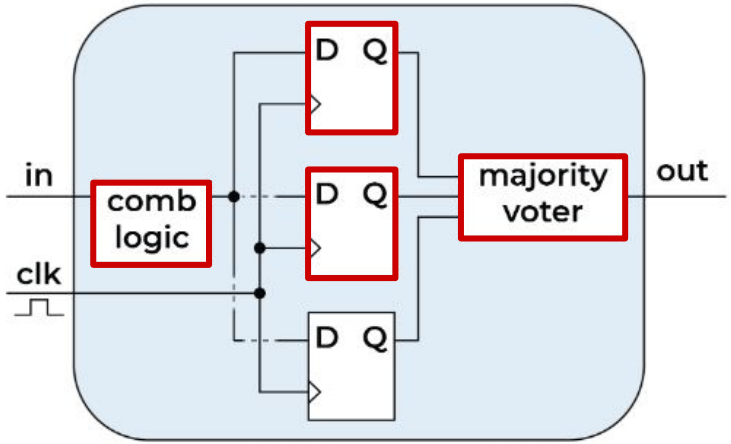





Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

# ECON-T Radiation Tolerance

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




**TMR incurs ≥200% area overhead!**



Ref: Di Guglielmo et al. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. IEEE Trans. Nucl. Sci.'21.

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

Observation: Tolerance only applied to **hardware**

Observation: Tolerance only applied to **hardware**

What about **software**?

How should we assess the **fault sensitivity** of NN **software**?

# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models

# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models

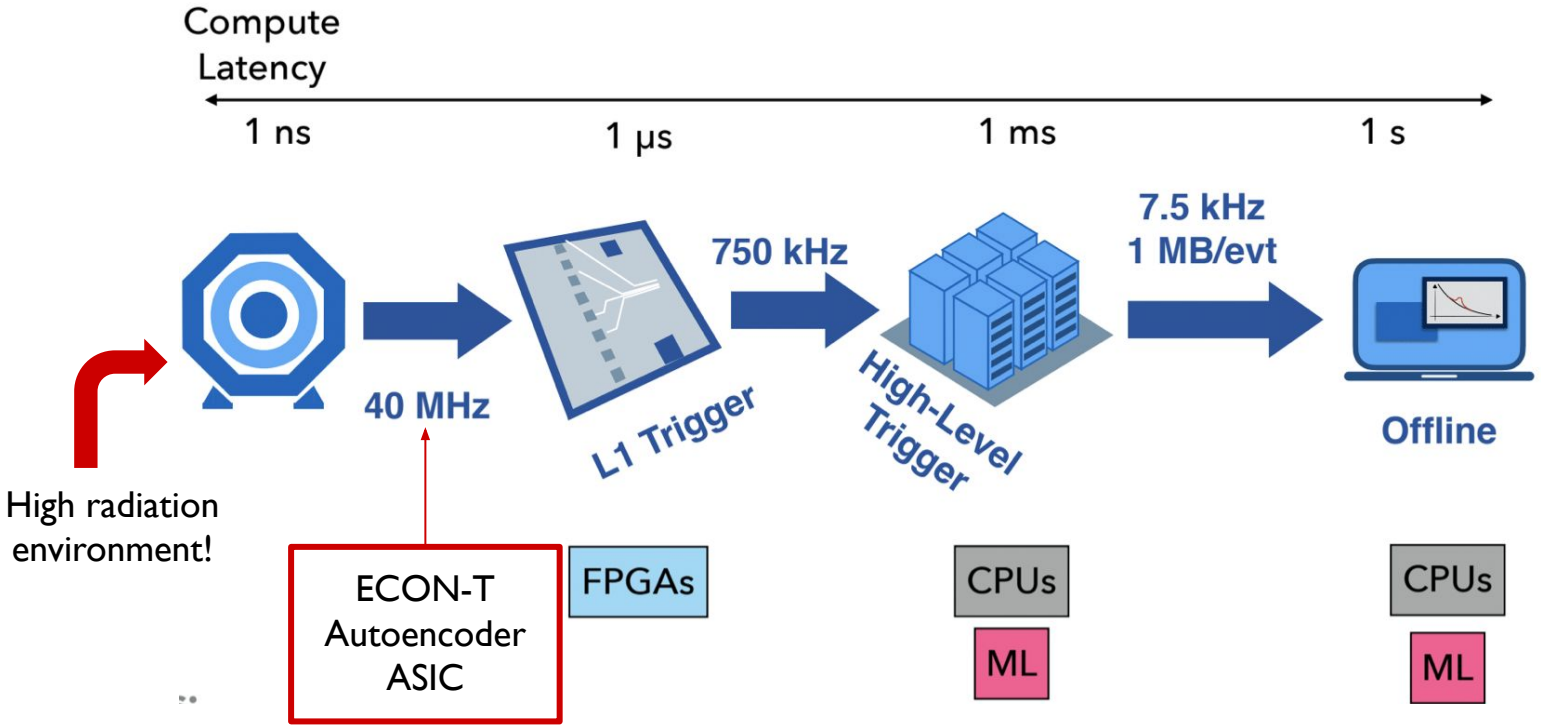
- Current features:
  - Fault injection
  - Bit-level sensitivity metrics

# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models

- Current features:
  - **Fault injection**
  - Bit-level sensitivity metrics

# Example: LHC's CMS Data Processing Pipeline

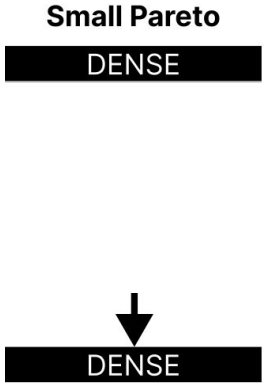





Ref: https://indico.fnal.gov/event/46746/contributions/210450/attachments/141293/177902/hirschauer_AE_CPAD_19mar2020.pdf

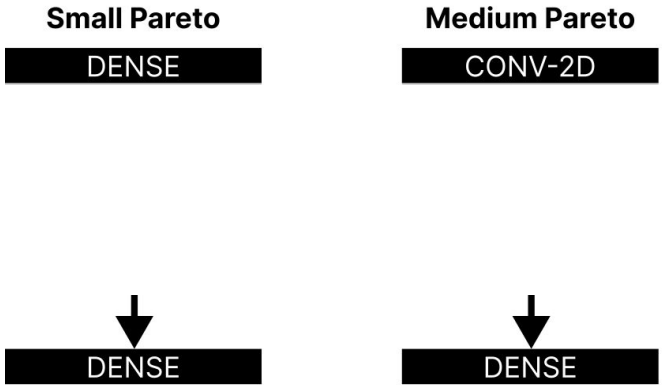
# ECON-T Autoencoder

- We study 3 Pareto-optimal models



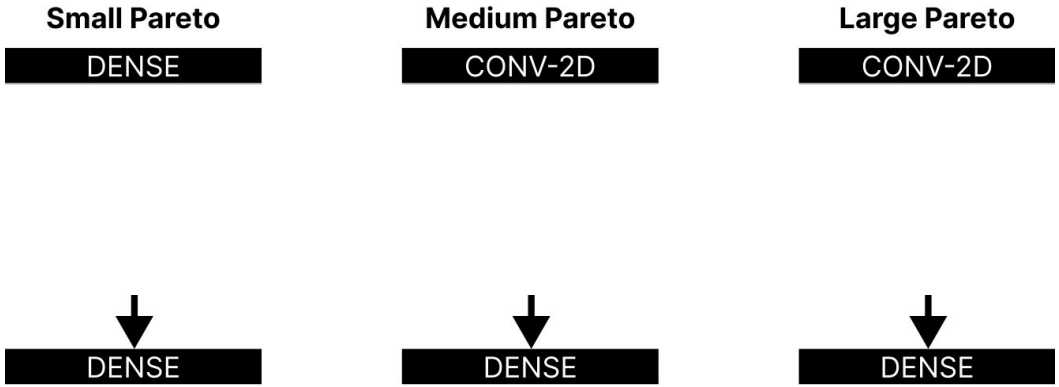
# ECON-T Autoencoder

- We study 3 Pareto-optimal models



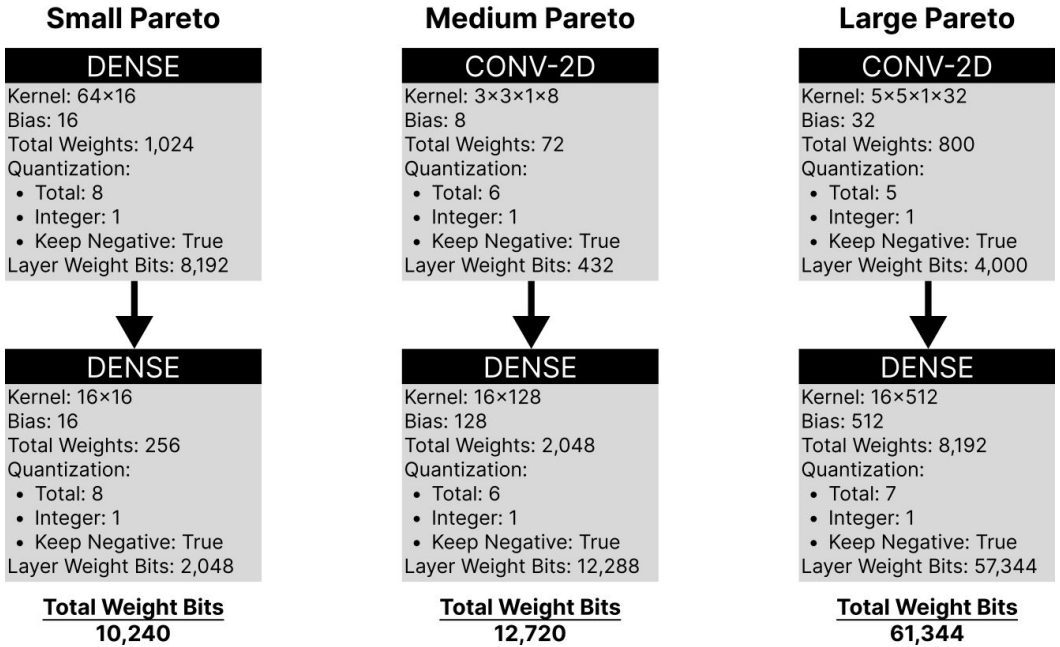
# ECON-T Autoencoder

- We study 3 Pareto-optimal models



# ECON-T Autoencoder

- We study 3 Pareto-optimal models

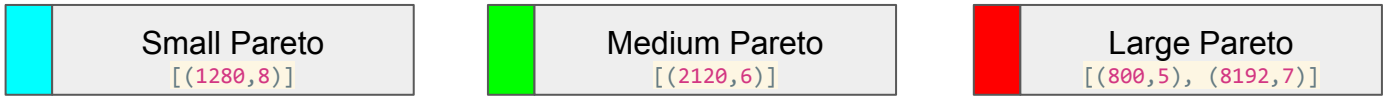


# ECON-T Fault Injection Campaign

- Experiment: Flip a bit and measure Earth Mover's Distance (EMD)
  - **Bit to protect** = EMD_flipped > Original EMD

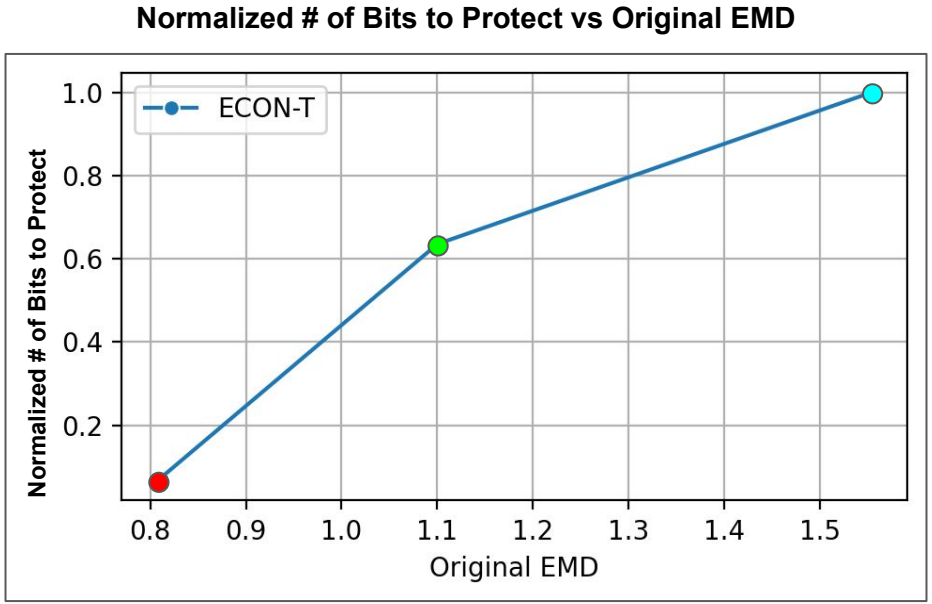
# ECON-T Fault Injection Campaign

- Experiment: Flip a bit and measure Earth Mover's Distance (EMD)
  - **Bit to protect** = EMD_flipped > Original EMD



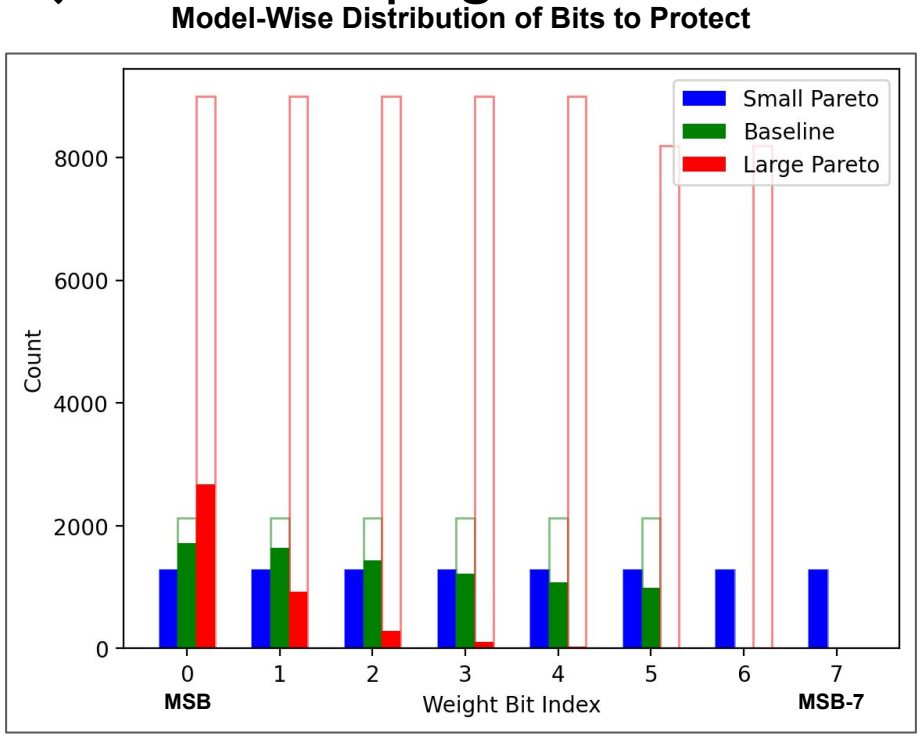
# ECON-T Fault Injection Campaign

- Experiment: Flip a bit and measure Earth Mover's Distance (EMD)
  - **Bit to protect** = EMD_flipped > Original EMD




| Small Pareto | Medium Pareto | Large Pareto |
|---|---|---|
| [(1280,8)] | [(2120,6)] | [(800,5), (8192,7)] |

# ECON-T Fault Injection Campaign




| Small Pareto | Medium Pareto | Large Pareto |
|---|---|---|
| `[(1280,8)]` | `[(2120,6)]` | `[(800,5), (8192,7)]` |

# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models

- Current features:
  - **Fault injection**
  - Bit-level sensitivity metrics

Fault injection campaigns are expensive…

Fault injection campaigns are expensive…

Can we **quantify** fault sensitivity a priori?

# 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: EMD_flipped >> Original EMD

# Bit-level Sensitivity Metrics

- We provide **bit-level sensitivity** metrics for ranking weight bits
  - **High** sensitivity: EMD_flipped >> Original EMD
  - **Low** sensitivity: EMD_flipped ≤ Original EMD

# Bit-level Sensitivity Metrics

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

# Bit-level Sensitivity Metrics

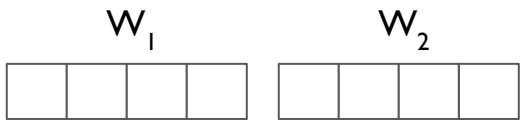
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - **Random**

Example: Two 4-bit weights

# Bit-level Sensitivity Metrics

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

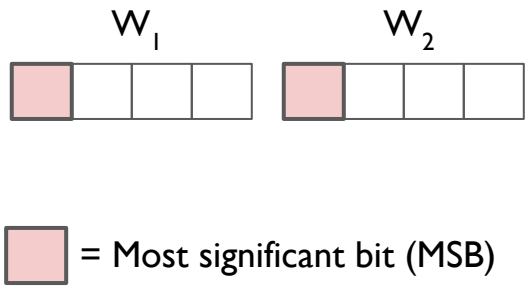
Example: Two 4-bit weights



# Bit-level Sensitivity Metrics

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

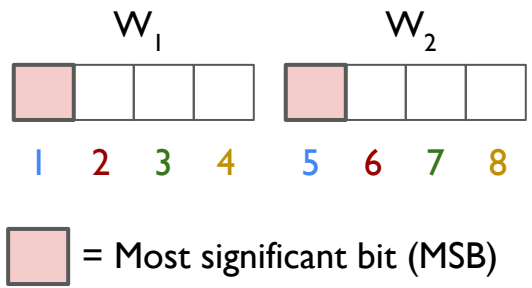
Example: Two 4-bit weights



# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights

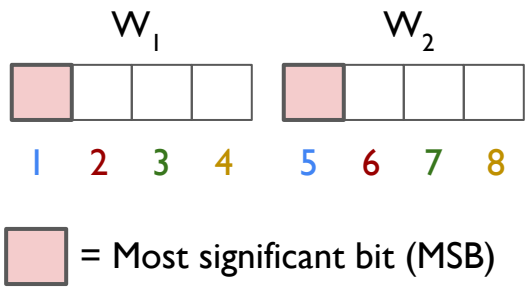


# Bit-level Sensitivity Metrics

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

Ranking:

Example: Two 4-bit weights



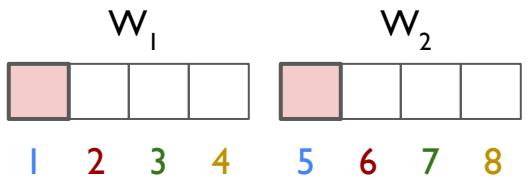
# Bit-level Sensitivity Metrics

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

Ranking:

← High sensitivity

Example: Two 4-bit weights



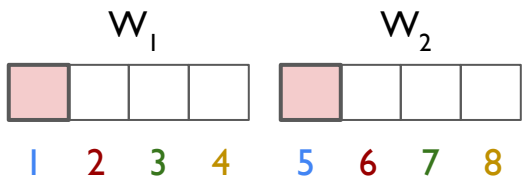

= Most significant bit (MSB)

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

3 ← High sensitivity
5
6
1
7
4
8
2 ← Low sensitivity

# Bit-level Sensitivity Metrics

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

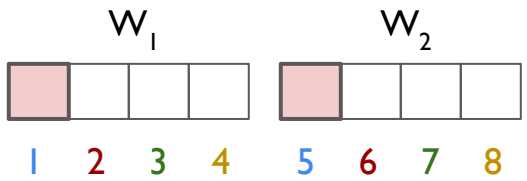
# Bit-level Sensitivity Metrics

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

Ranking:

← High sensitivity

Example: Two 4-bit weights




= Most significant bit (MSB)

← Low sensitivity

# Bit-level Sensitivity Metrics

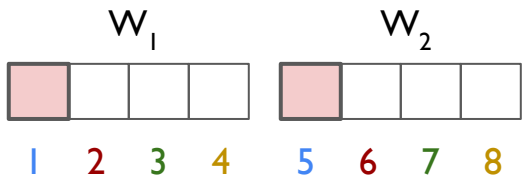
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - **Most significant bit (MSB) → Least significant bit (LSB)**

Ranking:

1 ← High sensitivity

5

Example: Two 4-bit weights



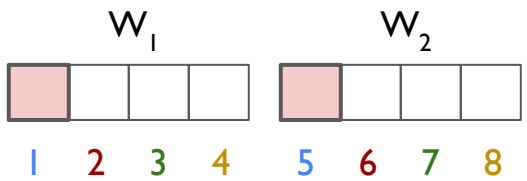

= Most significant bit (MSB)

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

1 ← High sensitivity

5

2

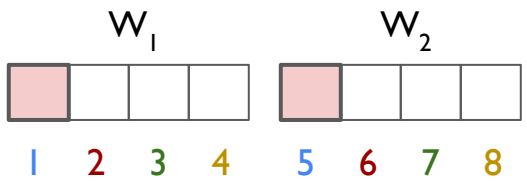
6

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

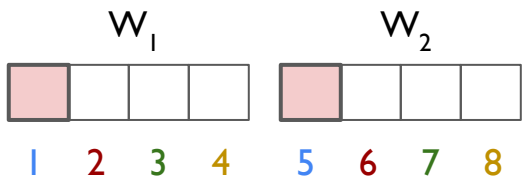
1 ← High sensitivity
5
2
6
3
7

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

1 ← High sensitivity
5
2
6
3
7
4
8 ← Low sensitivity

# Bit-level Sensitivity Metrics

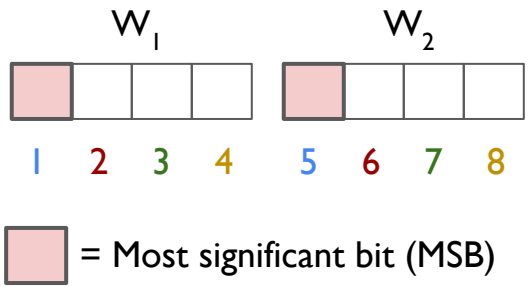
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - **Gradient**

# Bit-level Sensitivity Metrics

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

Ranking:

Example: Two 4-bit weights

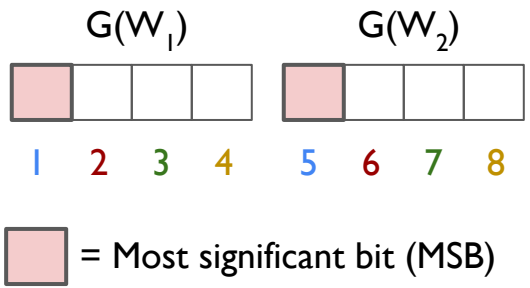


# Bit-level Sensitivity Metrics

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

Ranking:

Example: Two 4-bit weights

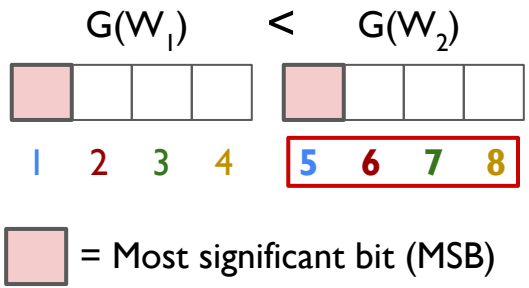


# Bit-level Sensitivity Metrics

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

Ranking:

Example: Two 4-bit weights



# Bit-level Sensitivity Metrics

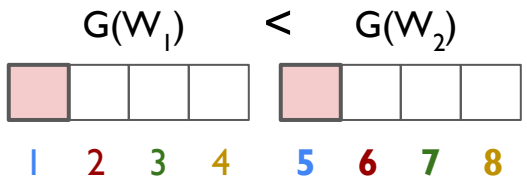
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - **Gradient**

Ranking:

5 ← High sensitivity

1

Example: Two 4-bit weights



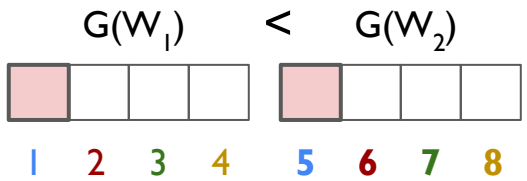

= Most significant bit (MSB)

← Low sensitivity

# Bit-level Sensitivity Metrics

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

**Example:** Two 4-bit weights




= Most significant bit (MSB)

**Ranking:**

5 ← High sensitivity

1

6

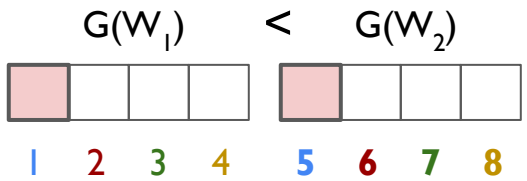
2

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

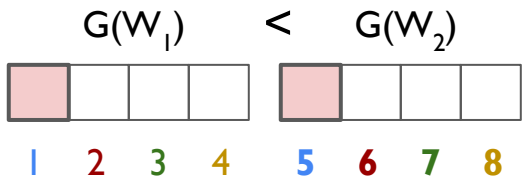
5 ← High sensitivity
1
6
2
7
3

← Low sensitivity

# Bit-level Sensitivity Metrics

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

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

5 ← High sensitivity
1
6
2
7
3
8
4 ← Low sensitivity

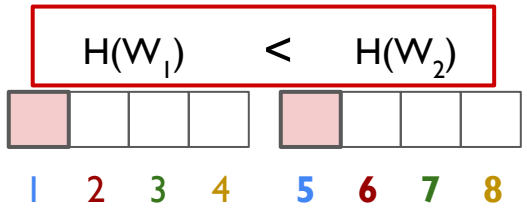
# Bit-level Sensitivity Metrics

- 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**

Example: Two 4-bit weights




= Most significant bit (MSB)

Ranking:

5 ← High sensitivity
1
6
2
7
3
8
4 ← Low sensitivity

# Bit-level Sensitivity Metrics

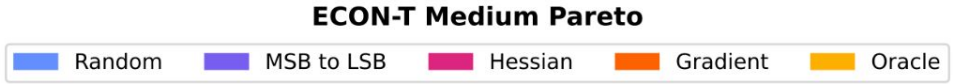
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - Gradient
  - Hessian

# Bit-level Sensitivity Metrics

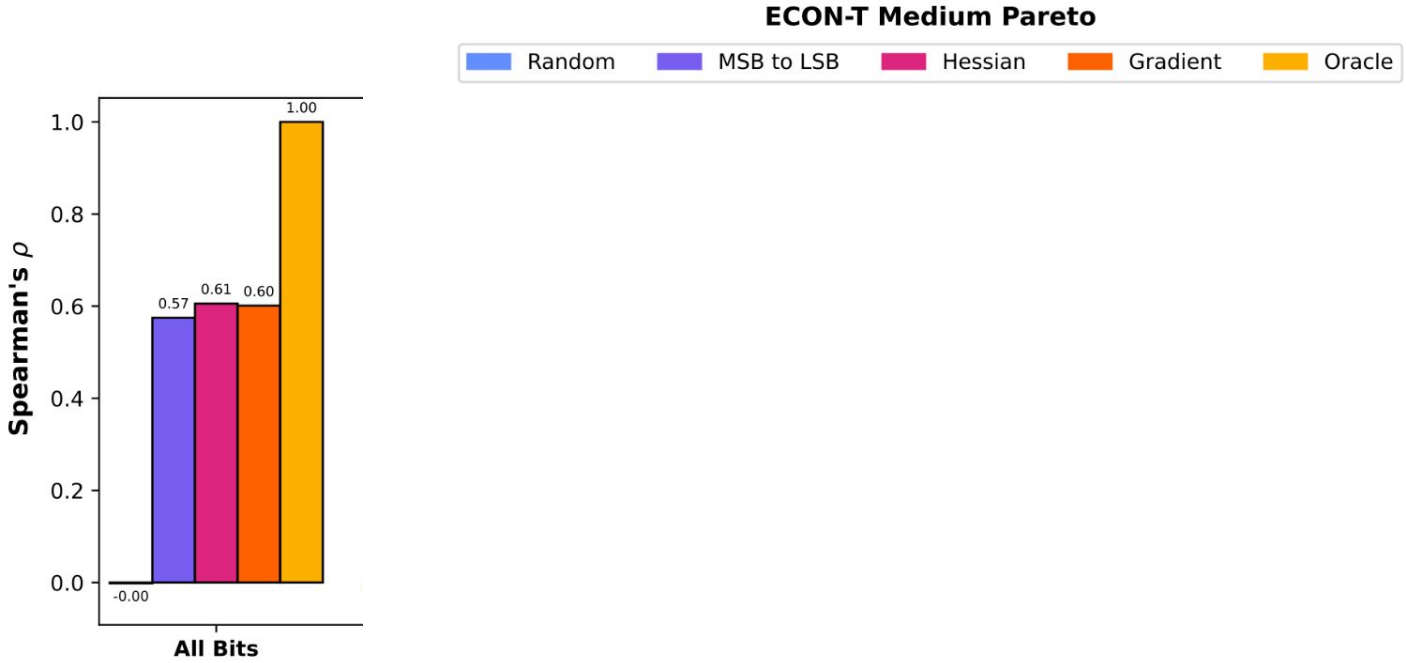
- We provide **bit-level sensitivity** metrics for ranking weight bits
  - Random
  - Most significant bit (MSB) → Least significant bit (LSB)
  - Gradient
  - Hessian

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

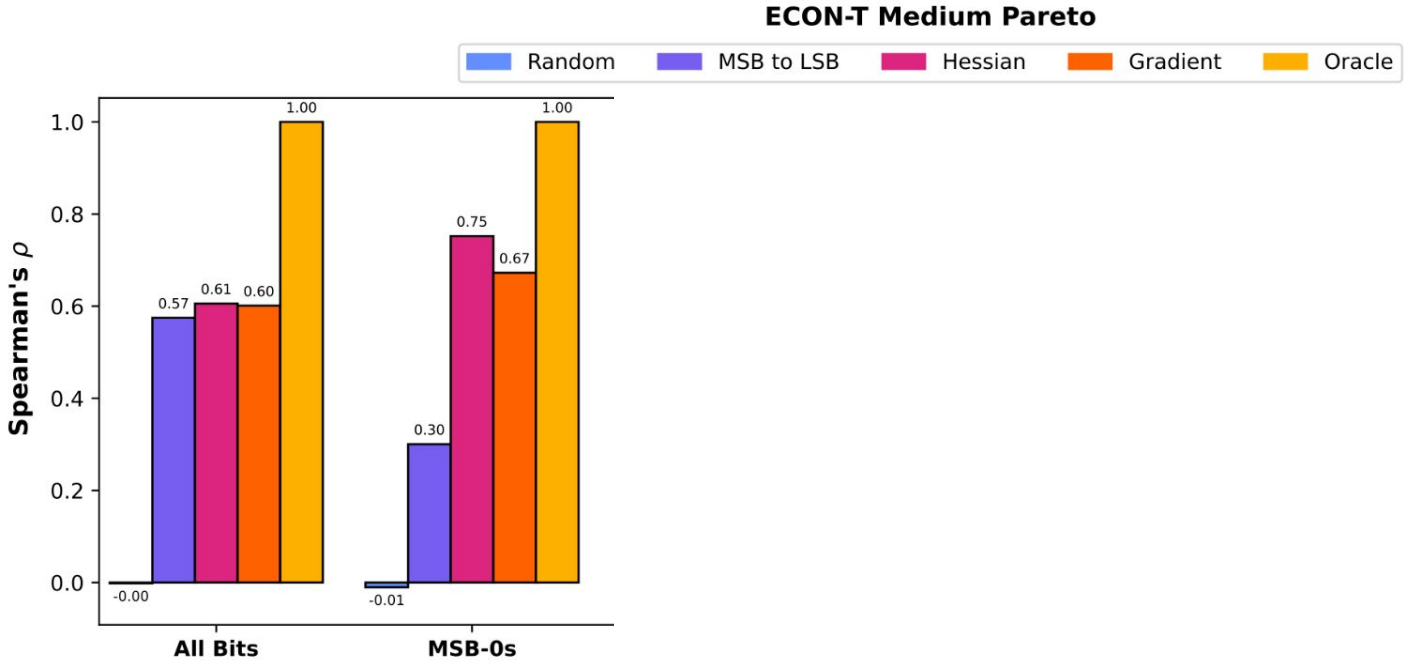
# How do our metrics compare with a perfect ranking?



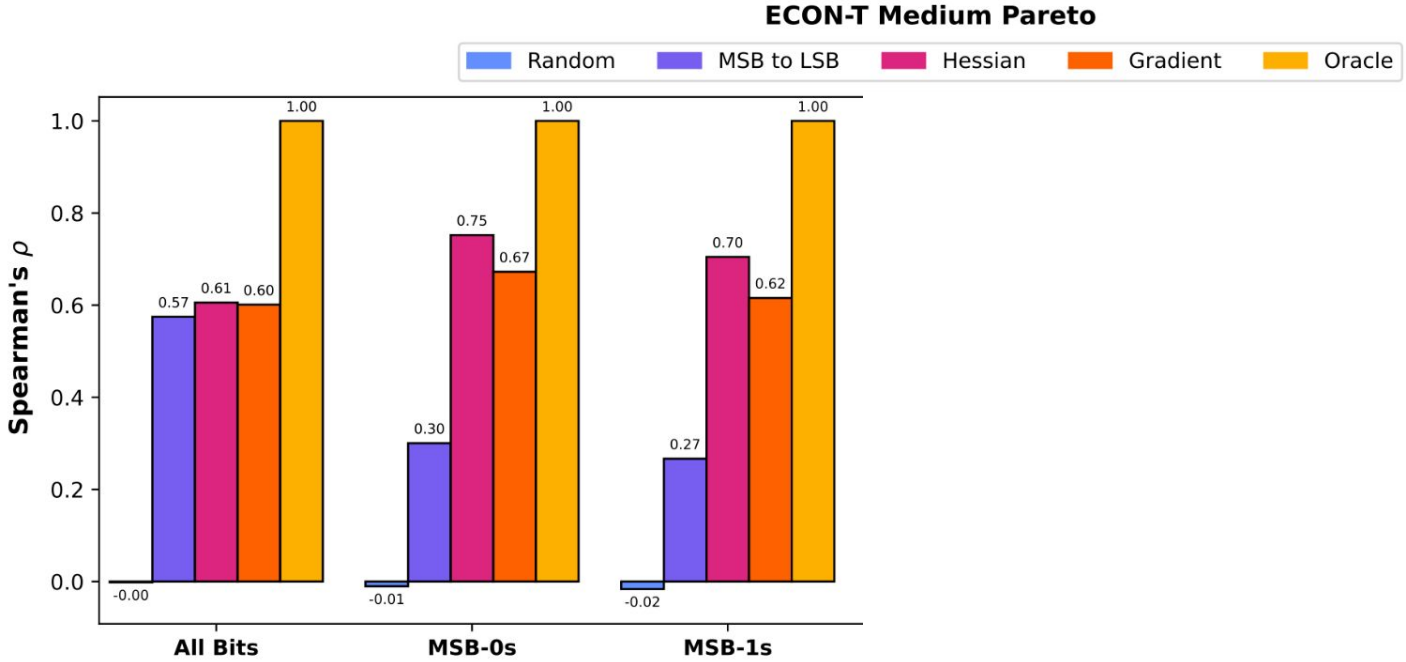
# How do our metrics compare with a perfect ranking?



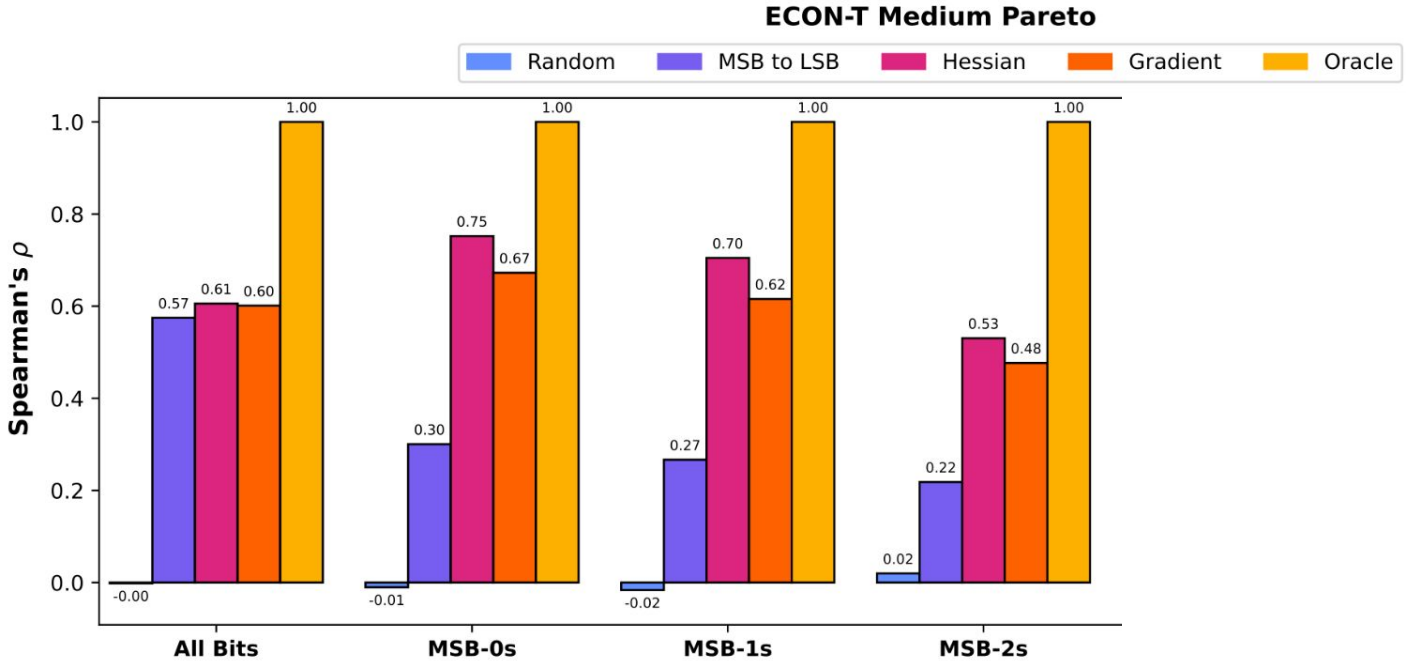
# How do our metrics compare with a perfect ranking?



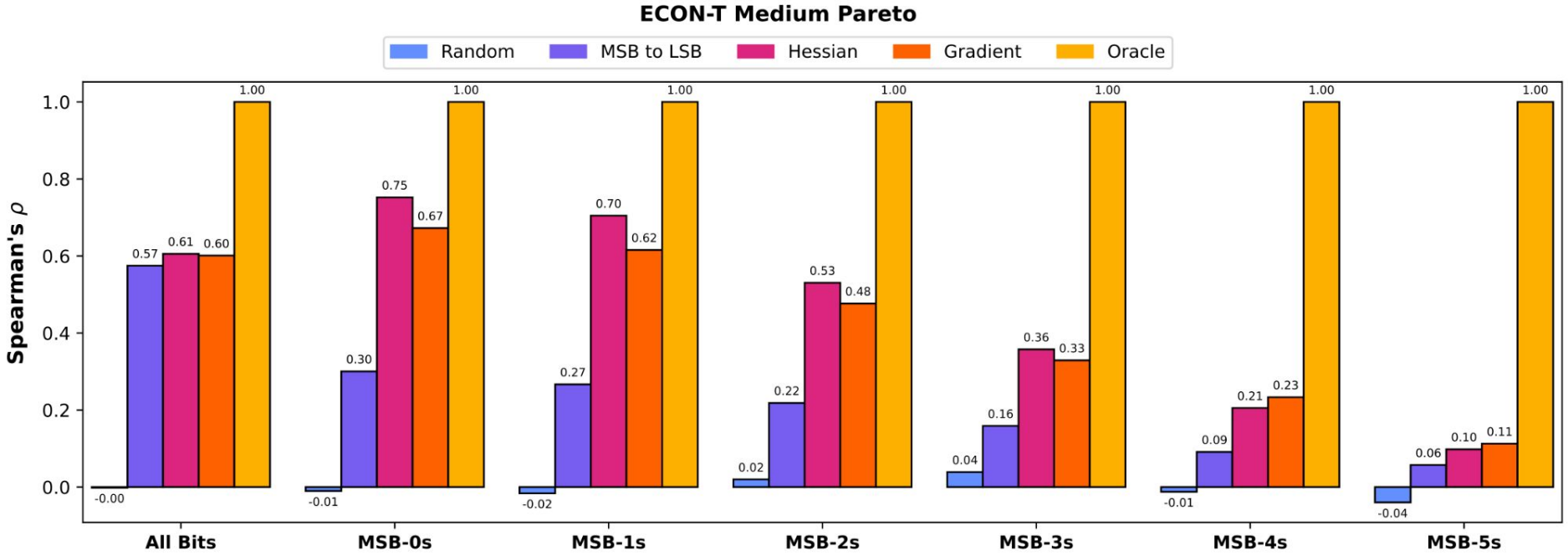
# How do our metrics compare with a perfect ranking?



# How do our metrics compare with a perfect ranking?



# How do our metrics compare with a perfect ranking?



# FKeras

- A library that **assesses the fault sensitivity** of (Q)Keras models

- Current features:
  - Fault injection
  - **Bit-level sensitivity metrics**

# Future Work

We want to use FKeras to:

# Future Work

We want to use FKeras to:

- Analyze more edge NNs and datasets

# Future Work

We want to use FKeras to:

- Analyze more edge NNs and datasets
  - How much can our metrics speed up fault injection campaigns?

# Future Work

We want to use FKeras to:

- Analyze more edge NNs and datasets
  - How much can our metrics speed up fault injection campaigns?

- Perform NN design space exploration that considers fault sensitivity using our metrics

# Future Work

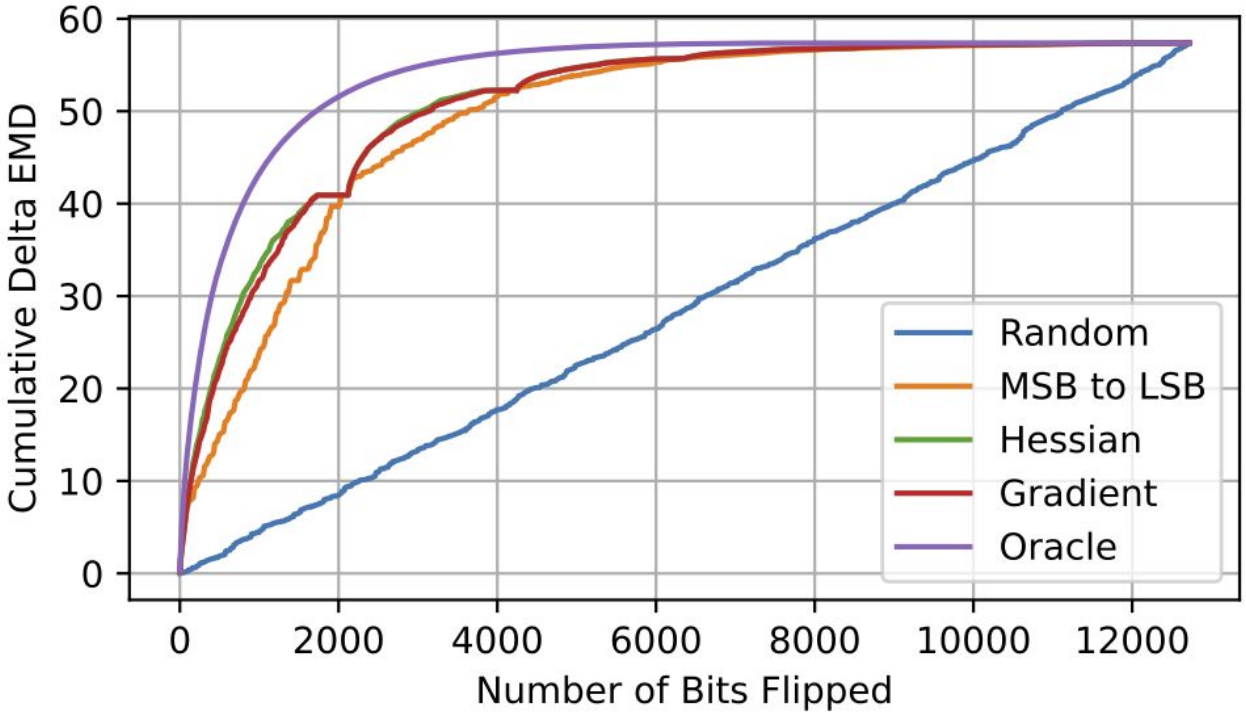
We want to use FKeras to:

- Analyze more edge NNs and datasets
  - How much can our metrics speed up fault injection campaigns?

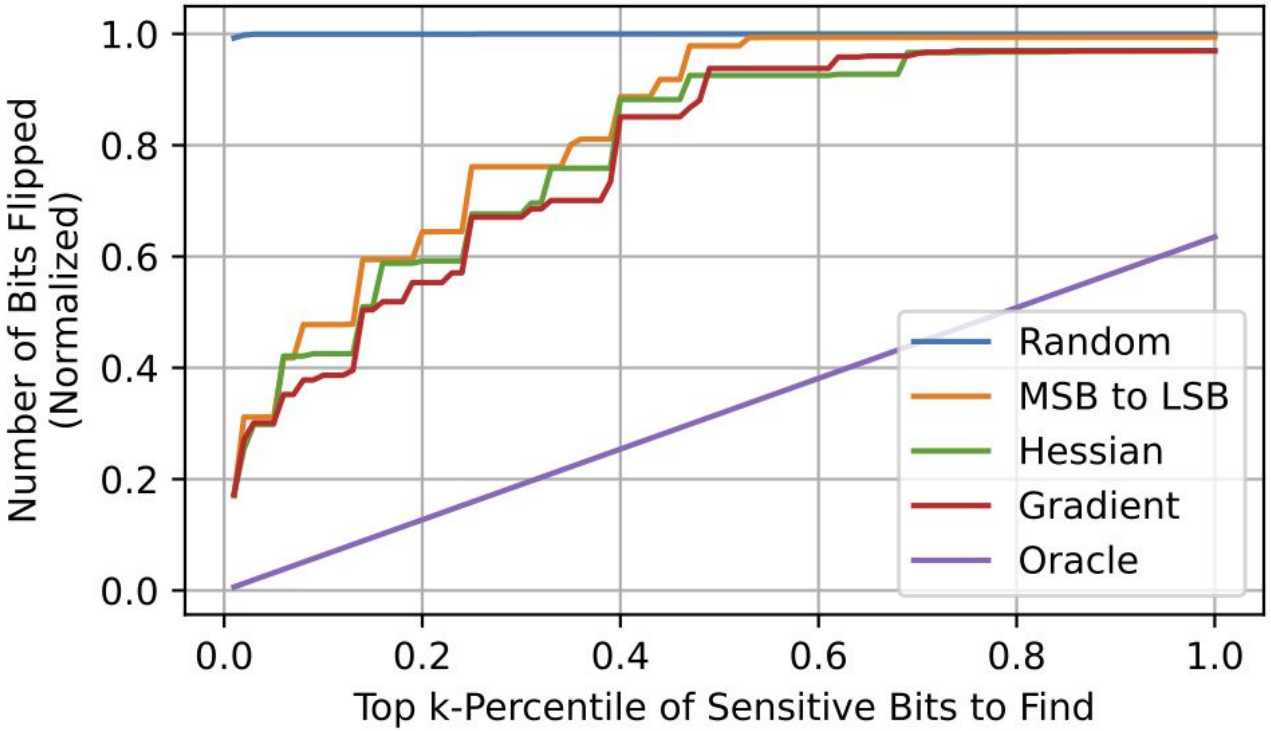
- Perform NN design space exploration that considers fault sensitivity using our metrics
  - How does fault sensitivity interact with performance, area, etc?

# Backup

# How do our metrics compare with a perfect ranking?



# How do our metrics compare with a perfect ranking?



# How to use bit-level sensitivity rankings?

# How to use bit-level sensitivity rankings?

- Speed up fault injection campaigns

# How to use bit-level sensitivity rankings?

- Speed up fault injection campaigns

- Design space exploration