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

Presented on November 2nd, 2023 at Fast ML for Science @ ICCAD 2023

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**

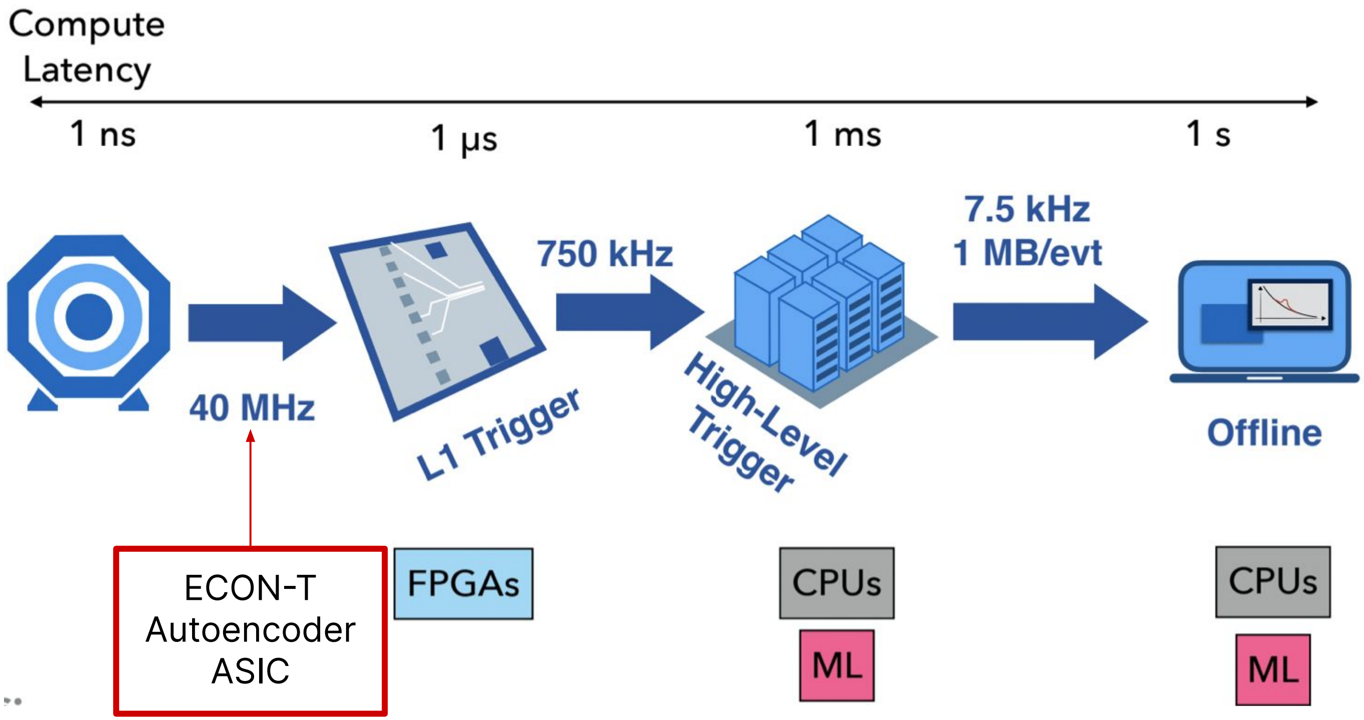
Why analyze the sensitivity of edge NNs?

- Edge NNs can be sensitive creatures

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

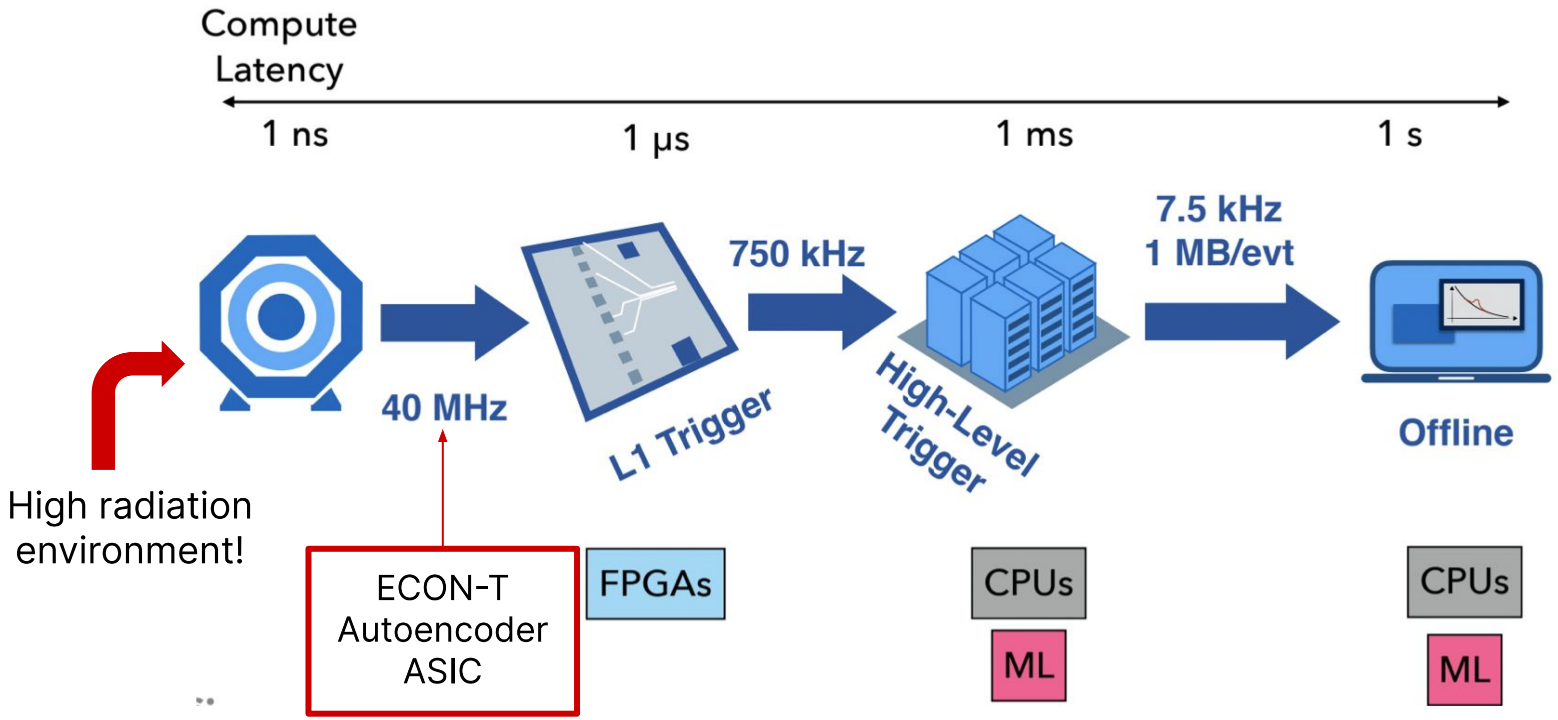
When do **hardware faults** occur?

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?

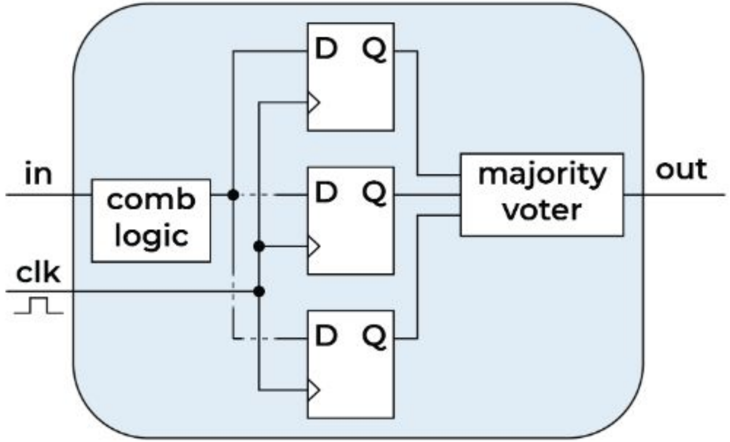
ECON-T Radiation Tolerance

- 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

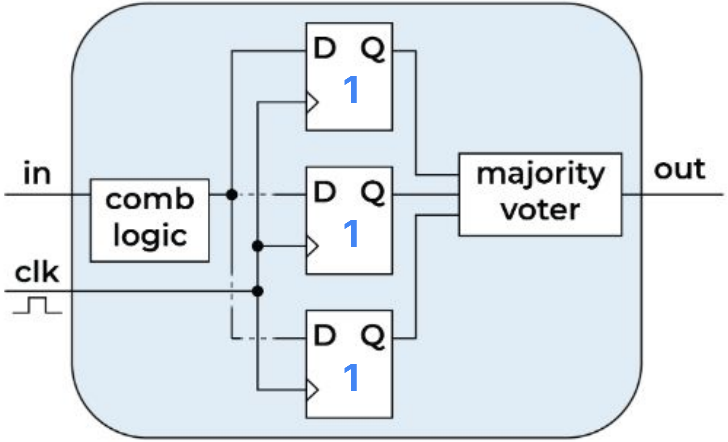
- ECON-T employs **triple modular redundancy (TMR)** to its registers



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

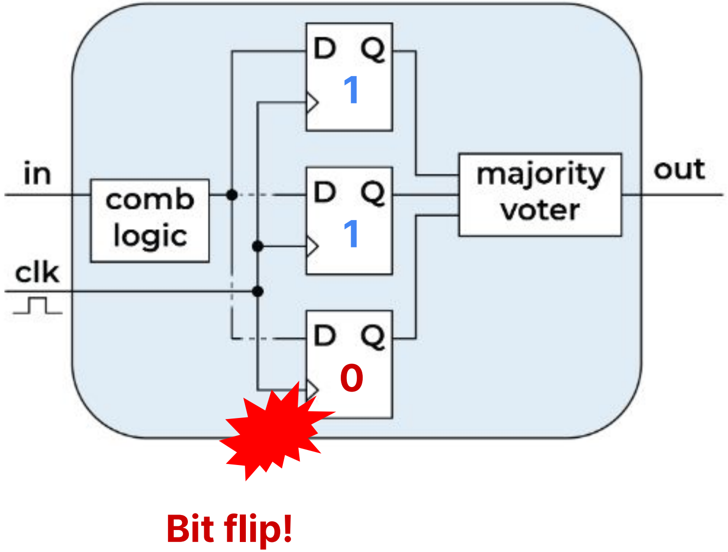
- ECON-T employs **triple modular redundancy (TMR)** to its registers



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

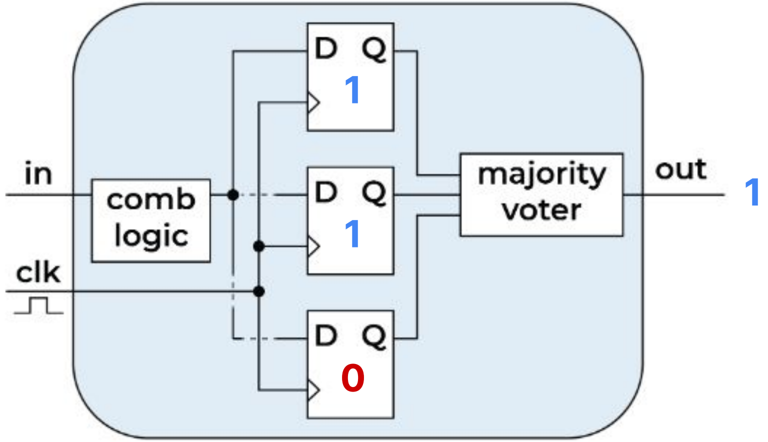
- ECON-T employs **triple modular redundancy (TMR)** to its registers



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

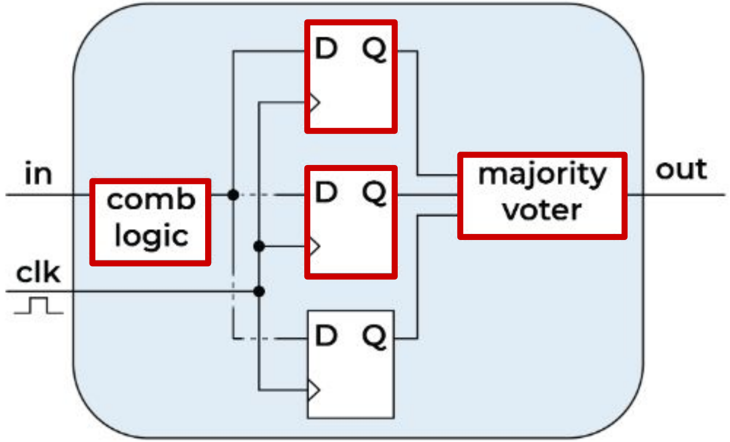
- ECON-T employs **triple modular redundancy (TMR)** to its registers



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

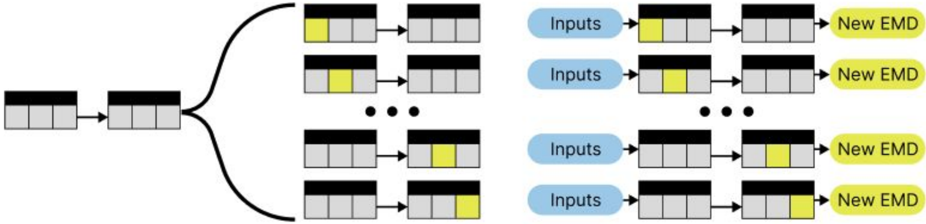
- A library that assesses the fault sensitivity of (Q)Keras models

- Current features include:
  - Bit-level fault injection with fine-grained control
  - Bit-level sensitivity metrics for ranking weight bits

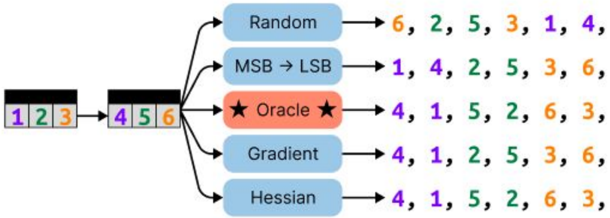
FKeras

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

- Current features include:
  - Bit-level fault injection with fine-grained control
  - Bit-level sensitivity metrics for ranking weight bits



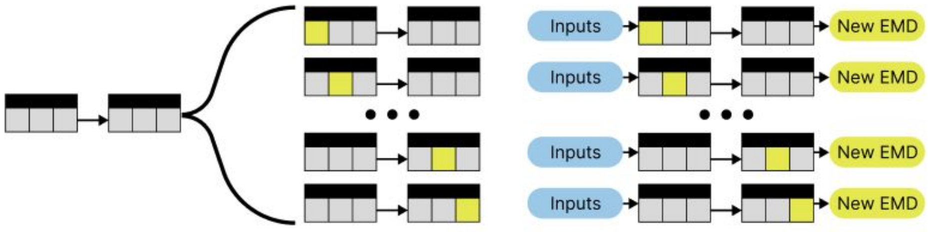
Bit-level fault injection with fine-grained control



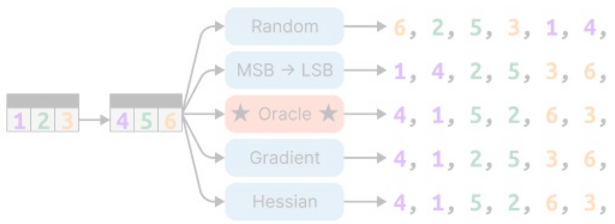
Bit-level sensitivity metrics for ranking weight bits (without fault injection)

FKeras

- A library that assesses the fault sensitivity of (Q)Keras models
- Current features include:
  - Bit-level fault injection with fine-grained control
  - Bit-level sensitivity metrics for ranking weight bits



Bit-level fault injection with fine-grained control



Bit-level sensitivity metrics for ranking weight bits (without fault injection)

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE		DENSE
Kernel: 64×16 Bias: 16 Total Weights: 1,024 Quantization: • Total: 8 • Integer: 1 • Keep Negative: True Layer Weight Bits: 8,192	→	Kernel: 16×16 Bias: 16 Total Weights: 256 Quantization: • Total: 8 • Integer: 1 • Keep Negative: True Layer Weight Bits: 2,048

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE
Kernel: 64×16
Bias: 16
Total Weights: 1,024
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 8,192

→

DENSE
Kernel: 16×16
Bias: 16
Total Weights: 256
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D
Kernel: 3×3×1×8
Bias: 8
Total Weights: 72
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 432

→

DENSE
Kernel: 16×128
Bias: 128
Total Weights: 2,048
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 12,288

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE	→	DENSE
Kernel: 64×16 Bias: 16 Total Weights: 1,024 Quantization: • Total: 8 • Integer: 1 • Keep Negative: True Layer Weight Bits: 8,192	→	Kernel: 16×16 Bias: 16 Total Weights: 256 Quantization: • Total: 8 • Integer: 1 • Keep Negative: True Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D	→	DENSE
Kernel: 3×3×1×8 Bias: 8 Total Weights: 72 Quantization: • Total: 6 • Integer: 1 • Keep Negative: True Layer Weight Bits: 432	→	Kernel: 16×128 Bias: 128 Total Weights: 2,048 Quantization: • Total: 6 • Integer: 1 • Keep Negative: True Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

CONV-2D	→	DENSE
Kernel: 5×5×1×32 Bias: 32 Total Weights: 800 Quantization: • Total: 5 • Integer: 1 • Keep Negative: True Layer Weight Bits: 4,000	→	Kernel: 16×512 Bias: 512 Total Weights: 8,192 Quantization: • Total: 7 • Integer: 1 • Keep Negative: True Layer Weight Bits: 57,344

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE
Kernel: 64×16
Bias: 16
Total Weights: 1,024
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 8,192

→

DENSE
Kernel: 16×16
Bias: 16
Total Weights: 256
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D
Kernel: 3×3×1×8
Bias: 8
Total Weights: 72
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 432

→

DENSE
Kernel: 16×128
Bias: 128
Total Weights: 2,048
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

CONV-2D
Kernel: 5×5×1×32
Bias: 32
Total Weights: 800
Quantization:
- Total: 5
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 4,000

→

DENSE
Kernel: 16×512
Bias: 512
Total Weights: 8,192
Quantization:
- Total: 7
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 57,344

- Conceptually, each fault injection campaign:

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE
Kernel: 64×16
Bias: 16
Total Weights: 1,024
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 8,192

→

DENSE
Kernel: 16×16
Bias: 16
Total Weights: 256
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D
Kernel: 3×3×1×8
Bias: 8
Total Weights: 72
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 432

→

DENSE
Kernel: 16×128
Bias: 128
Total Weights: 2,048
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

CONV-2D
Kernel: 5×5×1×32
Bias: 32
Total Weights: 800
Quantization:
- Total: 5
- Integer: 1
- Keep Negative: True

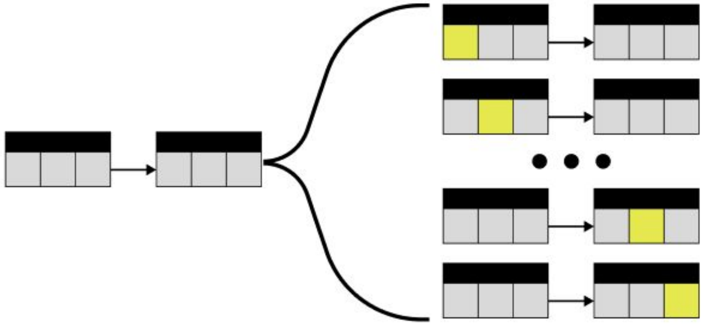
Layer Weight Bits: 4,000

→

DENSE
Kernel: 16×512
Bias: 512
Total Weights: 8,192
Quantization:
- Total: 7
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 57,344

- Conceptually, each fault injection campaign:



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

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE	DENSE
Kernel: 64×16	Kernel: 16×16
Bias: 16	Bias: 16
Total Weights: 1,024	Total Weights: 256
Quantization:	Quantization:
• Total: 8	• Total: 8
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 8,192	Layer Weight Bits: 2,048

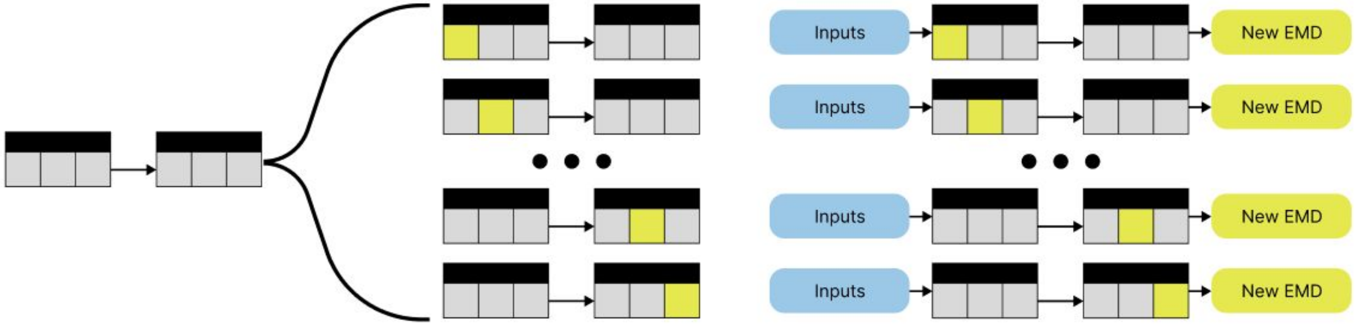
Medium Pareto (Total Weight Bits: 12,720)

CONV-2D	DENSE
Kernel: 3×3×1×8	Kernel: 16×128
Bias: 8	Bias: 128
Total Weights: 72	Total Weights: 2,048
Quantization:	Quantization:
• Total: 6	• Total: 6
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 432	Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

CONV-2D	DENSE
Kernel: 5×5×1×32	Kernel: 16×512
Bias: 32	Bias: 512
Total Weights: 800	Total Weights: 8,192
Quantization:	Quantization:
• Total: 5	• Total: 7
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 4,000	Layer Weight Bits: 57,344

- Conceptually, each fault injection campaign:



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

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

ECON-T: Fault Injection Campaign (Setup)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE	DENSE
Kernel: 64×16	Kernel: 16×16
Bias: 16	Bias: 16
Total Weights: 1,024	Total Weights: 256
Quantization:	Quantization:
• Total: 8	• Total: 8
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 8,192	Layer Weight Bits: 2,048

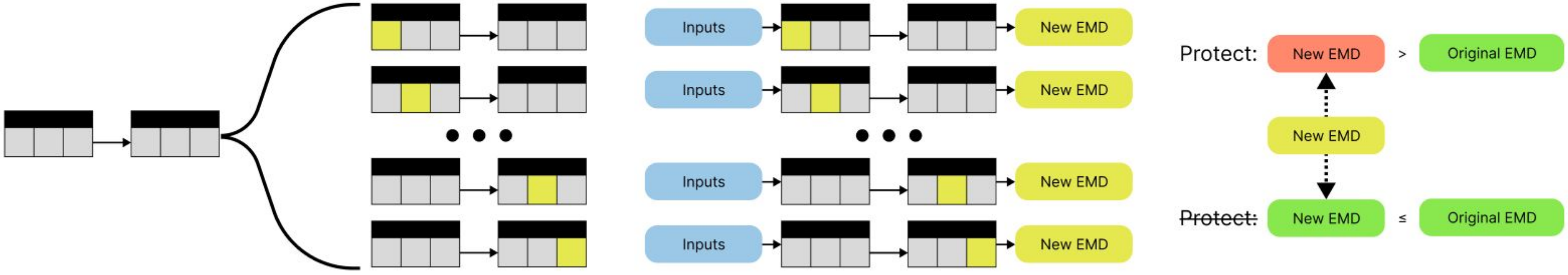
Medium Pareto (Total Weight Bits: 12,720)

CONV-2D	DENSE
Kernel: 3×3×1×8	Kernel: 16×128
Bias: 8	Bias: 128
Total Weights: 72	Total Weights: 2,048
Quantization:	Quantization:
• Total: 6	• Total: 6
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 432	Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

CONV-2D	DENSE
Kernel: 5×5×1×32	Kernel: 16×512
Bias: 32	Bias: 512
Total Weights: 800	Total Weights: 8,192
Quantization:	Quantization:
• Total: 5	• Total: 7
• Integer: 1	• Integer: 1
• Keep Negative: True	• Keep Negative: True
Layer Weight Bits: 4,000	Layer Weight Bits: 57,344

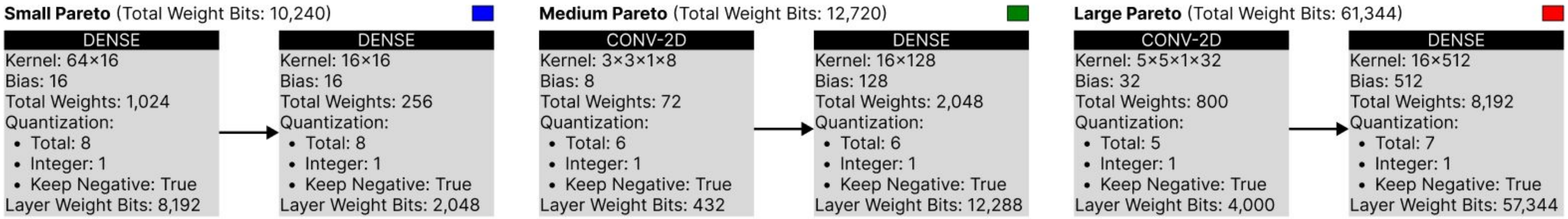
- Conceptually, each fault injection campaign:



1. Generates X "faulty" variants by flipping a single weight bit
2. Measures the new EMD on a set of test inputs
3. Determines the weight bits to protect

ECON-T: Fault Injection Campaign (Results)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models



ECON-T: Fault Injection Campaign (Results)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE
Kernel: 64×16
Bias: 16
Total Weights: 1,024
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 8,192

→

DENSE
Kernel: 16×16
Bias: 16
Total Weights: 256
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D
Kernel: 3×3×1×8
Bias: 8
Total Weights: 72
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 432

→

DENSE
Kernel: 16×128
Bias: 128
Total Weights: 2,048
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

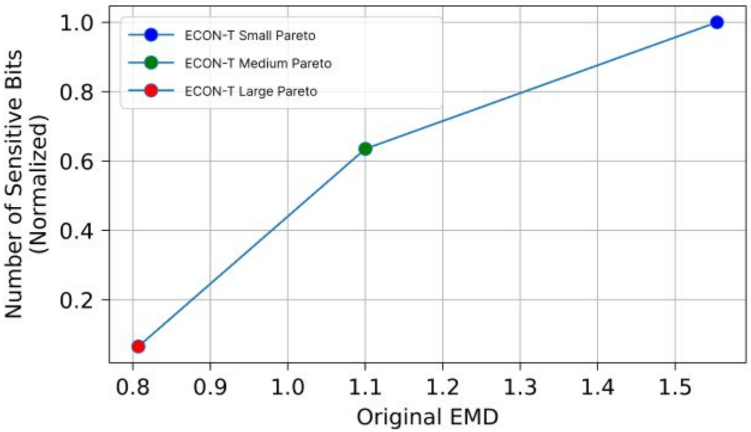
CONV-2D
Kernel: 5×5×1×32
Bias: 32
Total Weights: 800
Quantization:
- Total: 5
- Integer: 1
- Keep Negative: True

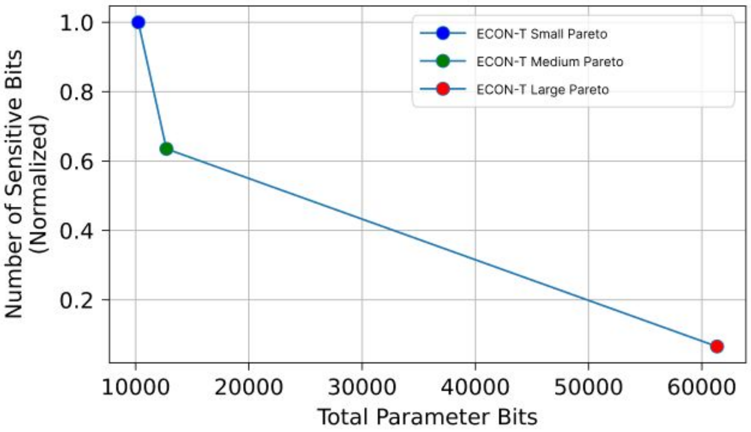
Layer Weight Bits: 4,000

→

DENSE
Kernel: 16×512
Bias: 512
Total Weights: 8,192
Quantization:
- Total: 7
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 57,344





ECON-T: Fault Injection Campaign (Results)

- We perform an exhaustive, bit-level fault injection campaign for 3 Pareto-optimal ECON-T models

Small Pareto (Total Weight Bits: 10,240)

DENSE
Kernel: 64×16
Bias: 16
Total Weights: 1,024
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 8,192

→

DENSE
Kernel: 16×16
Bias: 16
Total Weights: 256
Quantization:
- Total: 8
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 2,048

Medium Pareto (Total Weight Bits: 12,720)

CONV-2D
Kernel: 3×3×1×8
Bias: 8
Total Weights: 72
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 432

→

DENSE
Kernel: 16×128
Bias: 128
Total Weights: 2,048
Quantization:
- Total: 6
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 12,288

Large Pareto (Total Weight Bits: 61,344)

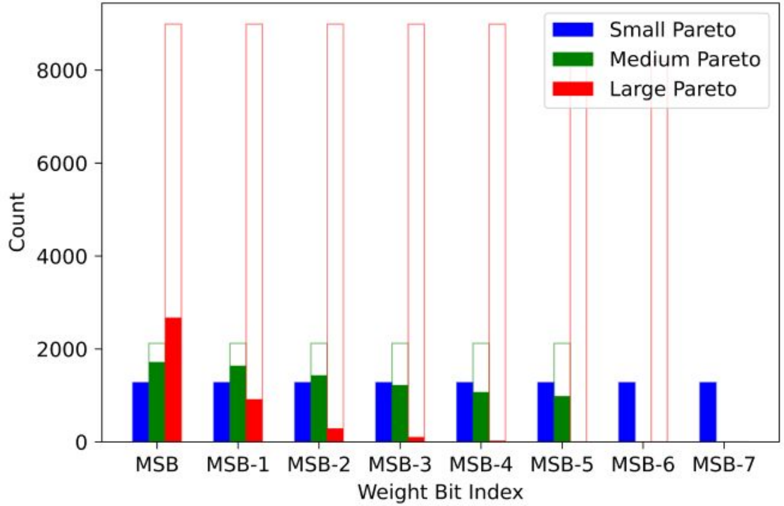
CONV-2D
Kernel: 5×5×1×32
Bias: 32
Total Weights: 800
Quantization:
- Total: 5
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 4,000

→

DENSE
Kernel: 16×512
Bias: 512
Total Weights: 8,192
Quantization:
- Total: 7
- Integer: 1
- Keep Negative: True

Layer Weight Bits: 57,344



Fault injection campaigns are expensive...

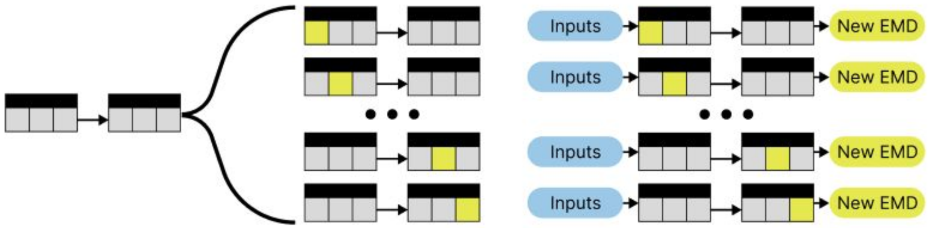
Fault injection campaigns are expensive...

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

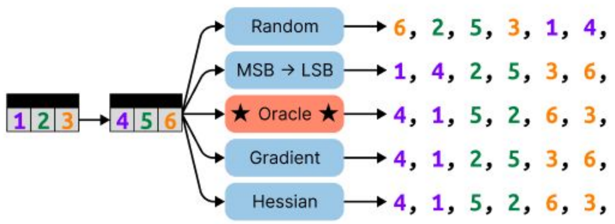
FKeras

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

- Current features include:
  - Bit-level fault injection with fine-grained control
  - Bit-level sensitivity metrics for ranking weight bits



Bit-level fault injection with fine-grained control

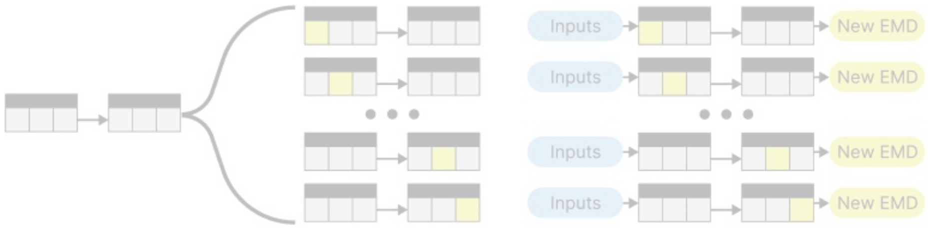


Bit-level sensitivity metrics for ranking weight bits (without fault injection)

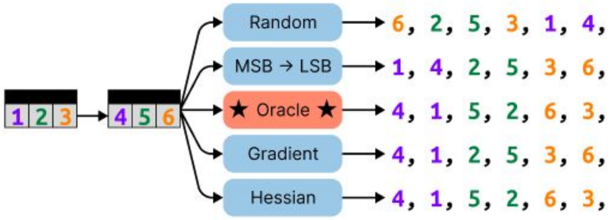
FKeras

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

- Current features include:
  - Bit-level fault injection with fine-grained control
  - Bit-level sensitivity metrics for ranking weight bits



Bit-level fault injection with fine-grained control



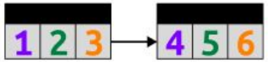
Bit-level sensitivity metrics for ranking weight bits (without fault injection)

Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD

Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- <u>Example:</u> Assume we have a model with two layers which each have one 3-bit weight



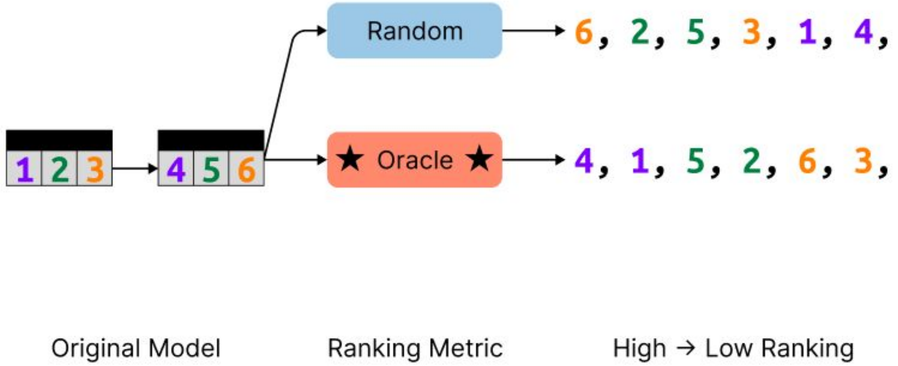
Original Model

Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- Example: Assume we have a model with two layers which each have one 3-bit weight
- Ranking Metric:
  - Oracle (requires fault injection)

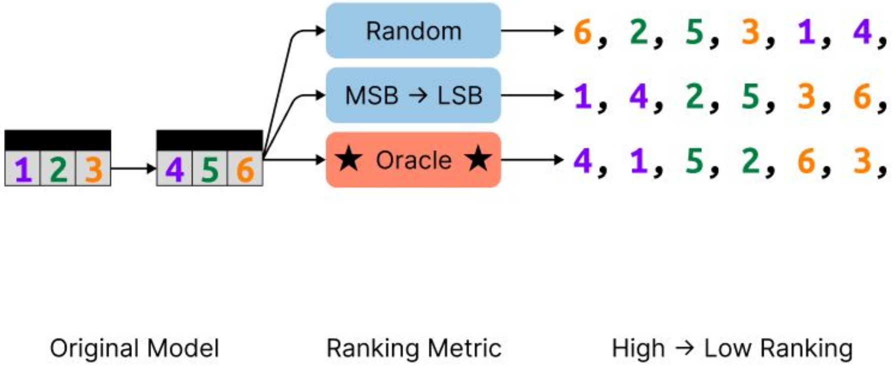
Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- Example: Assume we have a model with two layers which each have one 3-bit weight
- Ranking Metric:
  - Oracle (requires fault injection)
  - Random



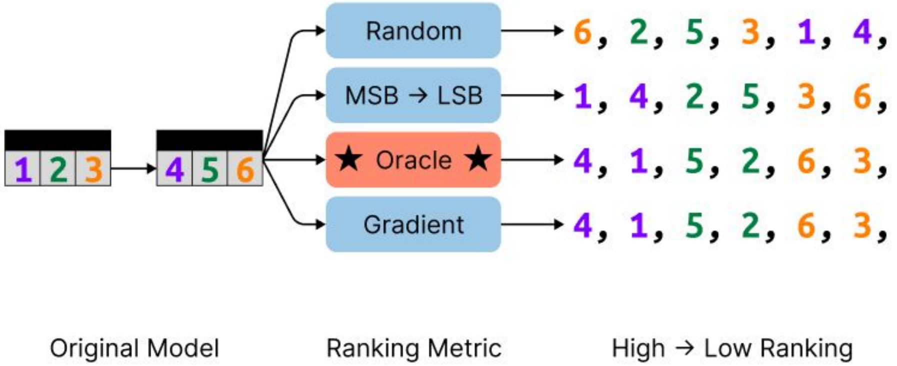
Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- Example: Assume we have a model with two layers which each have one 3-bit weight
- Ranking Metric:
  - Oracle (requires fault injection)
  - Random
  - Most significant bit → least significant bit



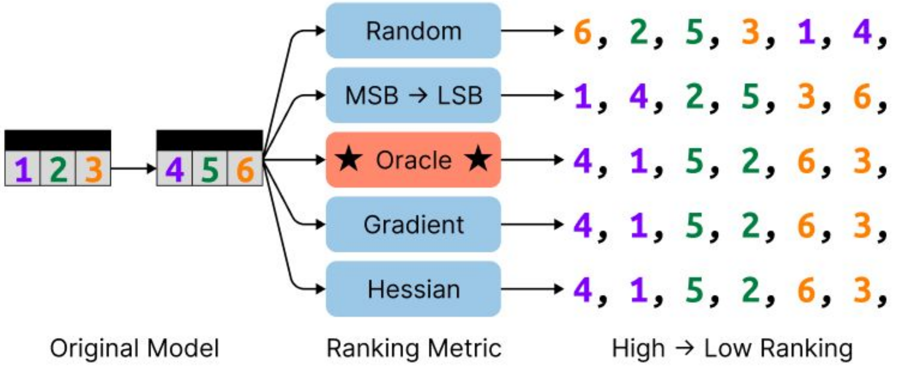
Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- Example: Assume we have a model with two layers which each have one 3-bit weight
- Ranking Metric:
  - Oracle (requires fault injection)
  - Random
  - Most significant bit → least significant bit
  - Gradient (computed at weight level)



Bit-level Sensitivity Metrics

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- Example: Assume we have a model with two layers which each have one 3-bit weight
- Ranking Metric:
  - Oracle (requires fault injection)
  - Random
  - Most significant bit → least significant bit
  - Gradient (computed at weight level)
  - Hessian (computed at weight level)

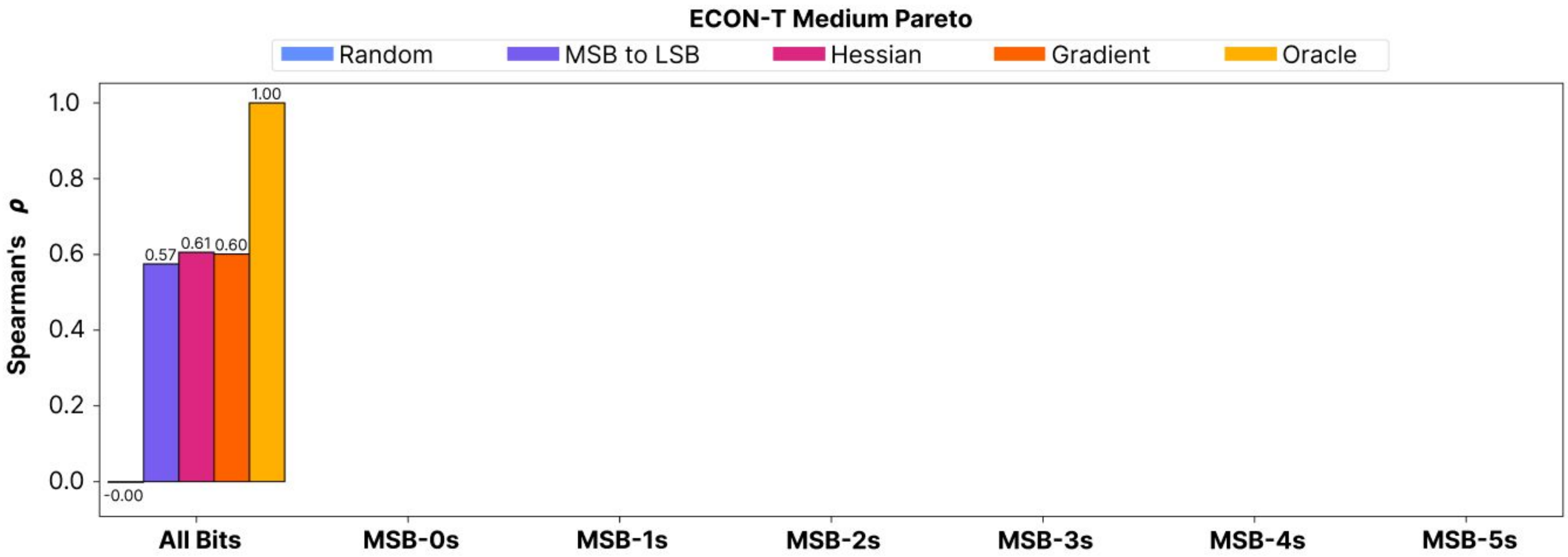


ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?

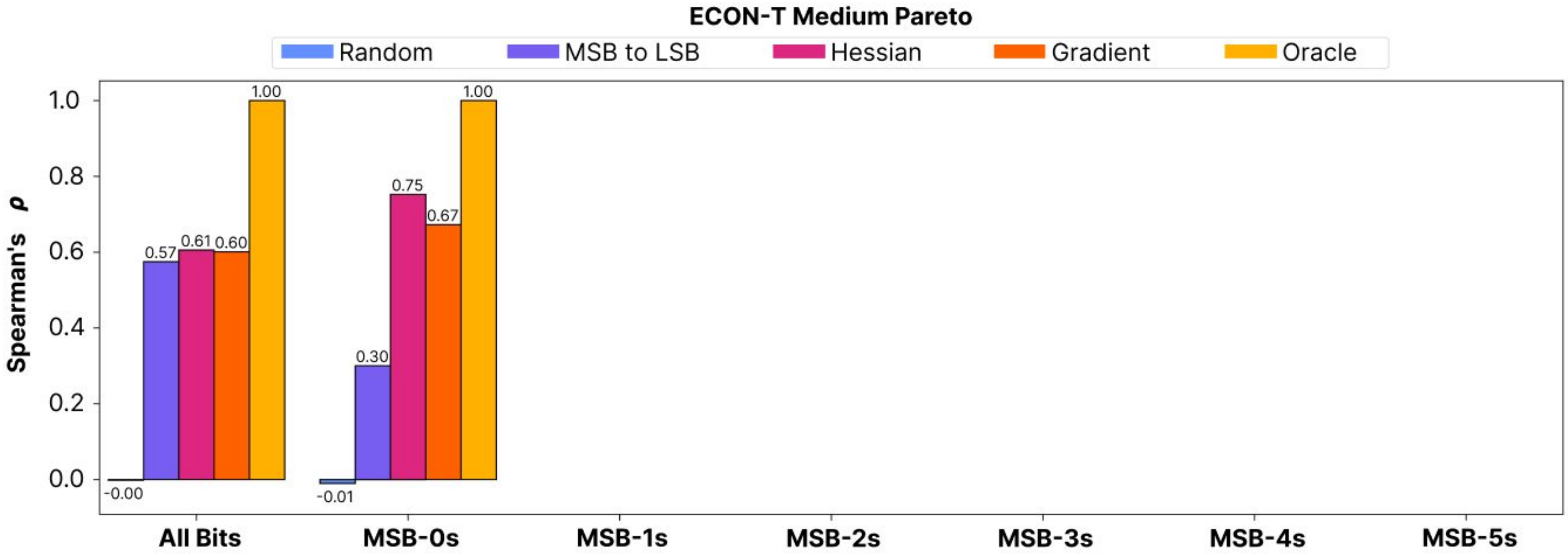
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



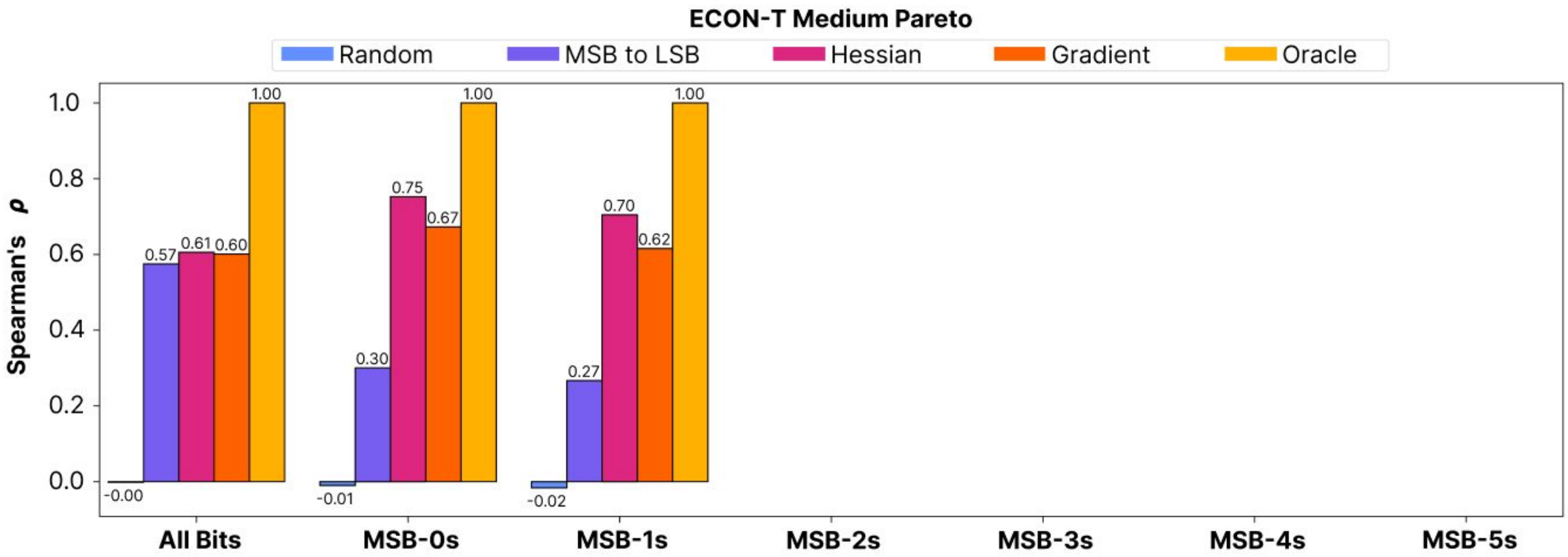
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



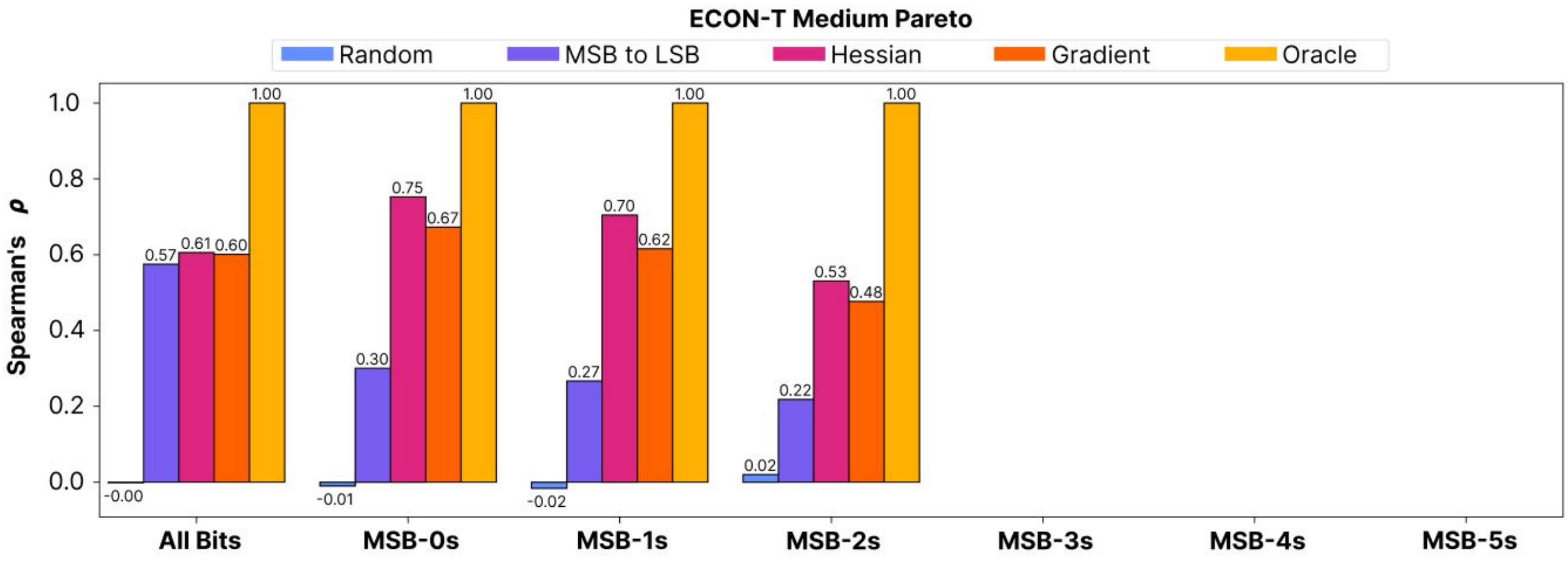
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



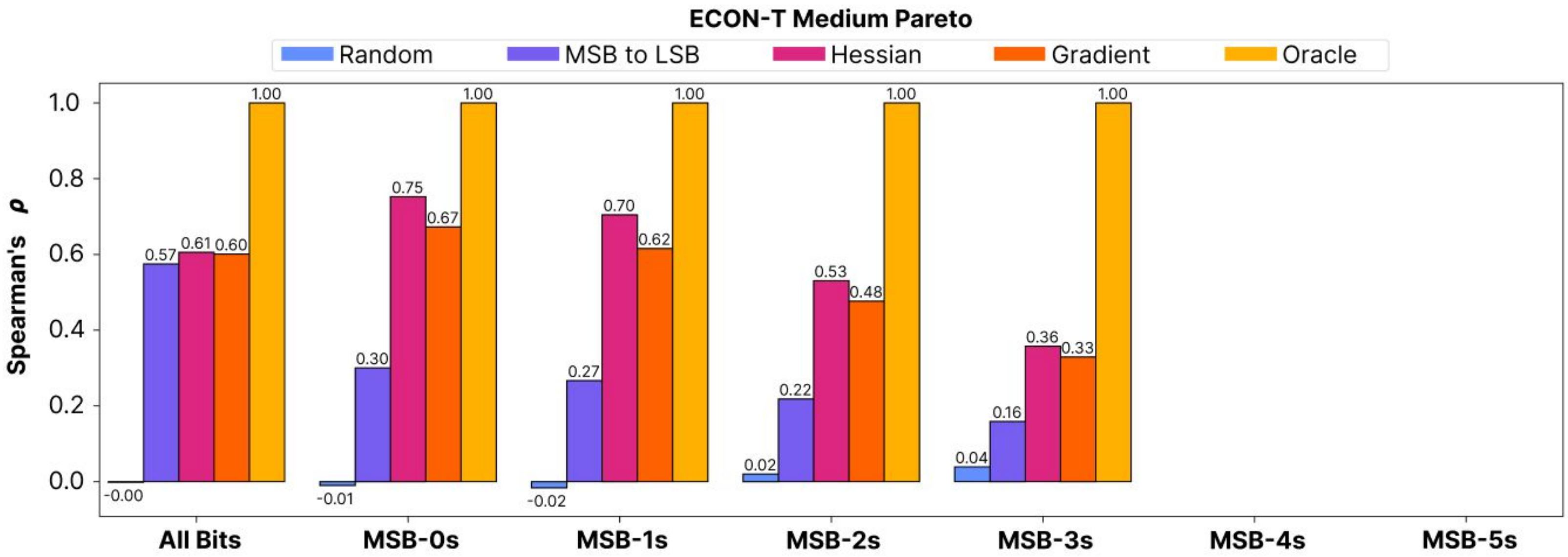
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



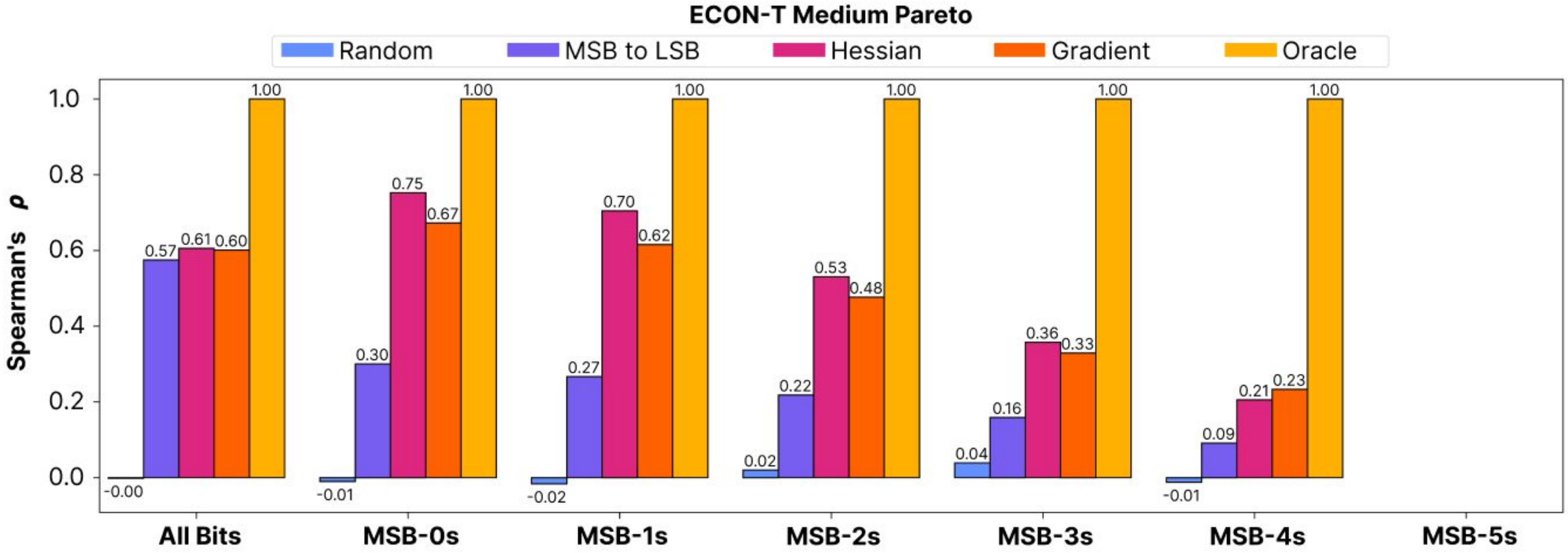
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



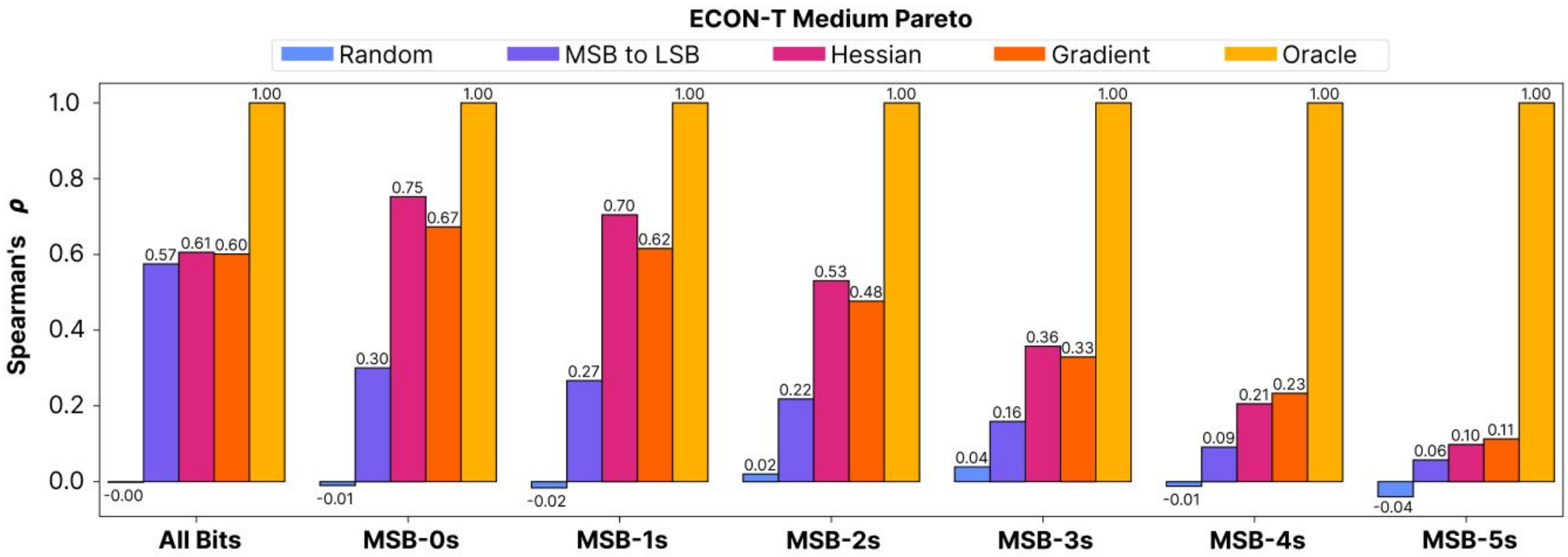
ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



ECON-T: Bit-level Sensitivity Metrics (Results)

- We provide bit-level sensitivity metrics for ranking weight bits from high to low sensitivity
  - High sensitivity: New EMD >> Original EMD
  - Low sensitivity: New EMD ≤ Original EMD
- How do our metrics compare with a perfect but costly oracle ranking?



FKeras: 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?

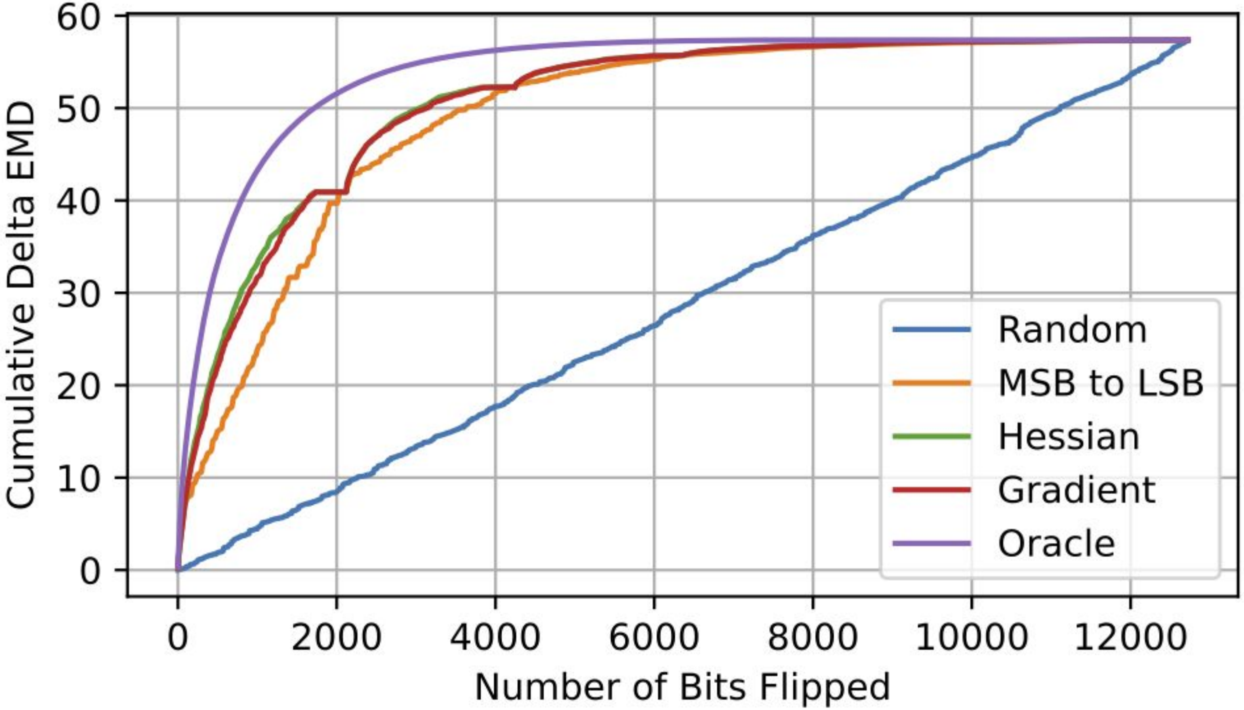
Thank you! Questions?

Thank you! Questions?

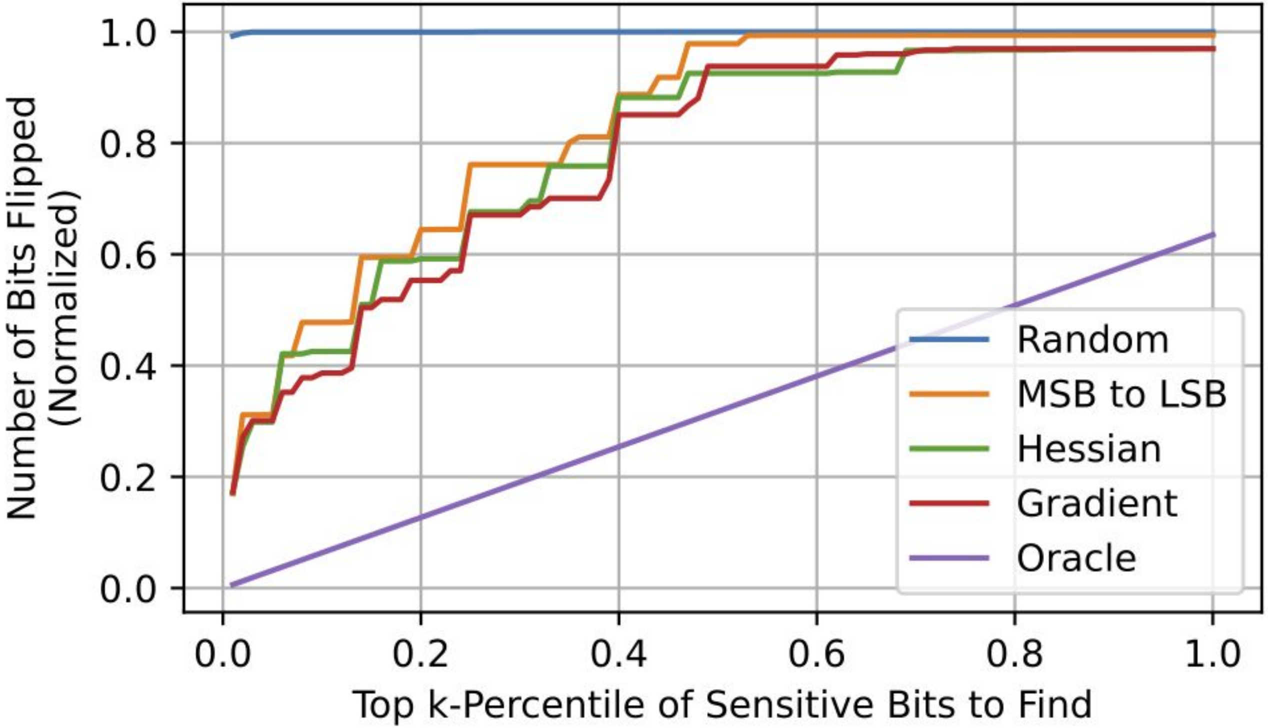
FKeras Repo: https://github.com/KastnerRG/fkeras

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