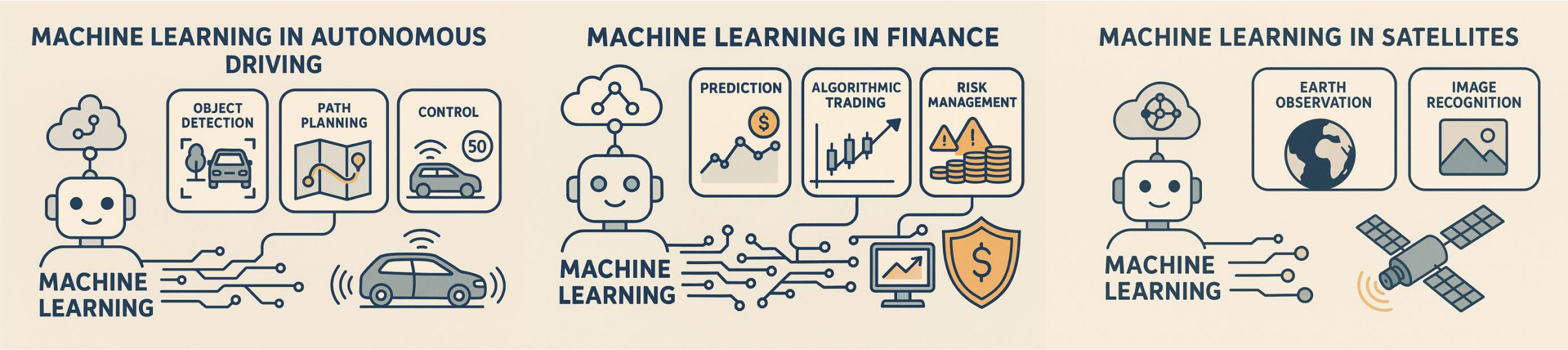
2025 USENIX Annual Technical Conference

SAVE: Software-Implemented Fault Tolerance for Model Inference against GPU Memory Bit Flips

Wenxin Zheng, **Bin Xu**, Jinyu Gu, Haibo Chen

IPADS Lab, Shanghai Jiao Tong University

Characteristics of Machine Learning in the Edge: Result Critical

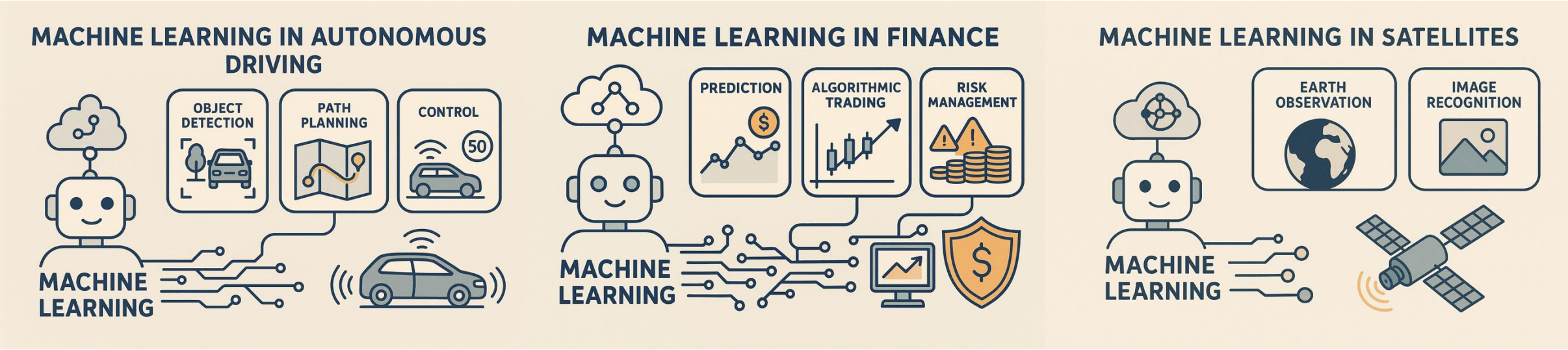


Autonomous Driving

Financial System

Satellite in Orbit

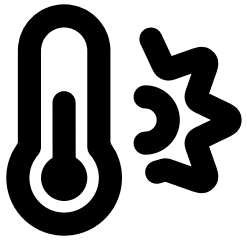
Characteristics of Machine Learning in the Edge: Result Critical



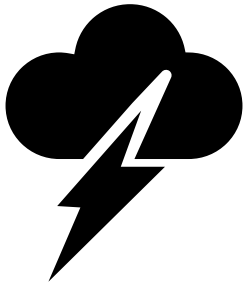
Autonomous Driving

Financial System

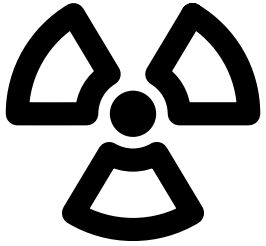
Satellite in Orbit



Changing Temperature

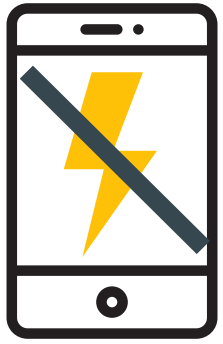


Unstable Voltage

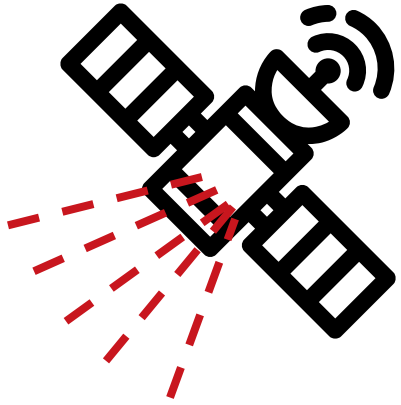


Radiation

Characteristics of Machine Learning in the Edge: Bit Sensitive



3,900 bit flips/day



16 million bit flips/day

Model degradation due to bit flips

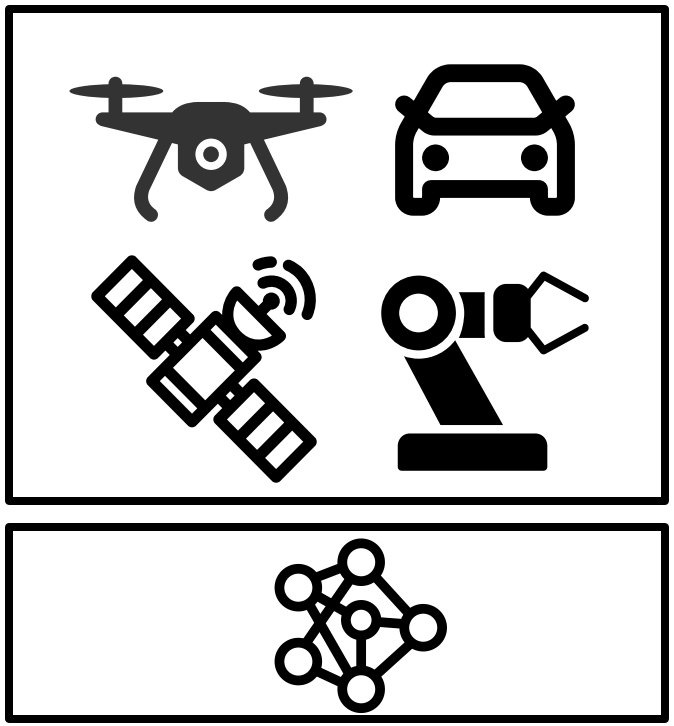
	Model	Flipped Bits	Accuracy Degradation
NaN [29]	EfficientNet	3	82.00% → 0%
FIASM [46]	Softmax(1)	1	97.4% → 62%
TBD [40]	CV(2)	1	up to 99% drop
BFA [73]	ResNet18(3)	13	69.8% → 0.1%
DHB [37]	ResNet20(3)	28	81.39% drop
TBT [74]	ResNet18(3)	84	92% ASR(4)
TA-LBF [9]	ResNet18(3)	1507	100% ASR(4)
T-BFA [75]	ResNet18(3)	27	100% ASR(4)
Our Experiments	ResNet	1	99% Drop
	ViT		
	CogACT		

Memory is **not** always reliable

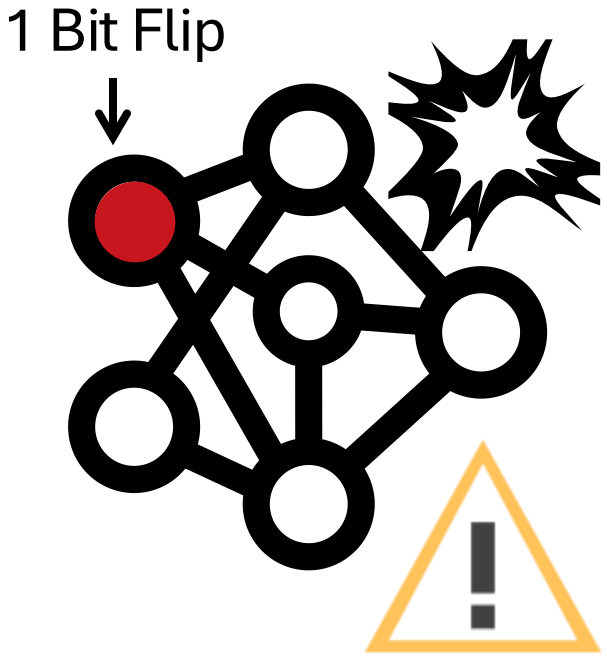
Even **1 Bit flip** can disable whole model

(1): The models with softmax. (2): ResNet, VGG. (3): 8-bit quantization models. (4): ASR: Attack Success Rate.

Protecting Machine Learning in the Edge

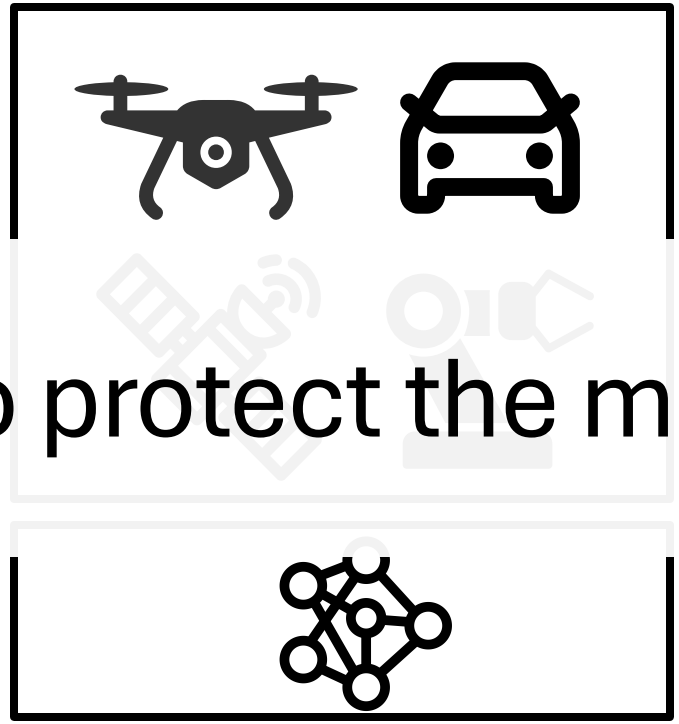


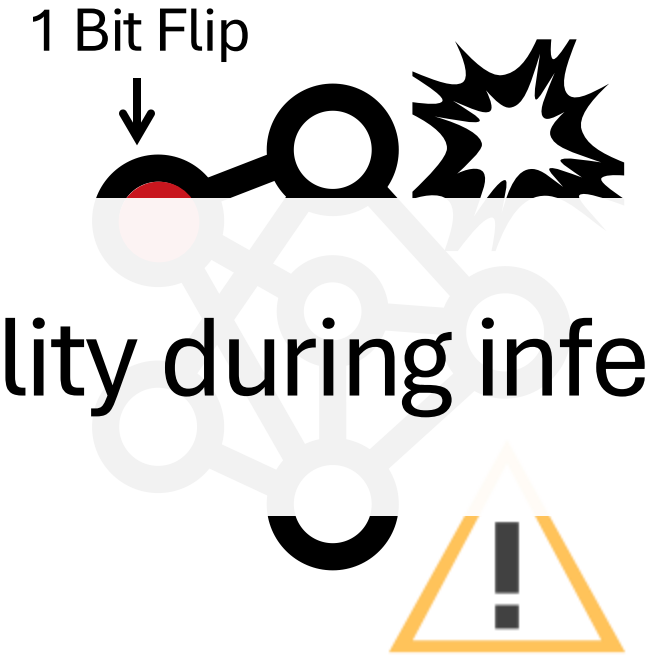
Result Critical



Bit Sensitive

Protecting Machine Learning in the Edge





How to protect the model reliability during inference?

Result Critical

Bit Sensitive

Current Solution

Hardware Solution

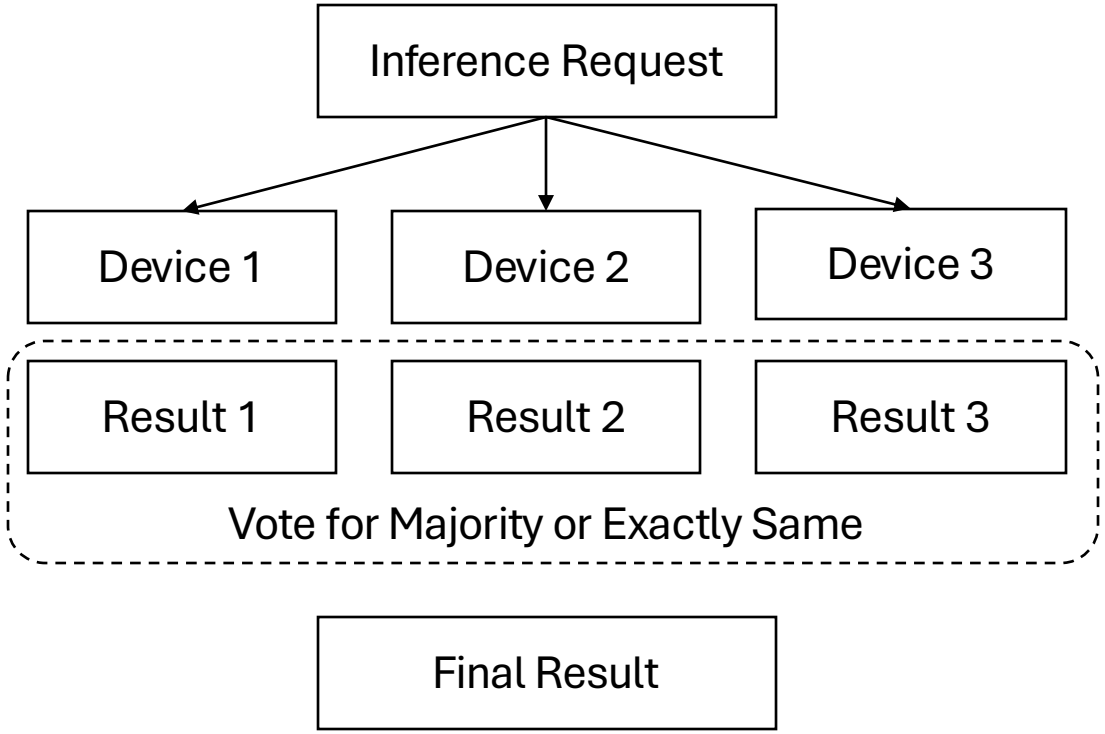
- TMR
- ECC
- Hardware Reliable Only

Software Solution

- Check Distribution of Activations
- Robust Activation Function

Hardware Solution

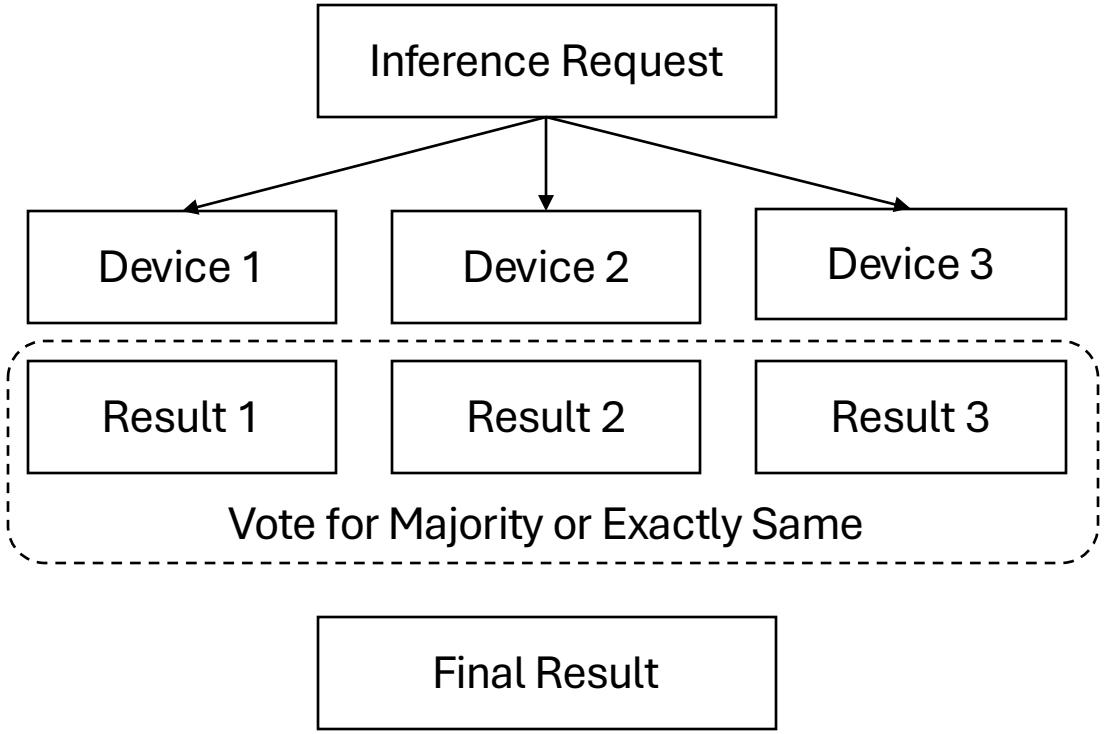
TMR Solution
Triple Modular Redundancy



Require **3x** of computation resources

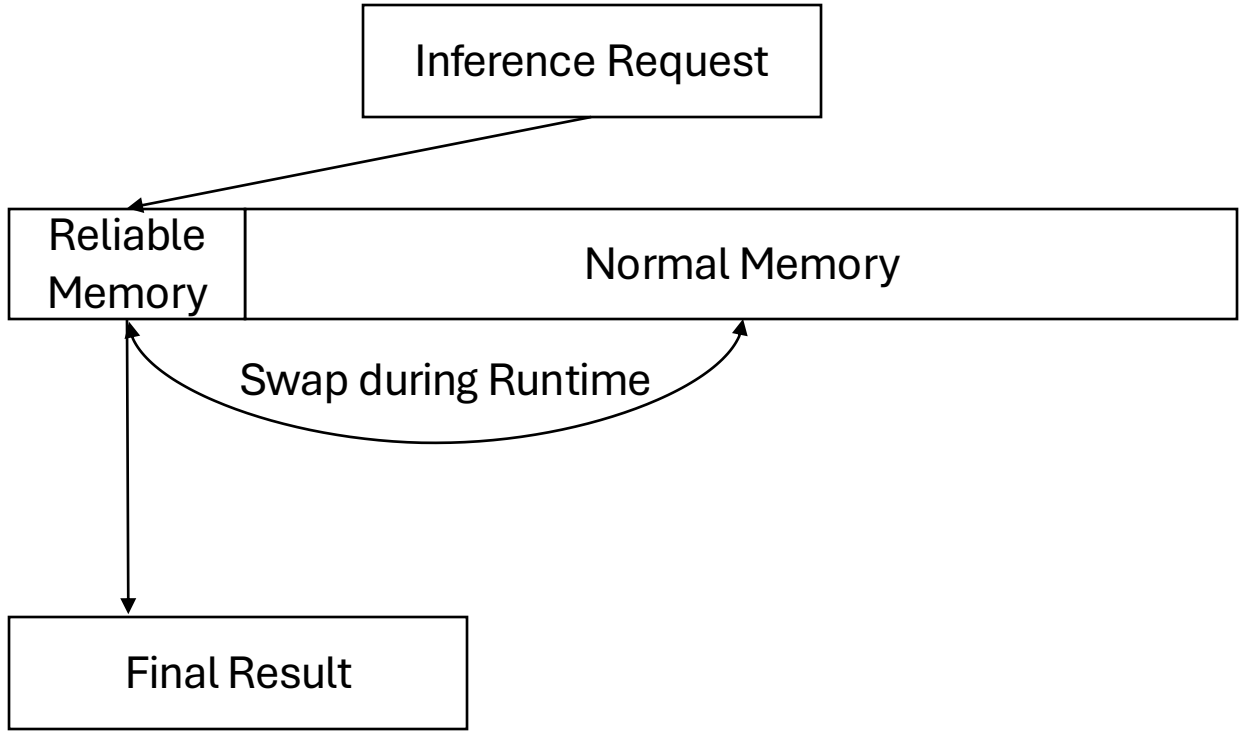
Hardware Solution

TMR Solution
Triple Modular Redundancy



Require **3x** of computation resources

Hardware Reliable Memory Solution
Use reliable memory for all calculation

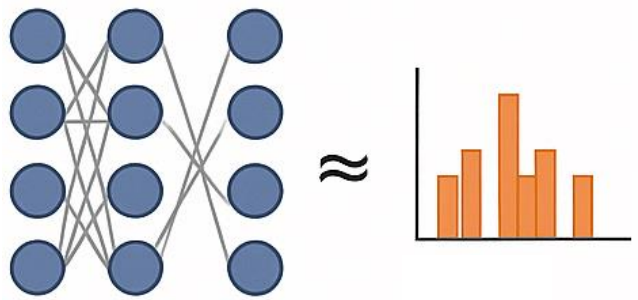


Additional **1000x** runtime overhead or infeasible

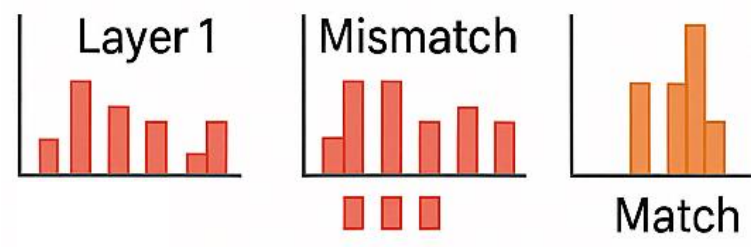
Software Solution

Dr.DNA Solution [ASPLOS'24]

Patch the output with profiled distribution



Step 1: Profile the Distribution



Step 2: Runtime Identification and patching

Accuracy Affected

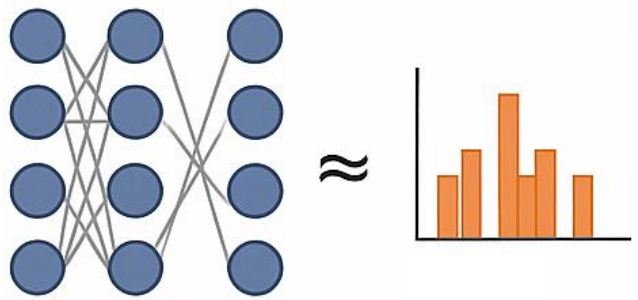
Cannot handle Out-of-Distribution Data

[1] Ma, Dongning, et al. "Dr. DNA: Combating silent data corruptions in deep learning using distribution of neuron activations." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 2024

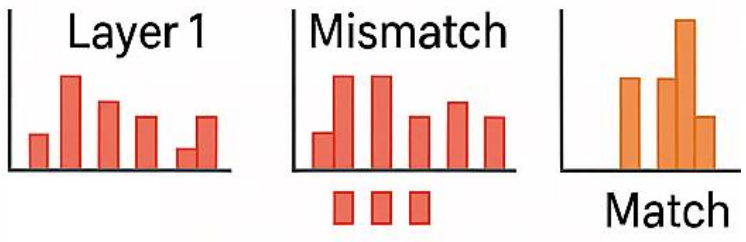
Software Solution

Dr.DNA Solution [ASPLOS'24]

Patch the output with profiled distribution



Step 1: Profile the Distribution



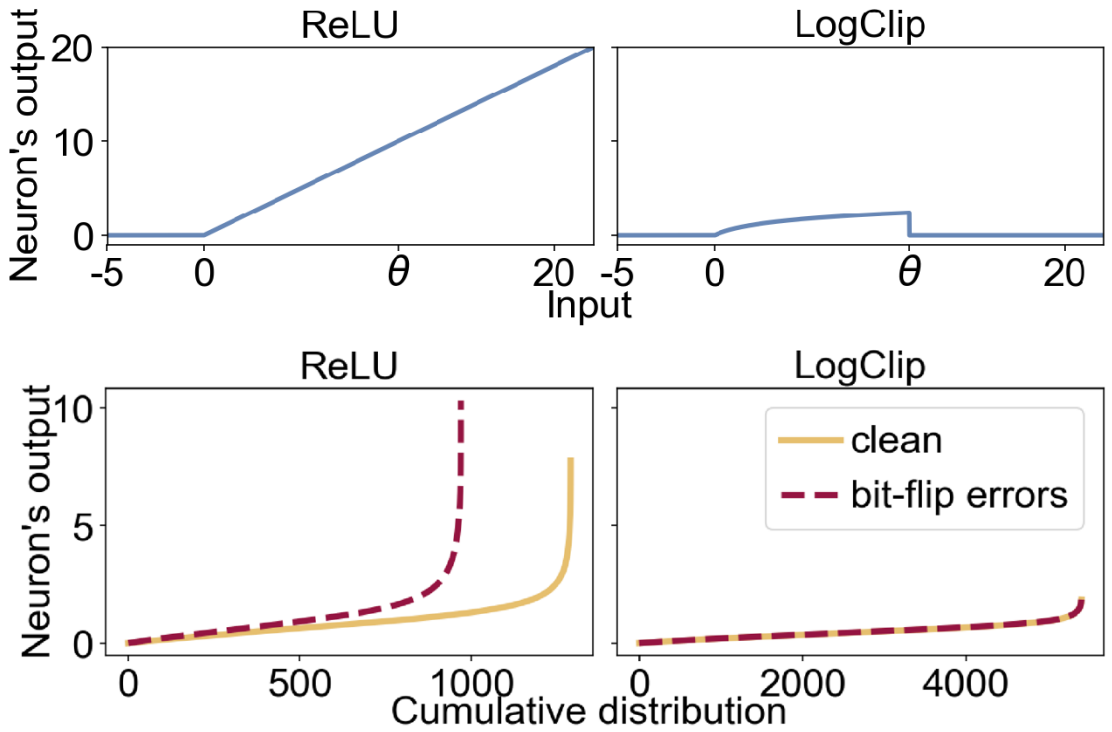
Step 2: Runtime Identification and patching

Accuracy Affected

Cannot handle Out-of-Distribution Data

RedNet Solution [2]

Change the Activation function into Robust one



Retrain Required

Cannot handle activation function other than ReLU

[1] Ma, Dongning, et al. "Dr. DNA: Combating silent data corruptions in deep learning using distribution of neuron activations." *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 2024

[2] Wang, Meiqi, et al. "A case for application-aware space radiation tolerance in orbital computing." arXiv preprint arXiv:2407.11853 (2024)

Our Key Insights

Our Key Insights

Insight-1: Not all hardware bits are created equal.

Our Key Insights

Insight-1: Not all hardware bits are created equal.

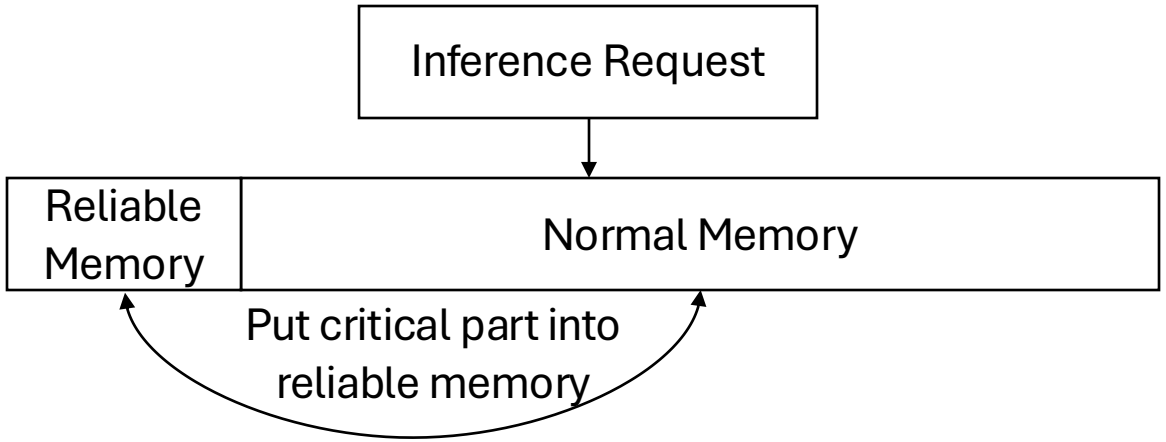
Reliable Memory	Normal Memory

Reliable Memory:

- Soft Error Detection in SRAM
- A small portion of DRAM

Our Key Insights

Insight-1: Not all hardware bits are created equal.

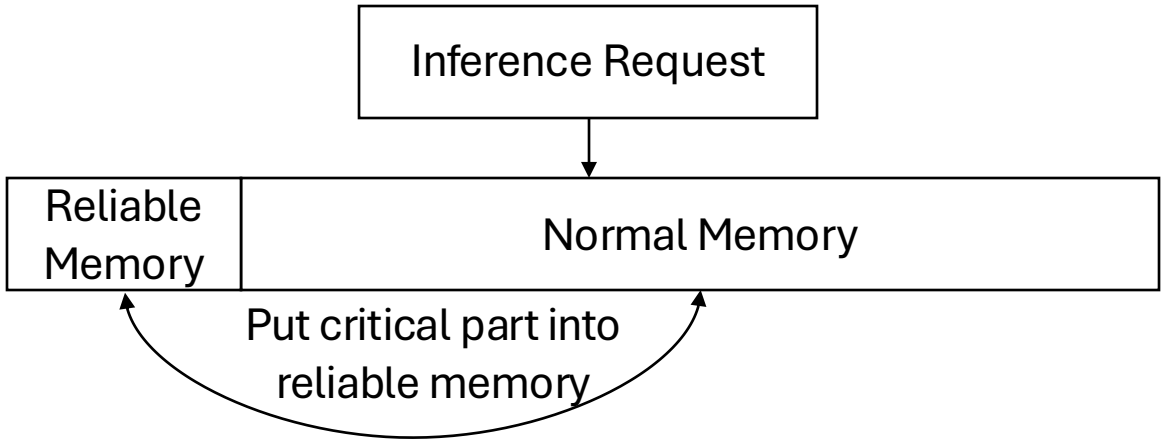


Reliable Memory:

- Soft Error Detection in SRAM
- A small portion of DRAM

Our Key Insights

Insight-1: Not all hardware bits are created equal.



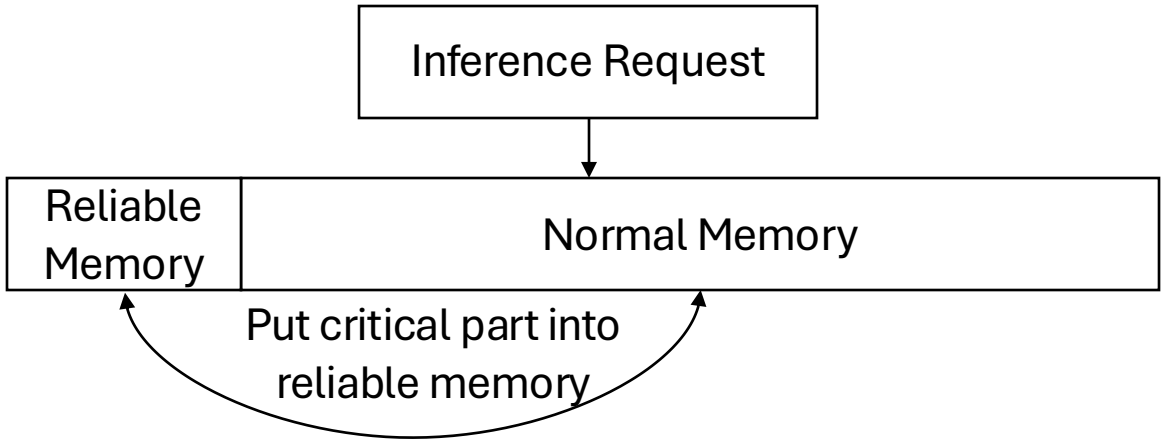
Reliable Memory:

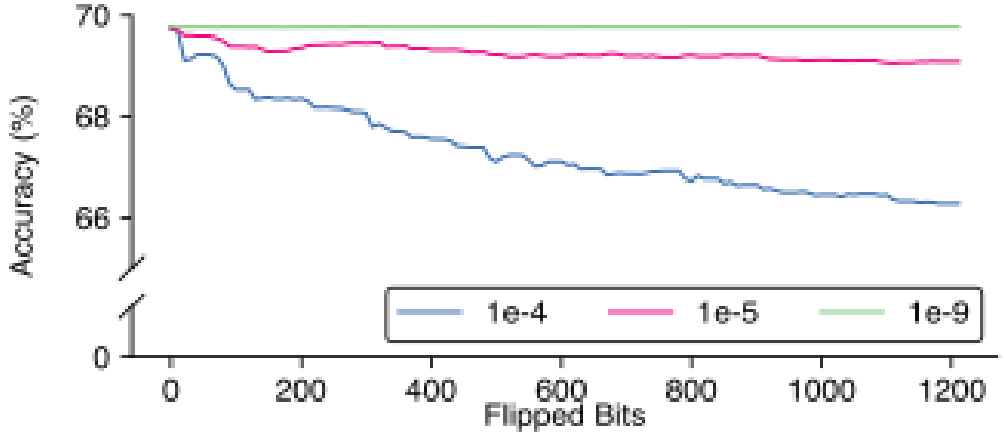
- Soft Error Detection in SRAM
- A small portion of DRAM

Insight-2: Not all bit flips in software are fatal or silent.

Our Key Insights

Insight-1: Not all hardware bits are created equal.





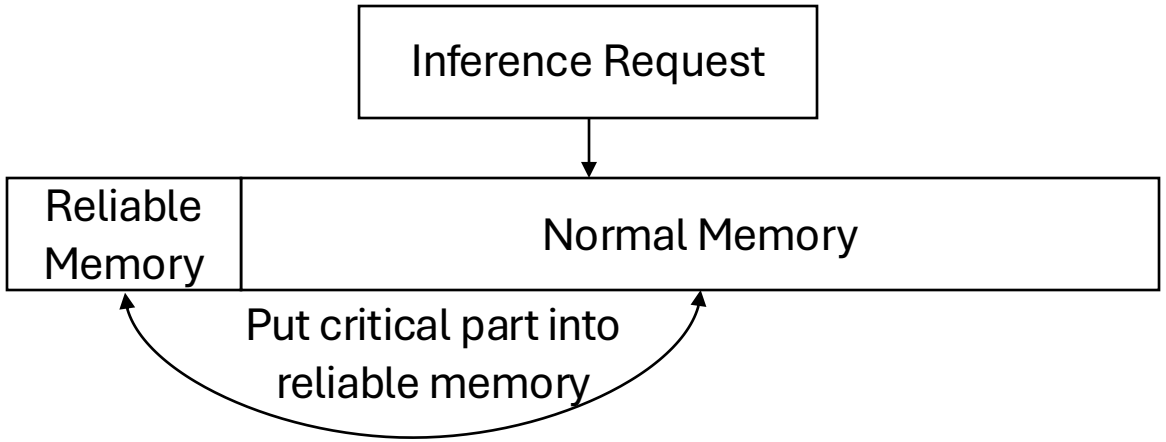
Reliable Memory:

- Soft Error Detection in SRAM
- A small portion of DRAM

Insight-2: Not all bit flips in software are fatal or silent.

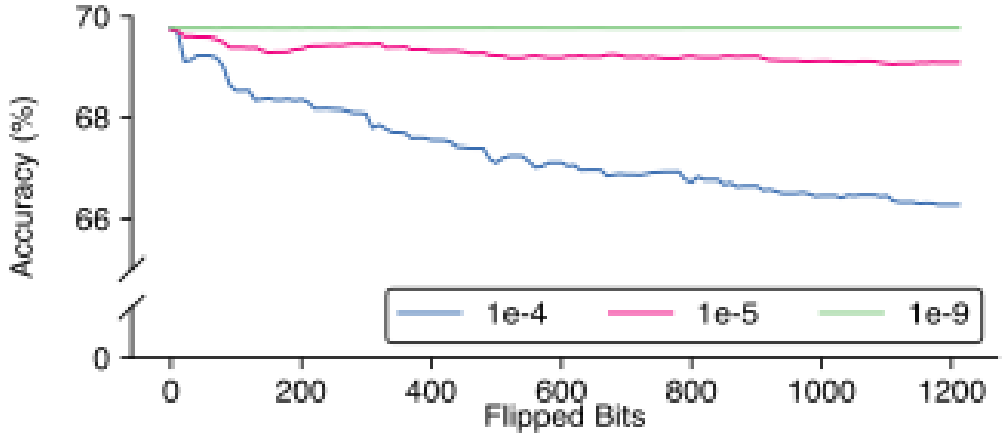
Our Key Insights

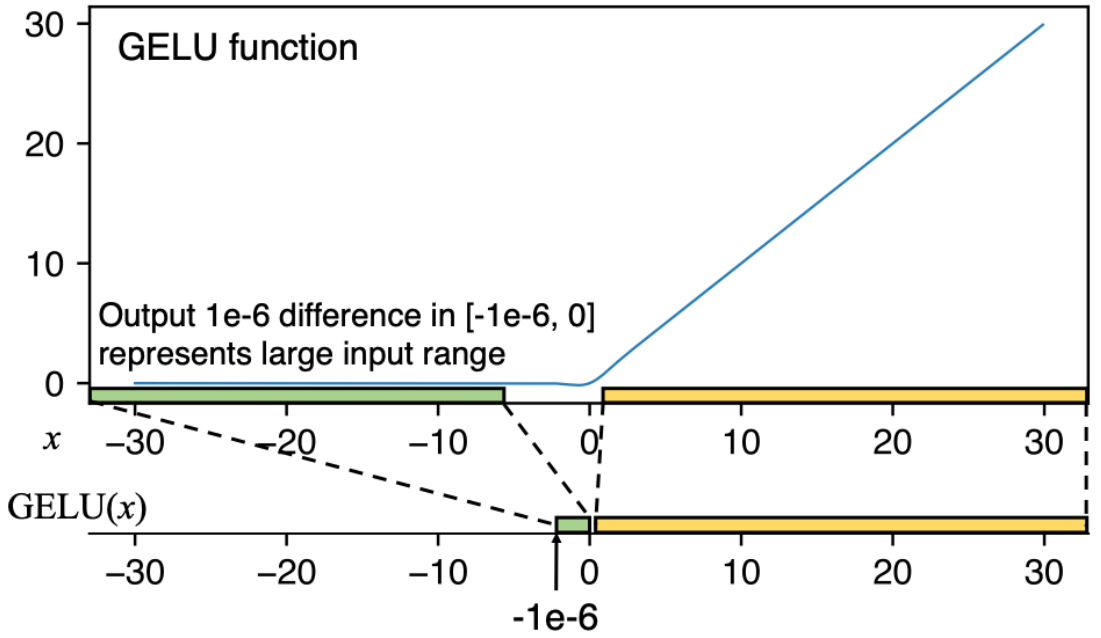
Insight-1: Not all hardware bits are created equal.



Reliable Memory:

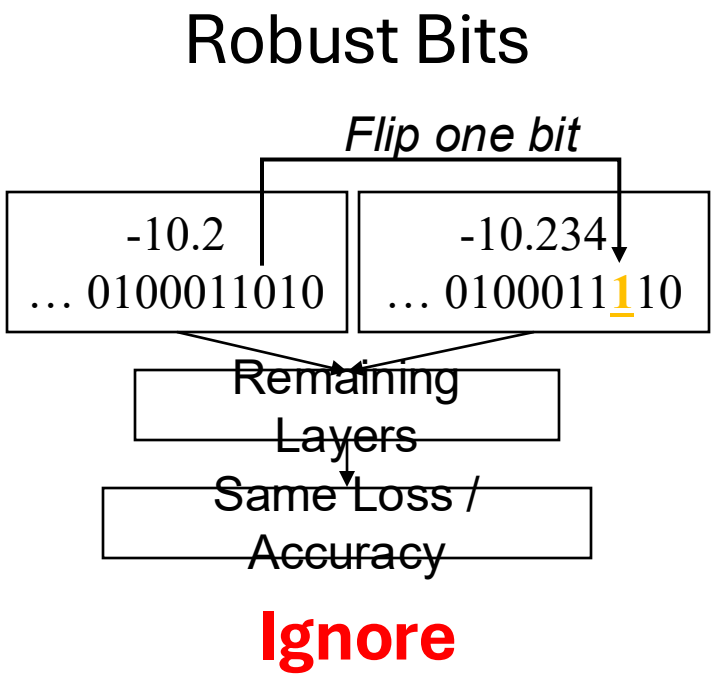
- Soft Error Detection in SRAM
- A small portion of DRAM



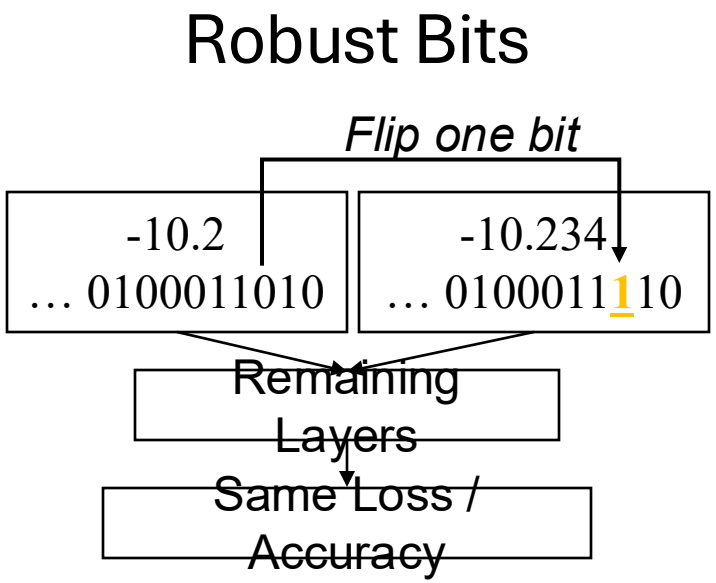


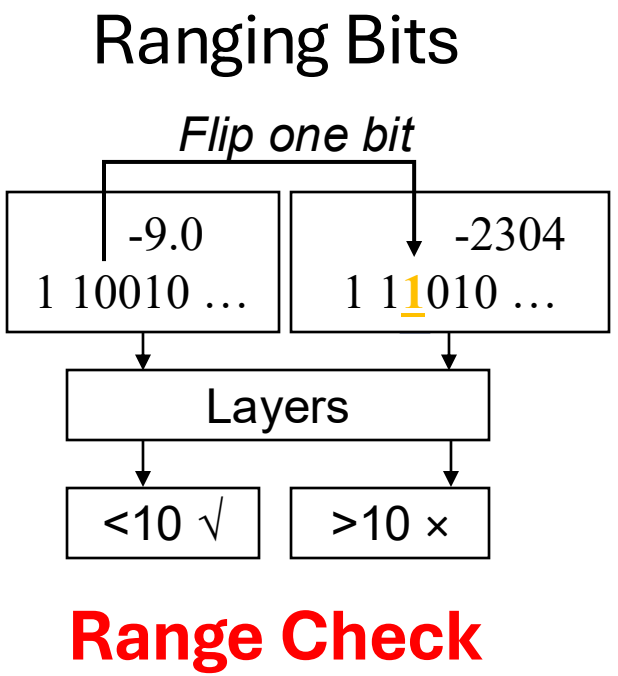
Insight-2: Not all bit flips in software are fatal or silent.

Using Bit Robustness



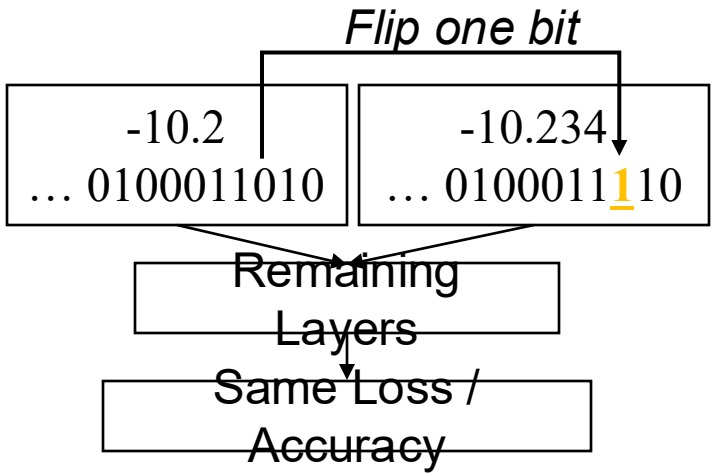
Using Bit Robustness





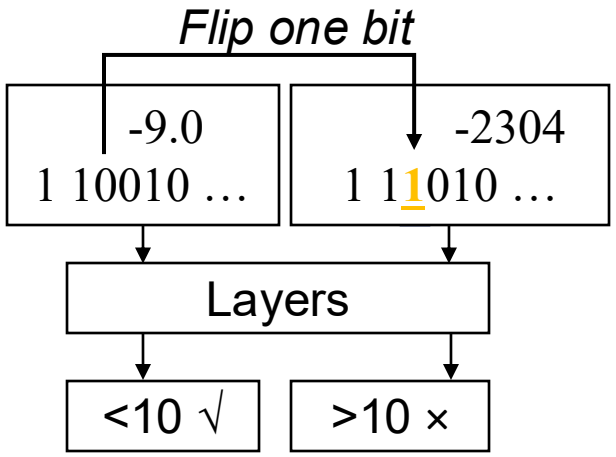
Using Bit Robustness

Robust Bits



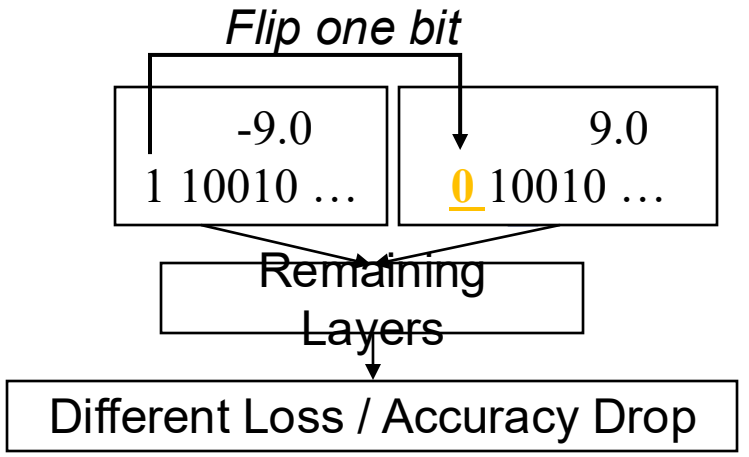
Ignore

Ranging Bits



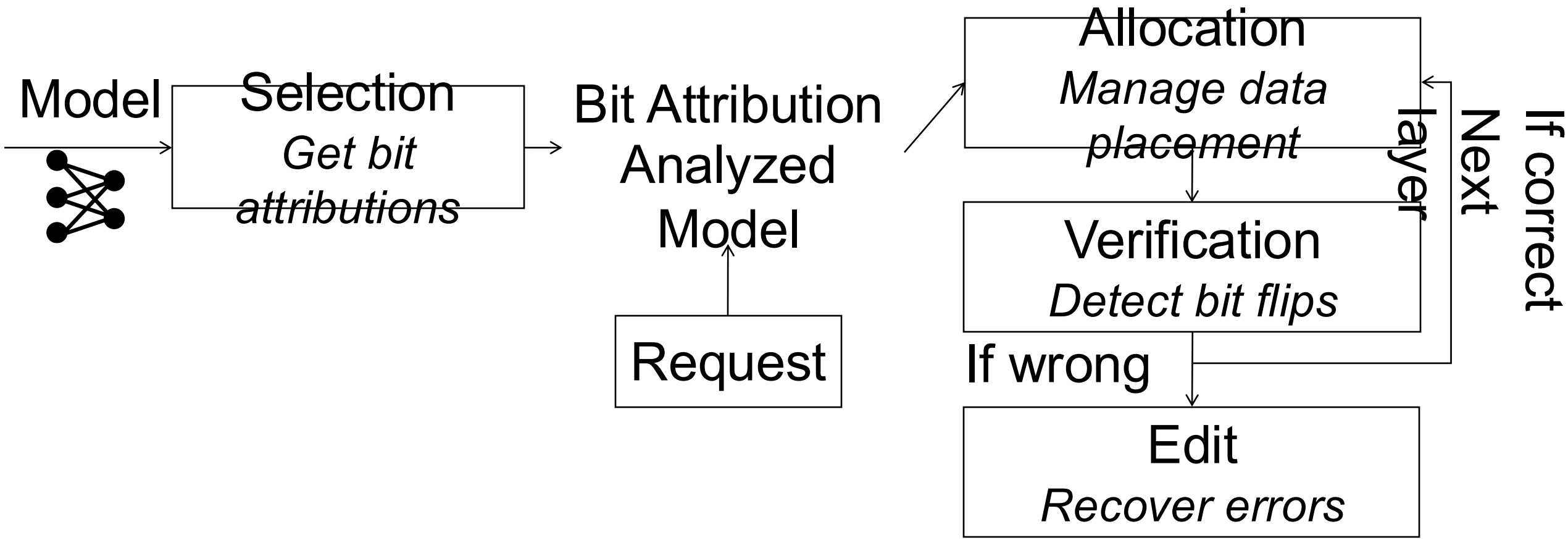
Range Check

Vulnerable Bits

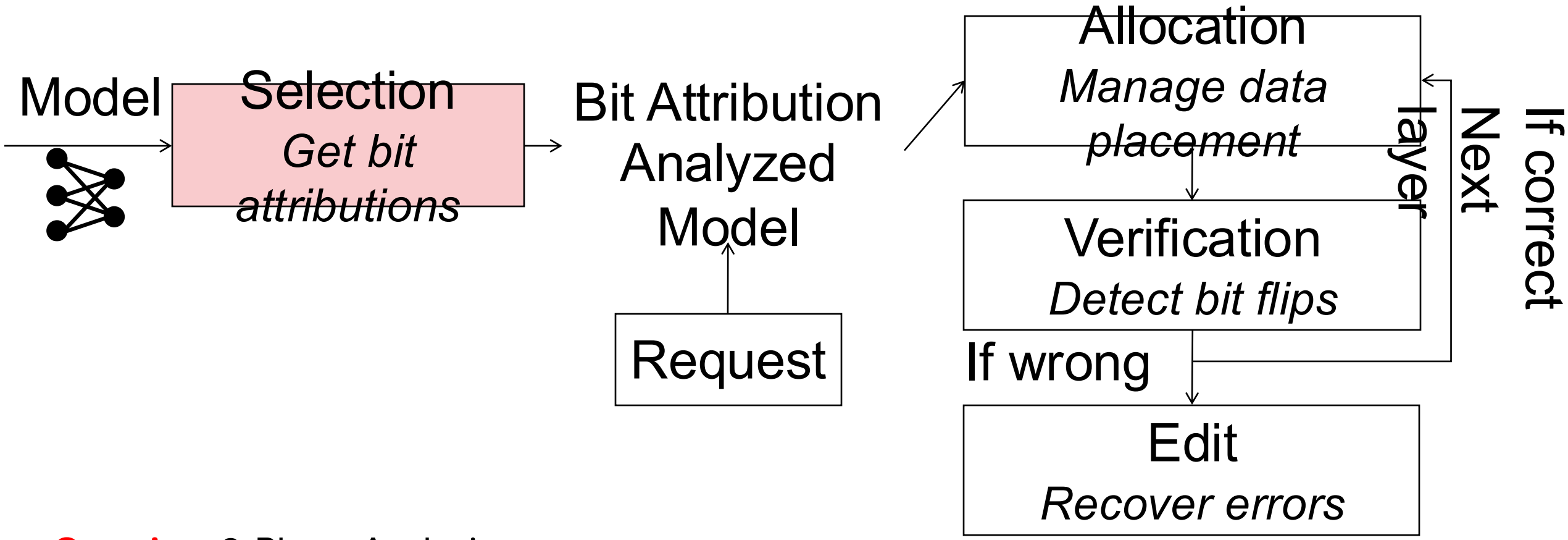


DMR

SAVE Design


Model
Selection
Get bit attributions
Bit Attribution Analyzed Model
Request
Allocation
Manage data placement
Verification
Detect bit flips
If wrong
Edit
Recover errors
Next layer
If correct

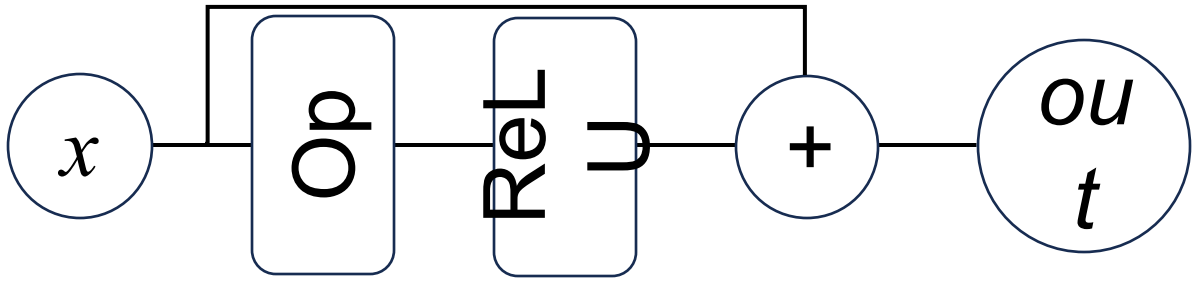
SAVE Design

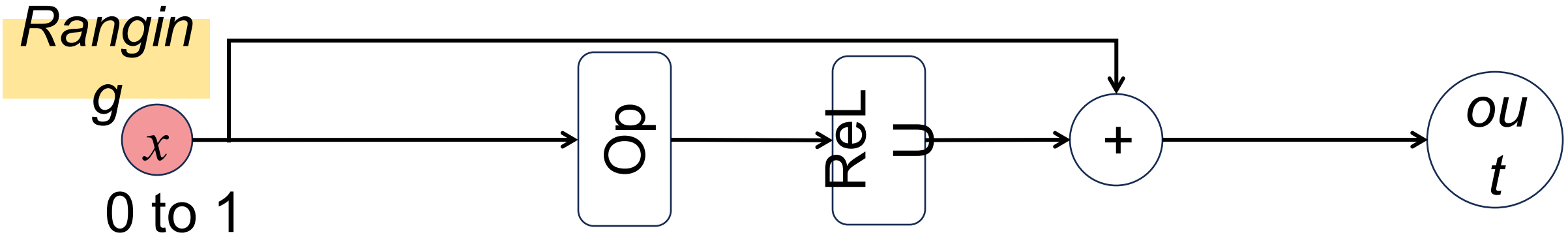


- **One-time** 2-Phase Analysis:
  - Range Analysis: Use DAG to propagate the expected range
  - Bit Attributions: Map ranges into per value bit attributions

Selection Stage-Range Analysis

```
def NonResidualExample(x):
    x = relu(Op(x)) + x
    return x
```

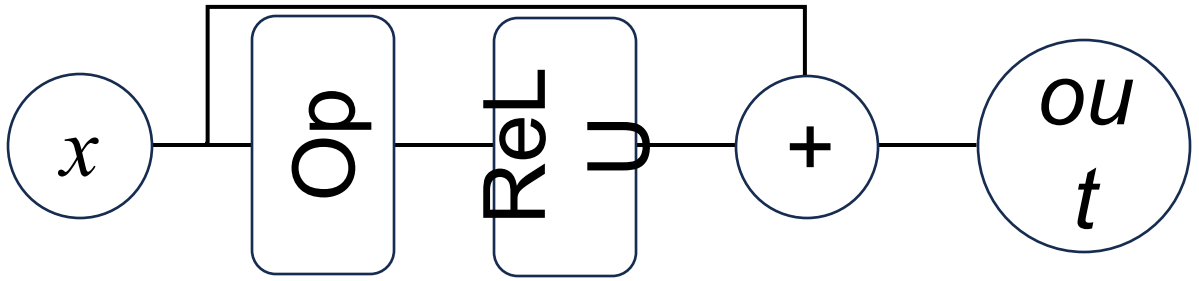


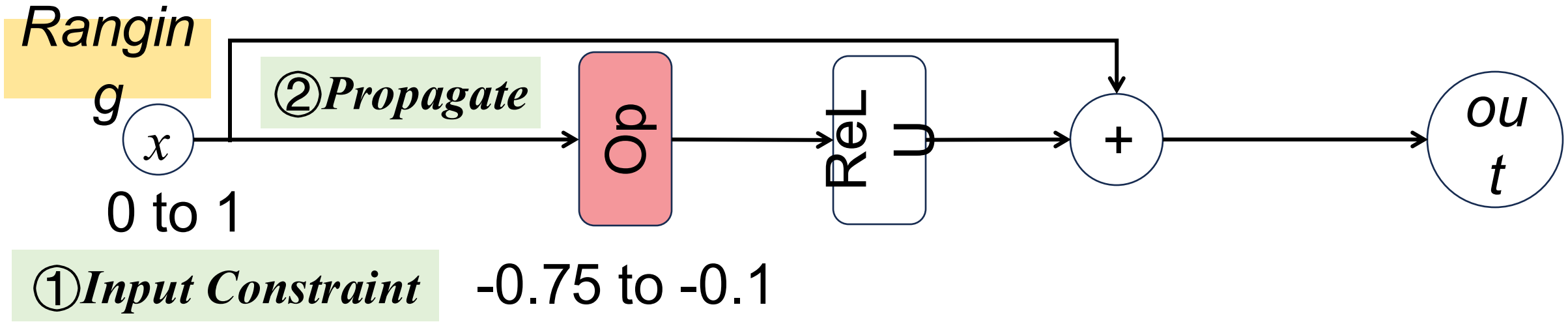


①*Input Constraint*

Selection Stage-Range Analysis

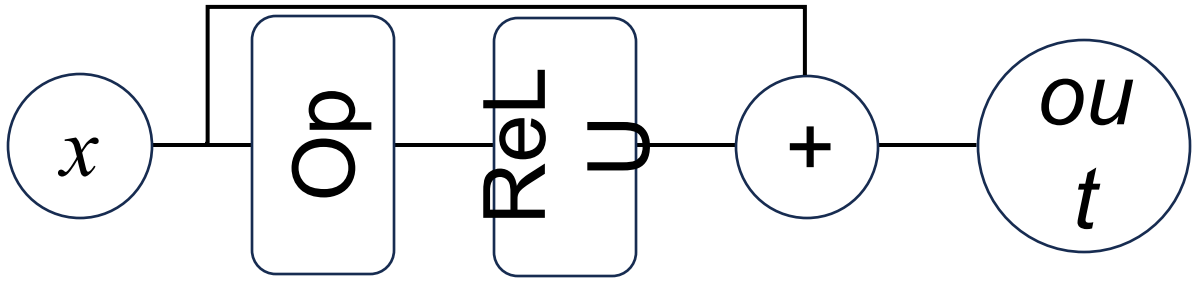
```
def NonResidualExample(x):
    x = relu(Op(x)) + x
    return x
```

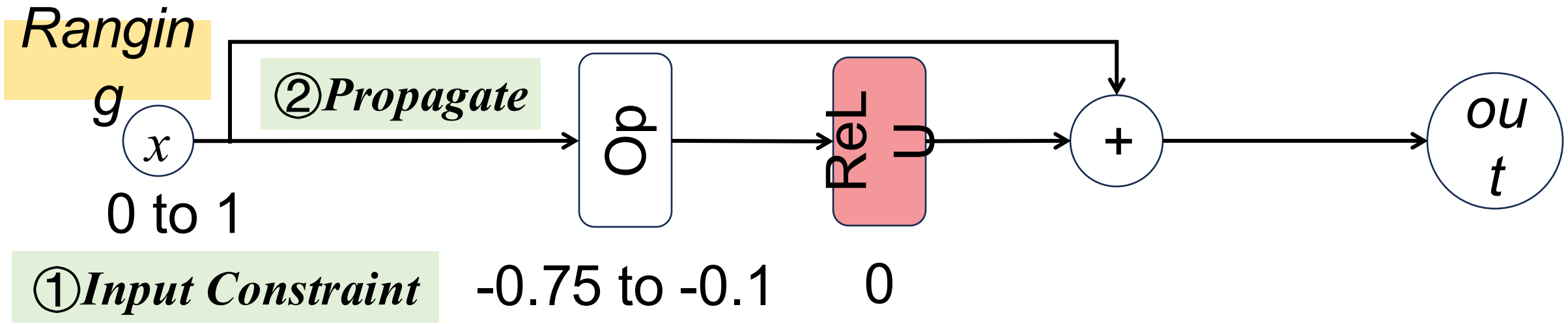




Selection Stage-Range Analysis

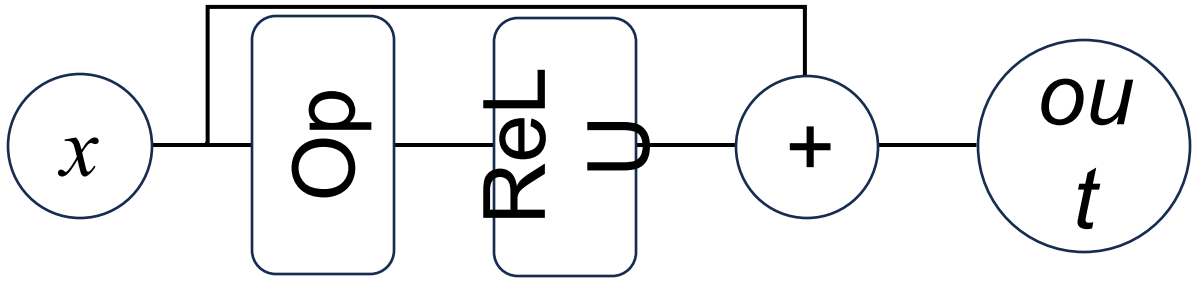
```
def NonResidualExample(x):
    x = relu(Op(x)) + x
    return x
```

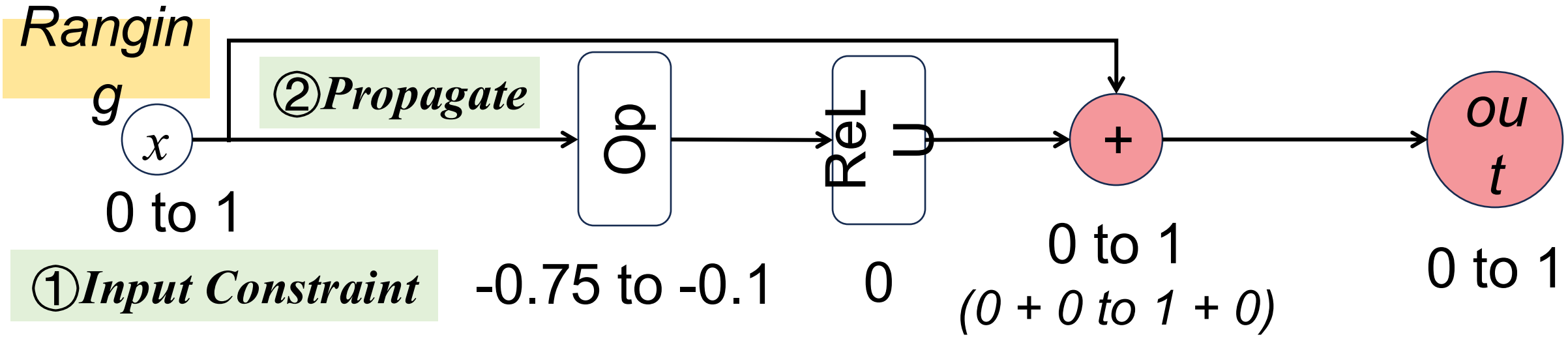




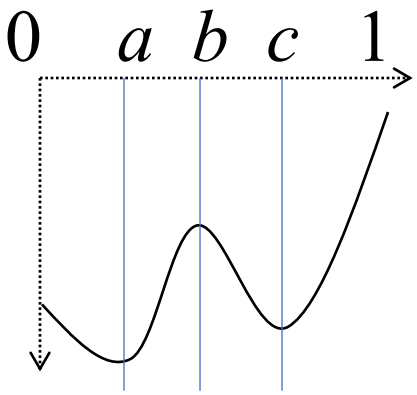
Selection Stage-Range Analysis

```
def NonResidualExample(x):
    x = relu(Op(x)) + x
    return x
```





Selection Stage-Bit Attribution Analysis



Ranging Bits

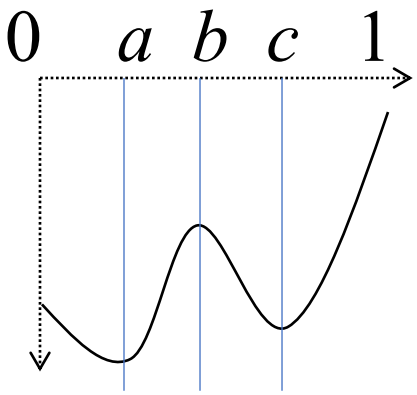
Input	Output	Ranging
0 to a	-0.65 to -0.75	9 Bits
…	…	…
c to 1	-0.7 to -0.1	6 Bits

Value	Binary
-0.65	**1 01111110** 0…
…	…
-0.75	**1 01111110** 1…

9 Ranging B

Map to bits

Selection Stage-Bit Attribution Analysis



Input	Output	Ranging
0 to a	-0.65 to -0.75	9 Bits
…	…	…
c to 1	-0.7 to -0.1	6 Bits

Value	Binary
-0.65	**1 01111110** 0…
…	…
-0.75	**1 01111110** 1…

Ranging Bits

9 Ranging B

Map to bits

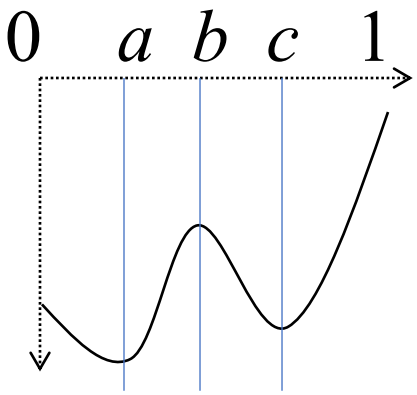
Range	Binary(Significand)	Value
-0.65 to -0.75	…10011**001100110**	-0.65
	…10011**100001110**	-0.65001

9 Robust

Robust by Numerical

Robust Bits

Selection Stage-Bit Attribution Analysis



Input	Output	Ranging
0 to a	-0.65 to -0.75	9 Bits
…	…	…
c to 1	-0.7 to -0.1	6 Bits

Value	Binary
-0.65	**1 01111110** 0…
…	…
-0.75	**1 01111110** 1…

Ranging Bits

9 Ranging B

Map to bits

Range	Binary(Significand)	Value
-0.65 to -0.75	…10011**001100110**	-0.65
	…10011**100001110**	-0.65001

9 Robust

Robust by Numerical

Robust Bits

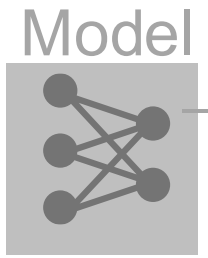
-0.75 to -1 → ReLU → 0

Any negative values give same results.

31 Robust Bits

Robust by Op

SAVE Design



Model

Selection
Get bit attributions

Bit Attribution
Analyzed Model

Allocation
Manage data placement

Next layer

If correct

Verification
Detect bit flips

If wrong

Edit
Recover errors

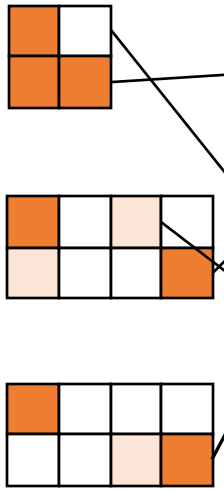
- Place the value with less robust bits into reliable memory

Input

Layer 1

Layer 2

...



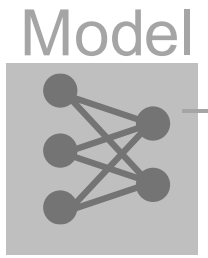
Hardware Reliable

Other Memory

Robust Bit Count

More Less

SAVE Design



Model

Selection
Get bit attributions

Bit Attribution
Analyzed Model

Allocation
Manage data placement

If correct
Next
layer

- Using different verification method
  - Robust bits: Ignore
  - Ranging bits: Range Check
  - Vulnerable bits: Double Modular Redundancy

Verification
Detect bit flips

If wrong

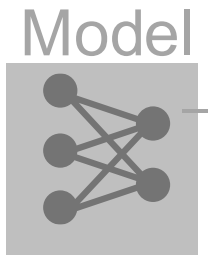
Edit
Recover errors

GPU — *Compute Op1*

GPU Async

CPU

SAVE Design



Model

Selection
Get bit attributions

Bit Attribution
Analyzed Model

Allocation
Manage data placement

If correct
Next
layer

- Using different verification method
  - Robust bits: Ignore
  - Ranging bits: Range Check
  - Vulnerable bits: Double Modular Redundancy

Verification
Detect bit flips

If wrong

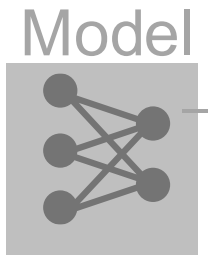
Edit
Recover errors

GPU | *Compute Op1* | *Compute Op2*

GPU Async | *Result, Weight Copy to CPU*

CPU | *Weight Op1*

SAVE Design



Selection
Get bit attributions

Bit Attribution
Analyzed Model

Allocation
Manage data placement

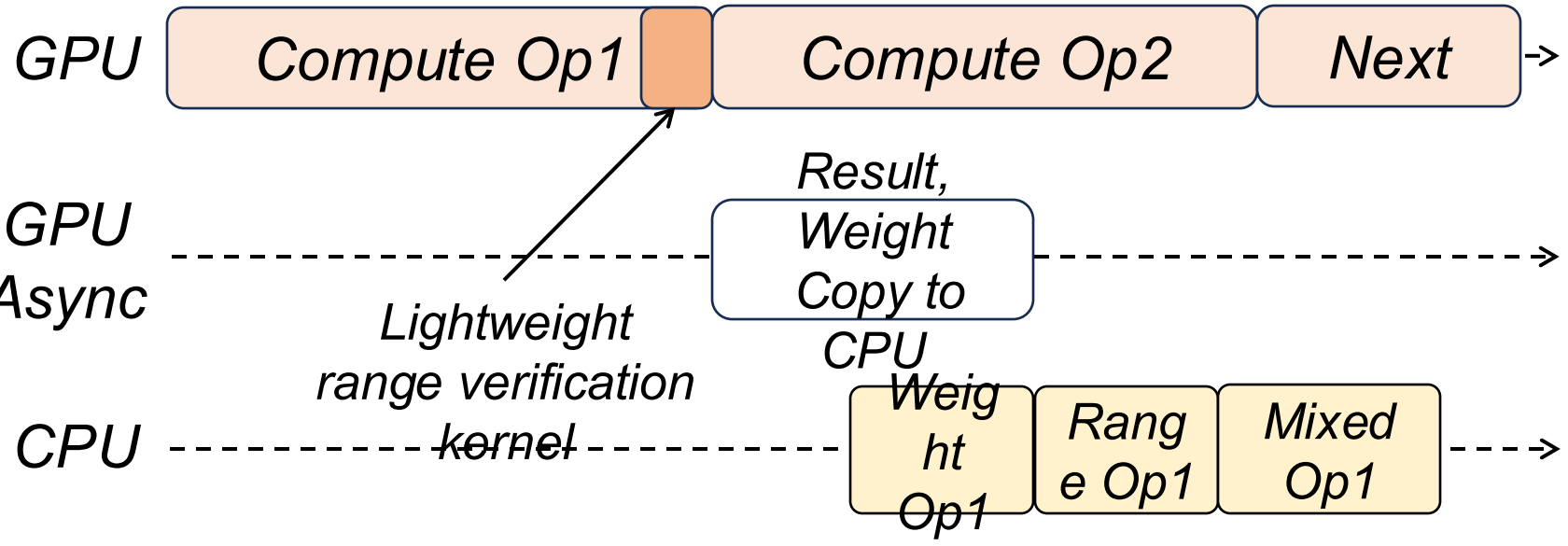
If correct
Next layer

- Using different verification method
  - Robust bits: Ignore
  - Ranging bits: Range Check
  - Vulnerable bits: Double Modular Redundancy

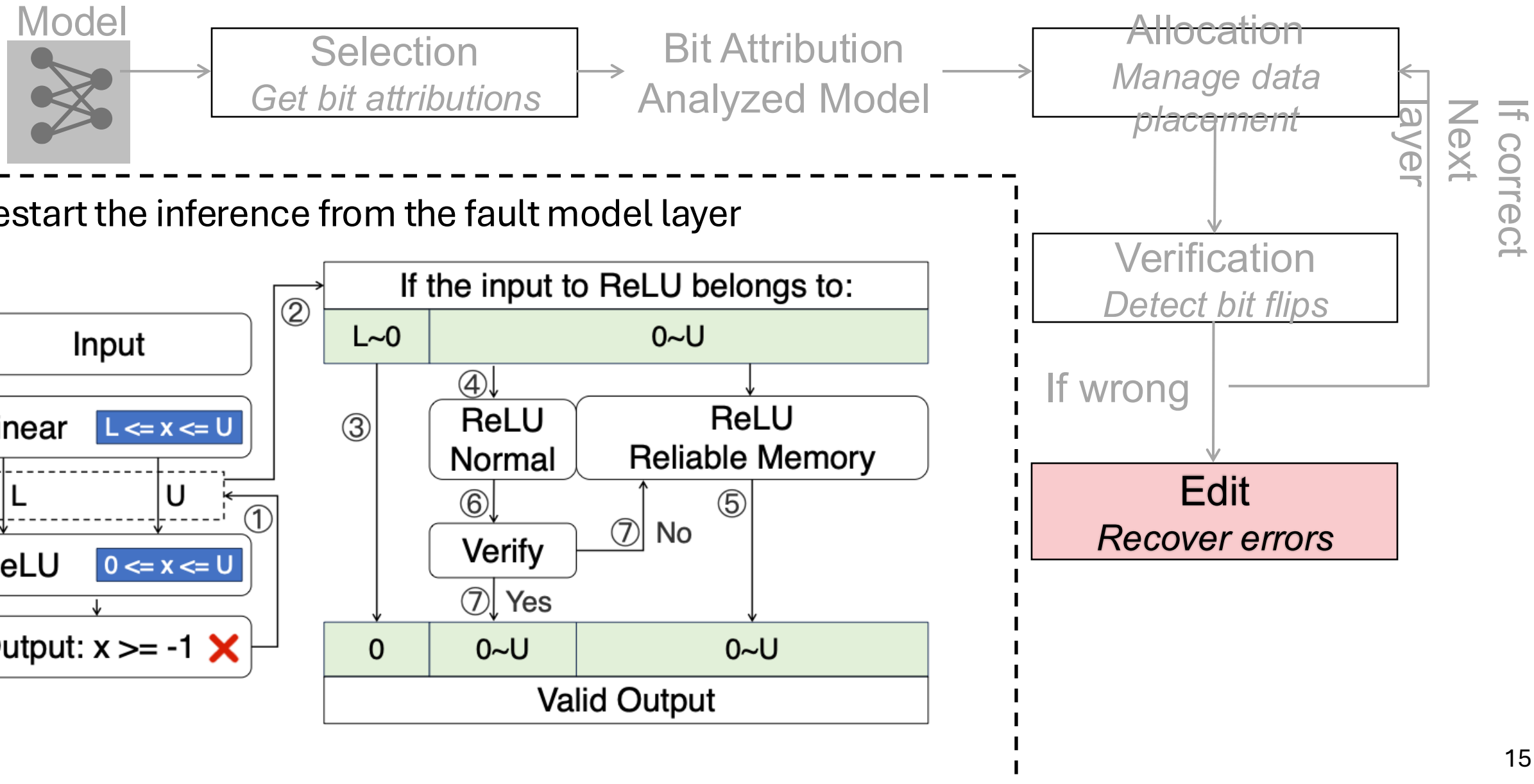
Verification
Detect bit flips

If wrong

Edit
Recover errors



SAVE Design



- Restart the inference from the fault model layer

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

Correct Inference

1 - ErrorRate

Latency

$$Latency \times (1 - ErrorRate)$$

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

Correct Inference
1 - ErrorRate

Latency

Wrong Inference
ErrorRate

Latency

$$Latency \times (1 - ErrorRate)$$

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

Correct Inference
1 - ErrorRate

Latency

$$Latency \times (1 - ErrorRate)$$

Wrong Inference
ErrorRate

Latency | Latency

$$2 \times Latency \times ErrorRate$$

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

Correct Inference
1 - ErrorRate

Latency

$$Latency \times (1 - ErrorRate)$$

Wrong Inference
ErrorRate

Latency | Latency

$$2 \times Latency \times ErrorRate$$

Total: $Latency \times (1 + ErrorRate)$

New Metric: AccurateLatency

$$AccurateLatency = Latency \times (1 + ErrorRate)$$

Assume the second inference must be right

Correct Inference
1 - ErrorRate

Latency

$Latency \times (1 - ErrorRate)$

Wrong Inference
ErrorRate

Latency | Latency

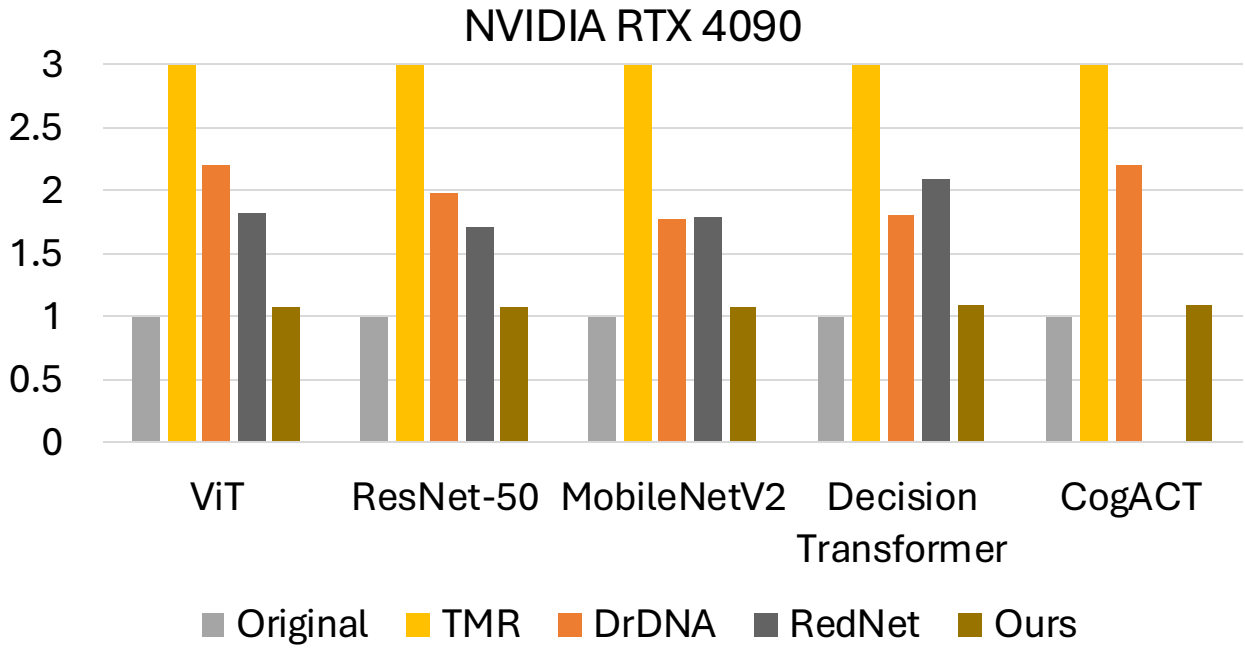
$2 \times Latency \times ErrorRate$

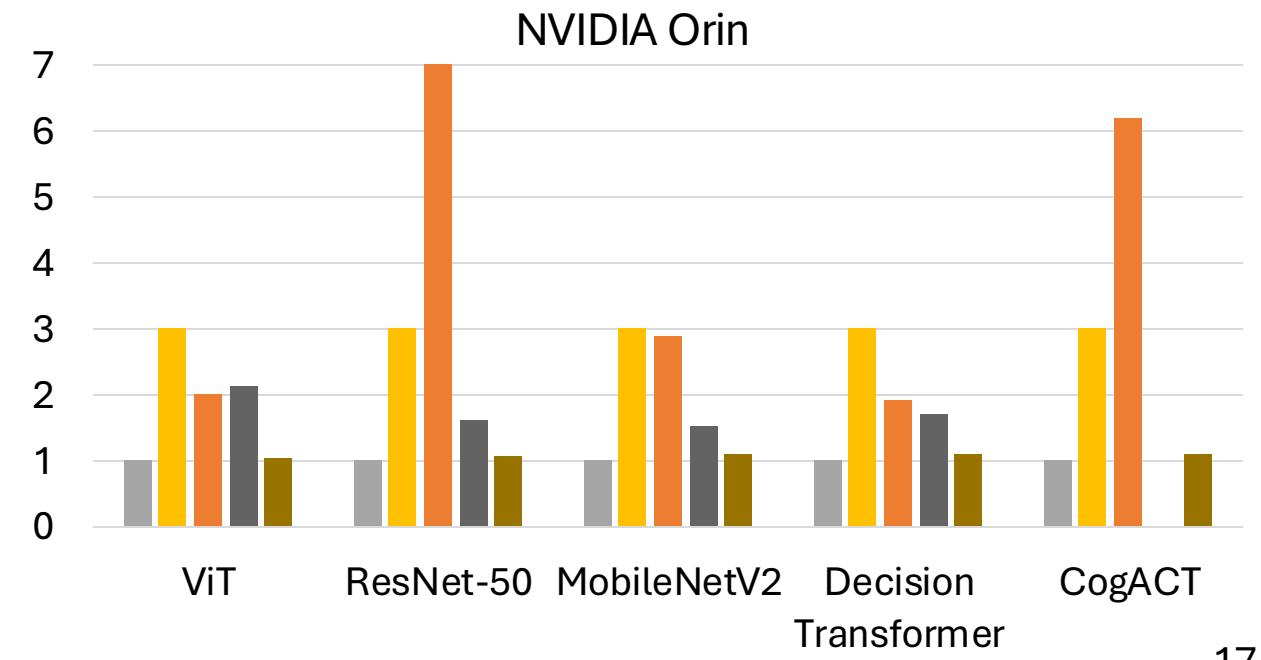
Total: $Latency \times (1 + ErrorRate)$

Add correctness into the latency

End-to-end Performance

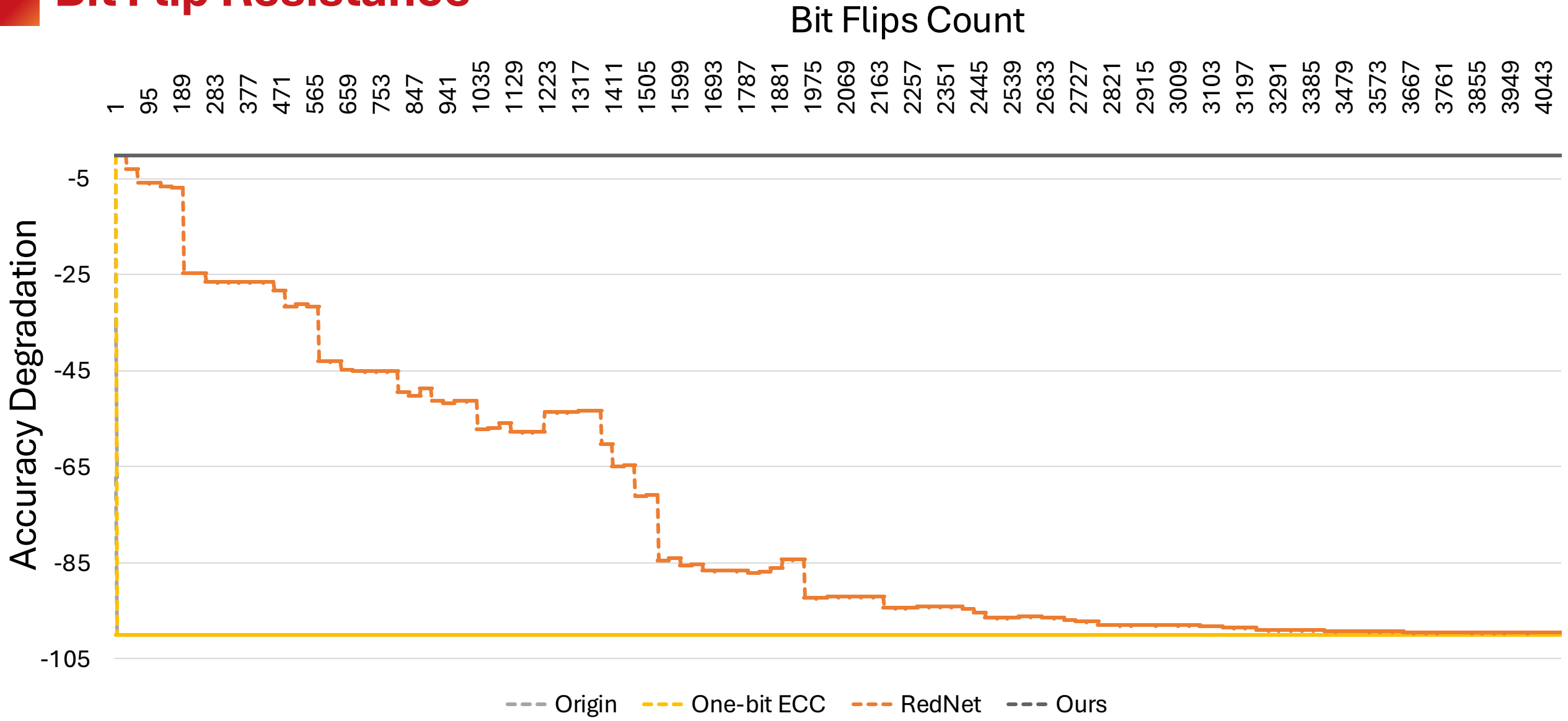
- Performance
  - Overall Accurate Latency **<9%**
  - Accuracy Degradation **<1%**
- Human Efforts
  - Modification to user code: **0 LoC**
  - Retraining Time: **0 Second**





More evaluation can be found in our paper

Bit Flip Resistance



SAVE can keep the model accuracy **under 4000 bit flips** across different models

Conclusion

- Software-implemented fault tolerance for machine learning models
- Key reason for Improvement: The differences in the bit properties inherent in computations within machine learning
- Evaluation shows that the overhead is as small as 9%
  - General to all models
  - No need for additional retraining or fine-tuning

Thanks!