Customized Algorithm-Based Fault Tolerance for Modern AI Accelerators

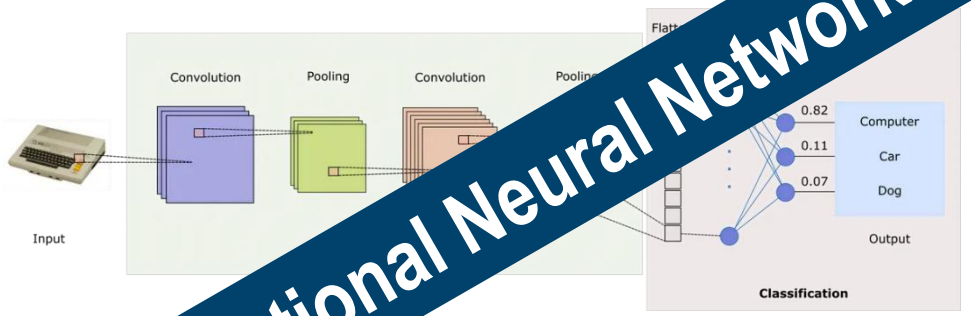
Giorgos Dimitrakopoulos

Electrical and Computer Engineering

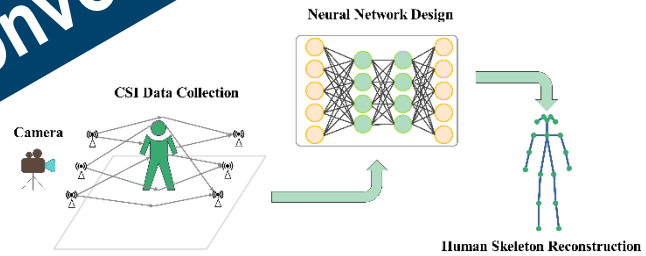
Democritus University of Thrace, Xanthi, Greece

Machine Learning

Convolutional Neural Networks



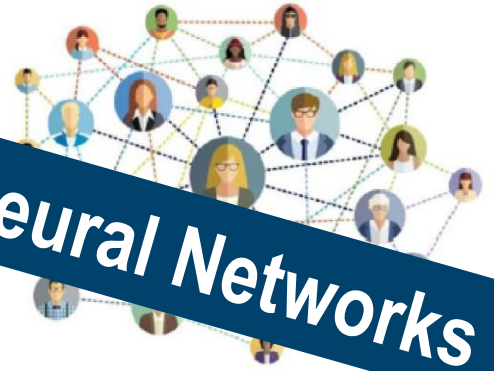
ge Classification



Human Skeletonization

Graph Neural Networks

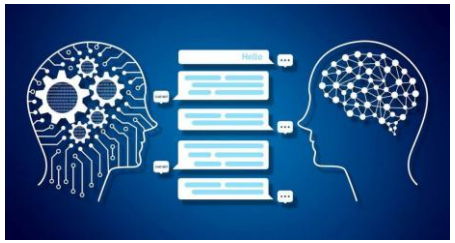
Underground Networks



Social Networks

Transformers

Text-to-
Transfor



Natural Language Processing

Communication Networks

Recommendation Systems

Reliability in ML applications

- **Prevent catastrophic failures:** Errors in safety-critical DNN applications (e.g., autonomous vehicles) can have severe real-world consequences.

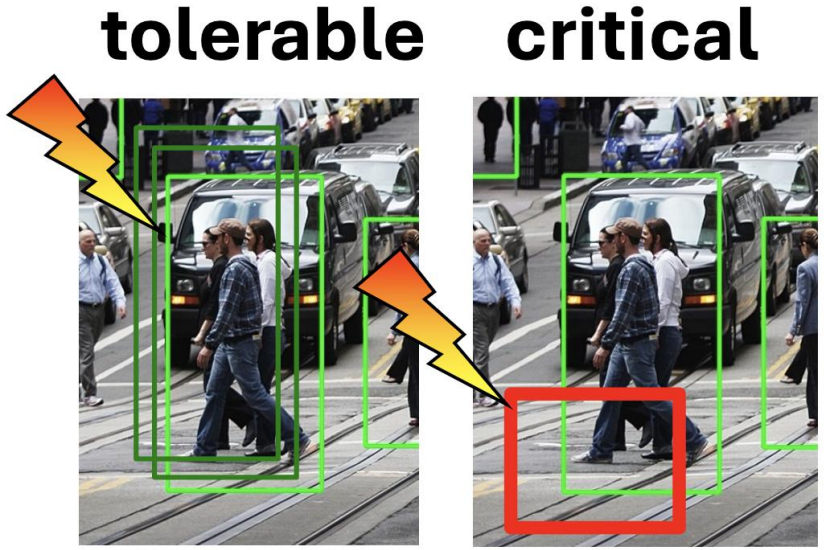
- **Increasing soft-error vulnerability:** Technology scaling makes modern hardware more susceptible to transient faults.

- **Protect active computations:** Stored data can be hardened, but computation requires runtime, algorithm-level error detection.
  - Detecting errors fast (close to their appearance) reduces recovery latency and re-computation cost.

Algorithm based Fault tolerance

- We use online techniques to detect **computation errors during execution**
  - **Signature-based detection:** Compare a compact computation signature (e.g., checksum) against the expected value.
  - **Invariant-based detection:** Verify mathematical properties that must always hold during execution.
  - **Checker logic:** Detect errors by validating signatures or invariants at runtime operating in parallel to design-under-test.

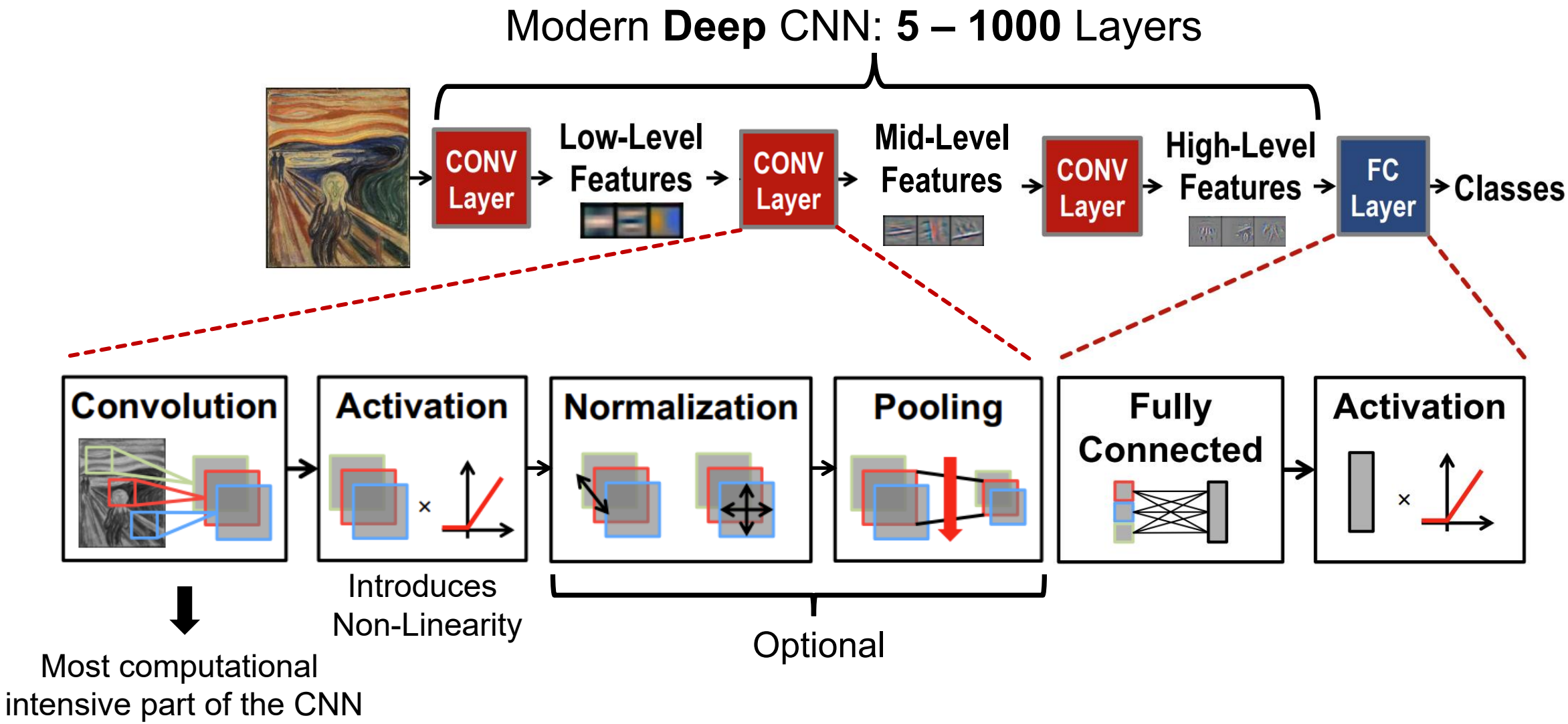
- **Focus on algorithmic correctness:** Detect all computation faults, even if they are not critical and do not change the final DNN prediction (e.g., unchanged classification result).



Part 1

CONVOLUTIONAL NEURAL NETWORKS

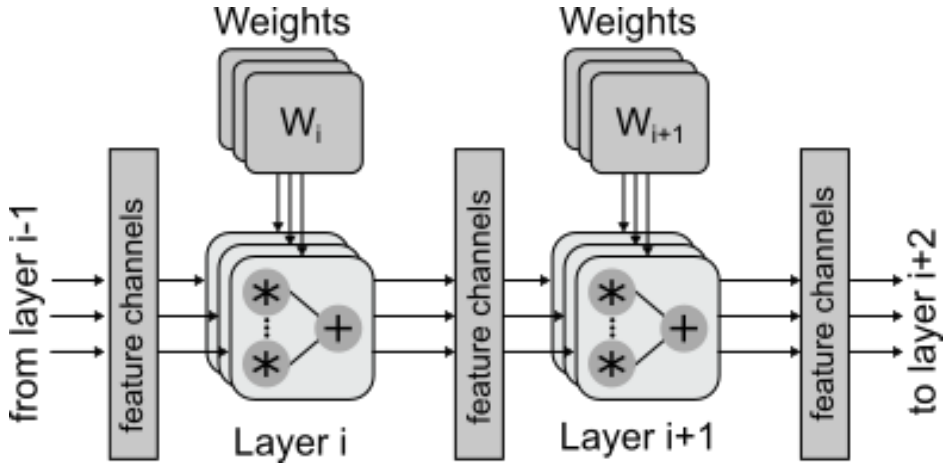
The structure of a Convolutional Neural Network



Types of CNN accelerators for ASICs and FPGAs

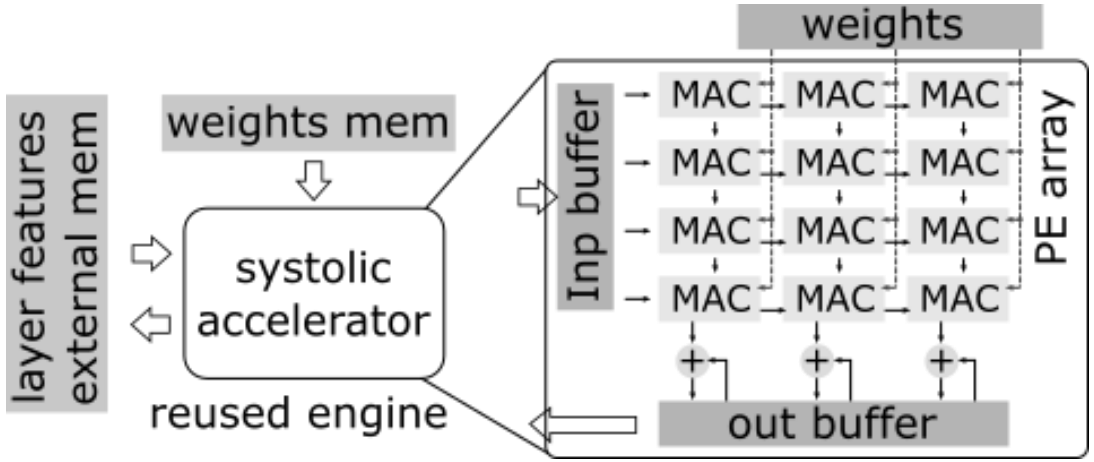
Spatial Pipeline Dataflow

- Dedicated HW for each convolution layer with locally stored weights
- The data are streamed inside the accelerator



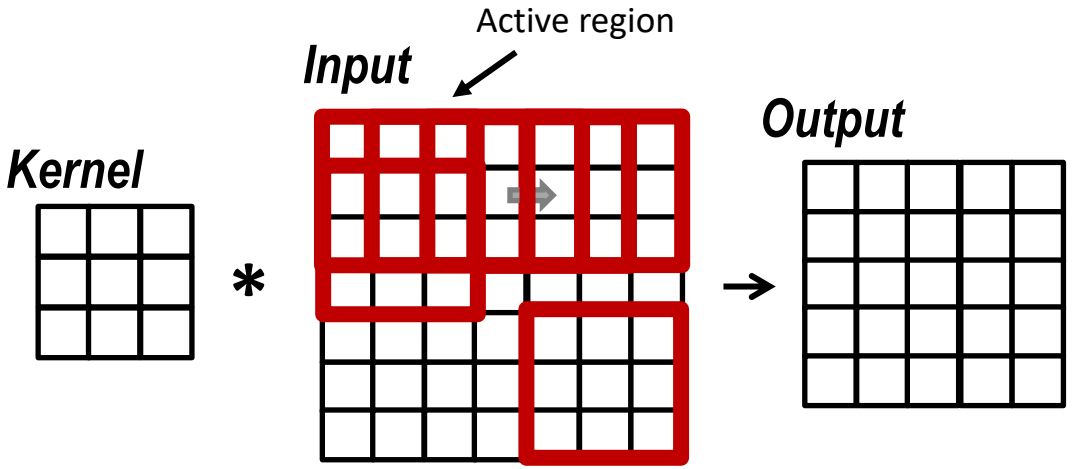
Systolic Array

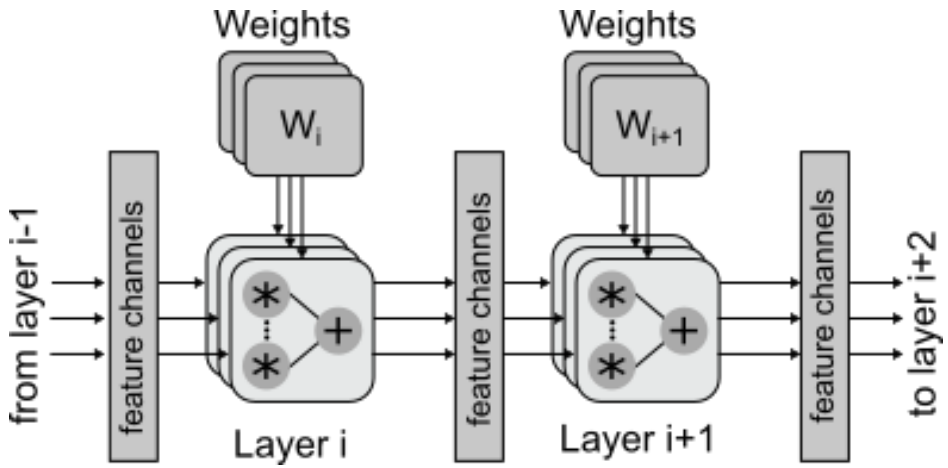
- Convolution mapped to GEMM
- DMAs are used to control the flow of data in the SA
- The Processing Elements perform MAC operations



The convolution operation

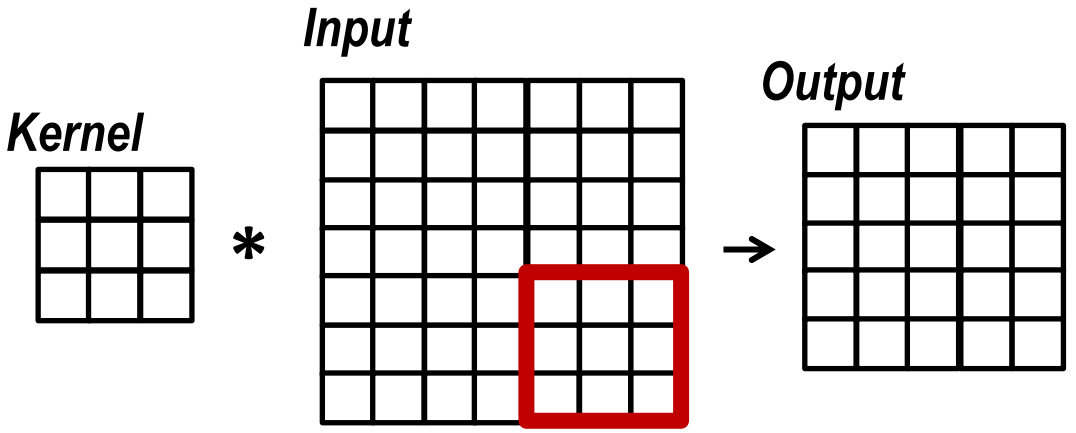
The kernel slides over the input feature map and computes an output feature map using the features in the active region, defined by the size of the kernel





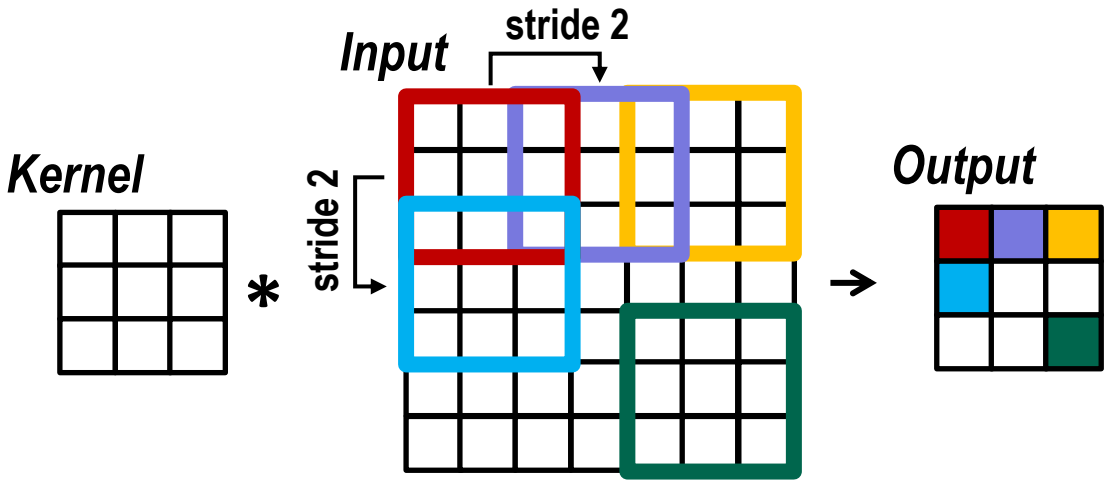
Other spatial forms of convolution

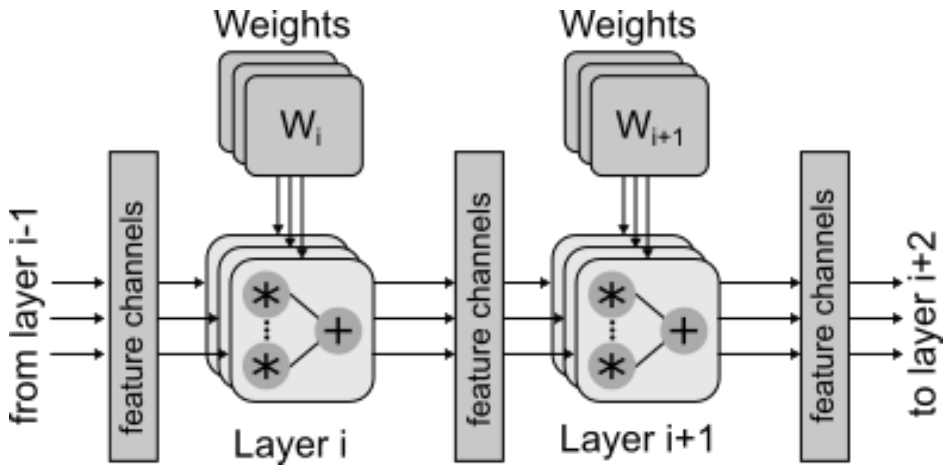
The kernel slides over the input feature map and computes an output feature map using the features in the active region, defined by the size of the kernel



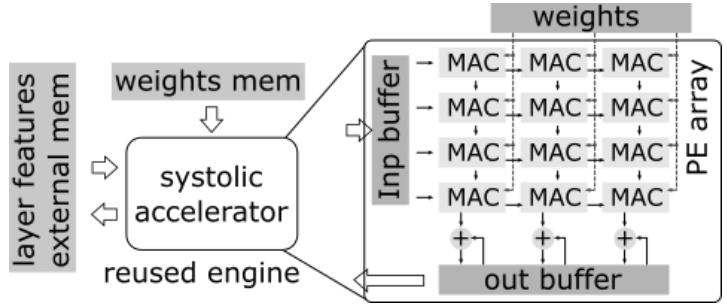
Strided Convolution

The filter moves with a step greater than 1





Convolution as matrix multiplication



	33	34	35	36
17	18	19	20	40
1	2	3	4	24	44
5	6	7	8	28	48
9	10	11	12	32	
13	14	15	16		

indices (i, j)

	(0,0)	(0,1)	(0,2)	(0,3)
(0,0)	(0,1)	(0,2)	(0,3)	(1,3)
(0,0)	(0,1)	(0,2)	(0,3)	(1,3)	(2,3)
(1,0)	(1,1)	(1,2)	(1,3)	(2,3)	(3,3)
(2,0)	(2,1)	(2,2)	(2,3)	(3,3)	
(3,0)	(3,1)	(3,2)	(3,3)		

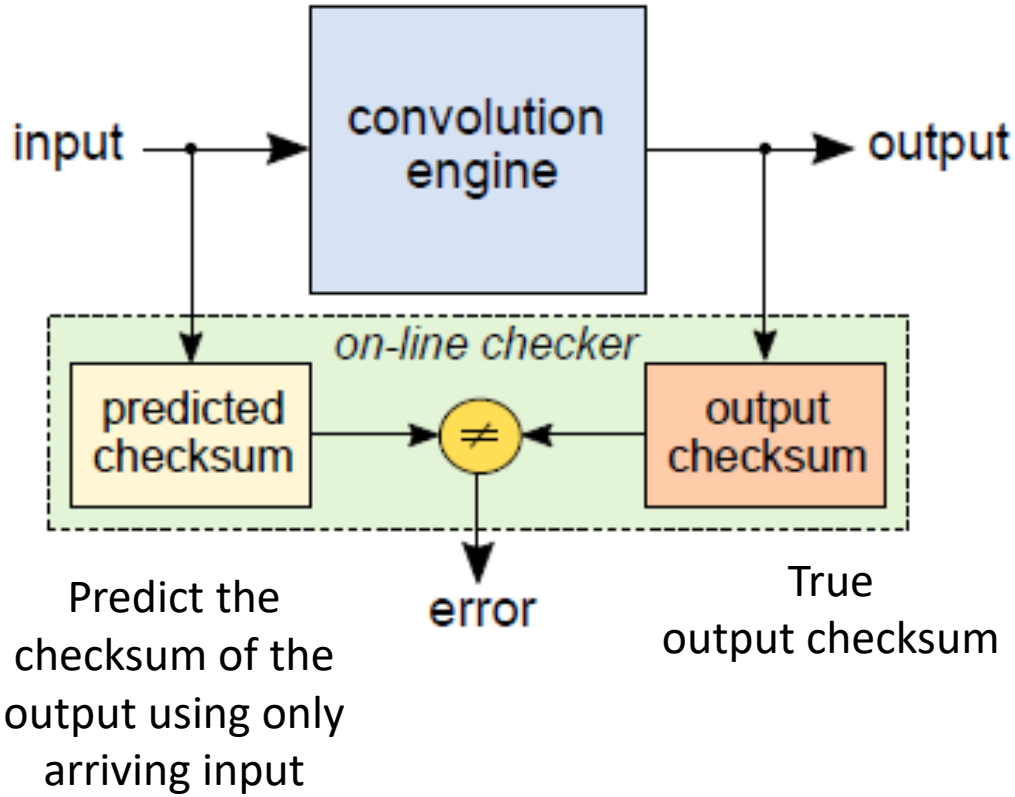
9 (possible sliding window position)

12

1	2	3	5	6	7	9	10	11
2	3	4	6	7	8	10	11	12
5	6	7	9	10	11	13	14	15
6	7	8	10	11	12	14	15	16
17	18	19	21	22	23	25	26	27
18	19	20	22	23	24	26	27	28
21	22	23	25	26	27	29	30	31
22	23	24	26	27	28	30	31	32
33	34	35	37	38	39	41	42	43
34	35	36	38	39	40	42	43	44
37	38	39	41	42	43	45	46	47
38	39	40	42	43	44	46	47	48

Functional safety on random hardware failures

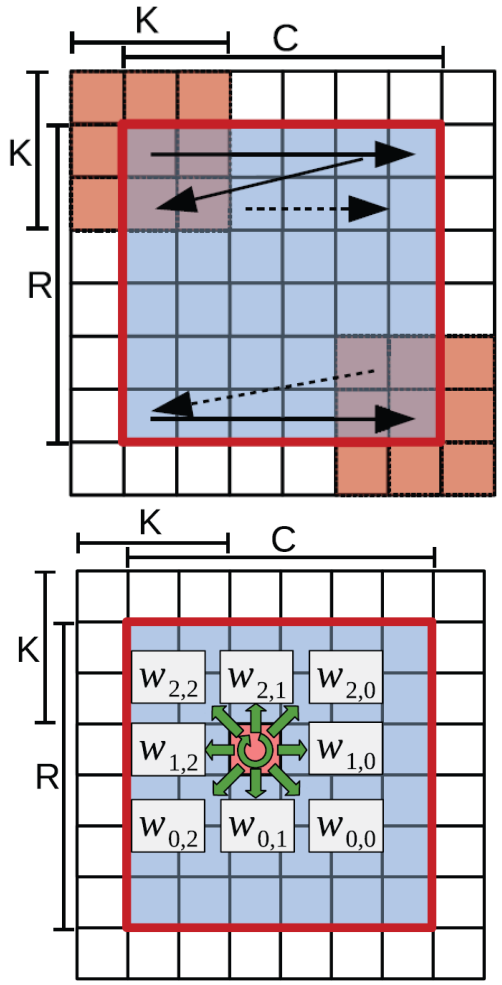
- The prevalence of CNNs in safety-critical systems increases the need for building resilient CNNs
  - Guaranteeing correct computation in the presence of random hardware faults is necessary for safety and possible standards compliance
- Managing random hardware faults requires special hardware modules for fault detection that allows faults to be detected online and rapidly



Convolution checksum

$$\sum_{r=0}^{R}\sum_{c=0}^{C} y_{r,c} = \sum_{r=0}^{R}\sum_{c=0}^{C}\sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\left(x_{r+i,c+j}\cdot w_{i,j}\right)$$

$$= \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\left(\sum_{r=0}^{R}\sum_{c=0}^{C} x_{r+i,c+j}\right)$$



To predict the output sum (checksum) compute multiple sum of inputs and multiply them to the weights they contribute

- Weights correspond to buckets that accumulate corresponding inputs (K^2 buckets in total)

How can we reduce the complexity of checksum prediction?

Baseline output checksum

$$\sum_{r=0}^{R}\sum_{c=0}^{C} y_{r,c} = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\left(\sum_{r=0}^{R}\sum_{c=0}^{C} x_{r+i,c+j}\right)$$

First we prove **a new convolution invariance condition** that decouples weights from input

$$\sum_{r=0}^{R}\sum_{c=0}^{C} y_{r,c} = \left(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\right)\left(\sum_{r}^{R}\sum_{c}^{C} x_{r,c}\right)$$

Sum of filter coefficients

Sum of input

The sum of convolution's output is the product of the sum of weights and the sum of input

Convolution invariance example

$$w = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad x = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \end{bmatrix} \qquad y = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 4 & \mathbf{10} & \mathbf{14} & 12 \\ 4 & \mathbf{10} & \mathbf{14} & 12 \\ 3 & 7 & 10 & 8 \end{bmatrix}$$

$$\sum_{r=0}^{R}\sum_{c=0}^{C} y_{r,c} = \left(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\right)\left(\sum_{r}^{R}\sum_{c}^{C} x_{r,c}\right)$$

10 x **12**

120

Convolution invariance extension

$$w = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad x = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \end{bmatrix} \qquad y = \begin{bmatrix} 1 & 3 & 4 & 4 \\ 4 & \mathbf{10} & \mathbf{14} & 12 \\ 4 & \mathbf{10} & \mathbf{14} & 12 \\ 3 & 7 & 10 & 8 \end{bmatrix}$$

True useful output

$$\sum_{r=0}^{R}\sum_{c=0}^{C}(y_{true}+y_{extra}) = \left(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\right)\left(\sum_{r}^{R}\sum_{c}^{C} x_{r,c}\right)$$

120 = 48 + 72

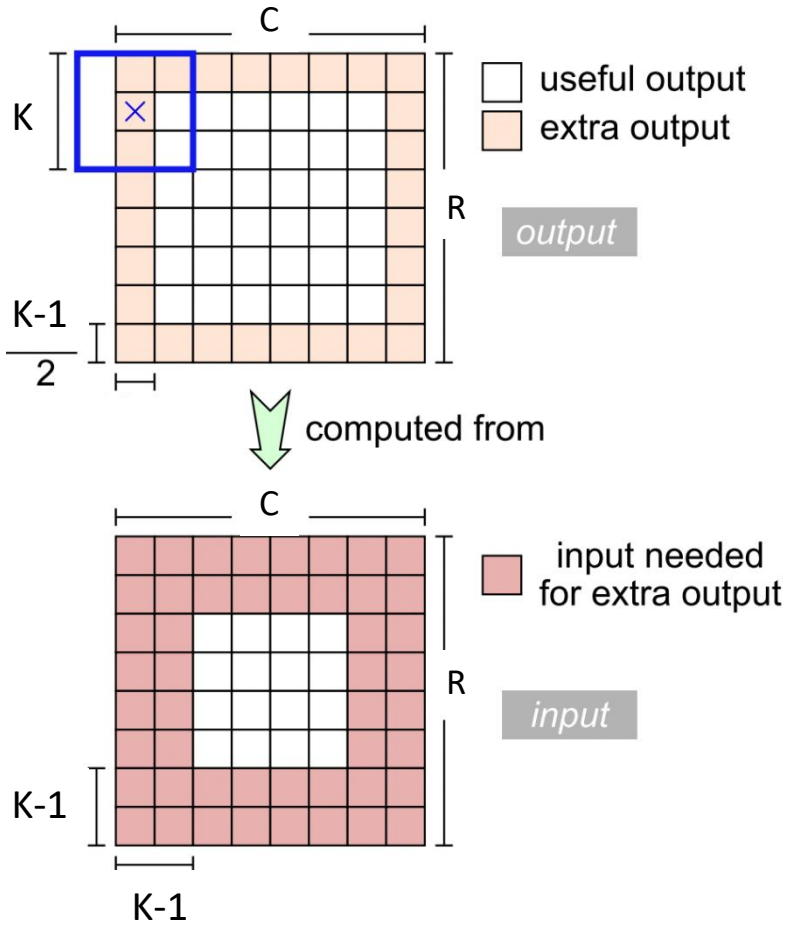
10 x **12**

How to use this convolution invariance extension?

$$\sum_{r=0}^{R}\sum_{c=0}^{C}(y_{true}+y_{extra}) = \left(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\right)\left(\sum_{r}^{R}\sum_{c}^{C} x_{ALL}\right)$$

- Compute the sum of true useful output (y_{true}) as it comes out of the accelerator
- Compute the sum of all input (x_{ALL}) as it enters the accelerator

$$\sum_{r=0}^{R}\sum_{c=0}^{C} y_{extra} = \left(\sum_{i=0}^{K-1}\sum_{j=0}^{K-1} w_{i,j}\right)\left(\sum_{r}^{R}\sum_{c}^{C} \boldsymbol{x_{periphery}}\right)$$

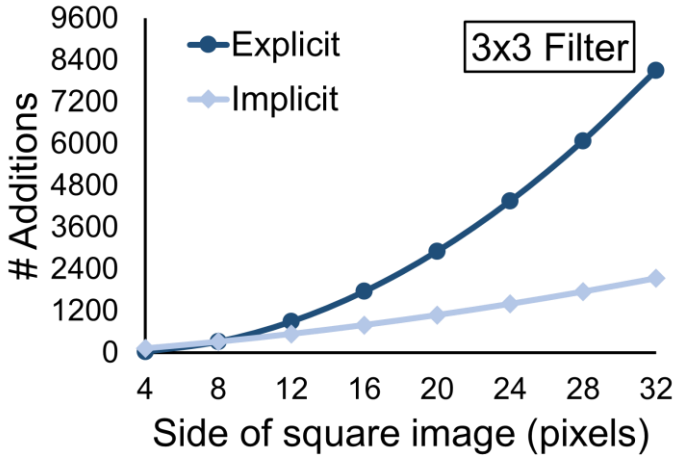


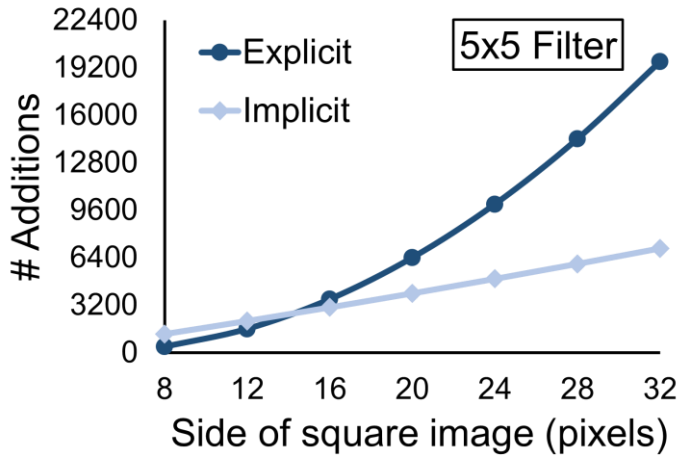
- Compute (predict) y_{extra} using only peripheral input (far less accumulations needed)

Evaluation of ConvGuard

ConvGuard uses a completely different method to predict the checksum

- Less operations are performed than explicit checksum prediction.
- No storage requirement for intermediate results.





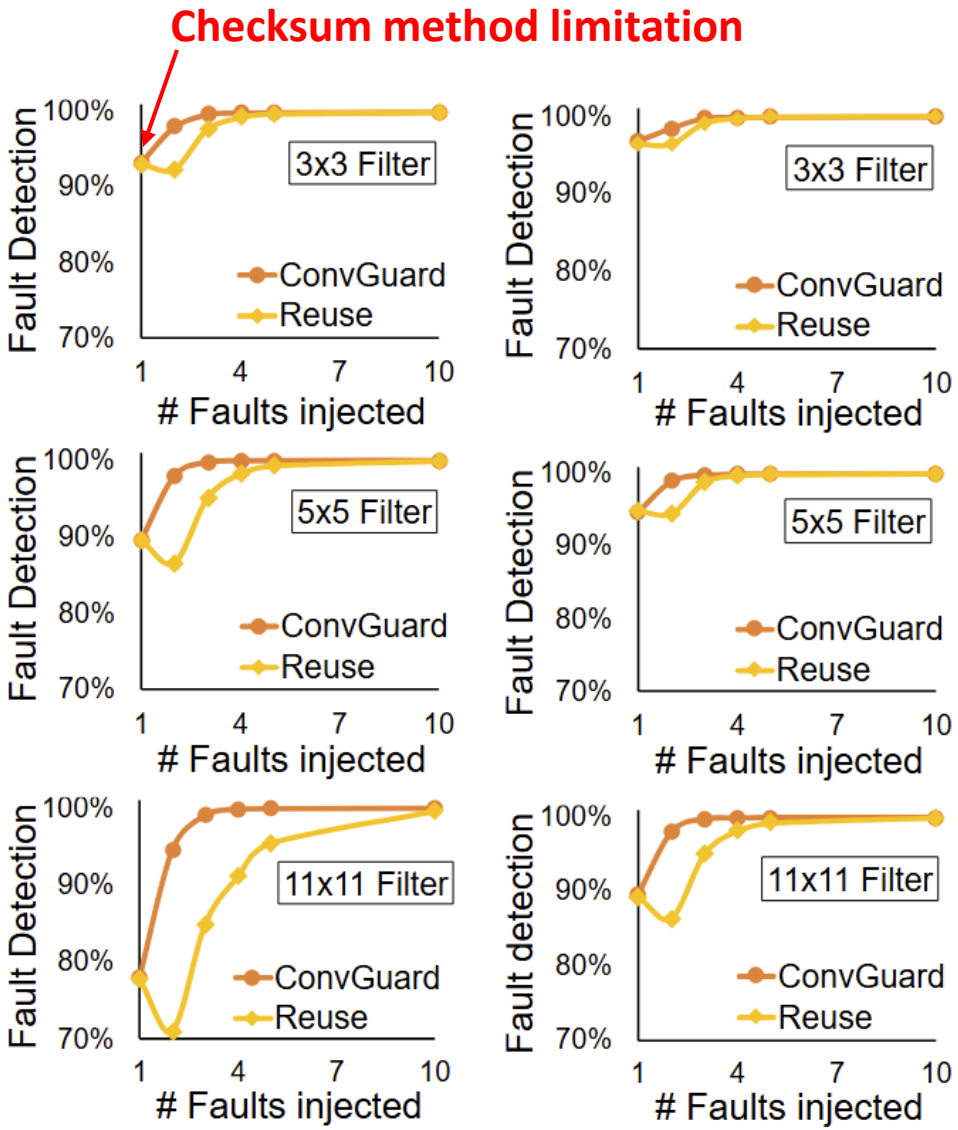
The implementation of the checker makes it less susceptible to faults.

ConvGuard's fault detection efficiency

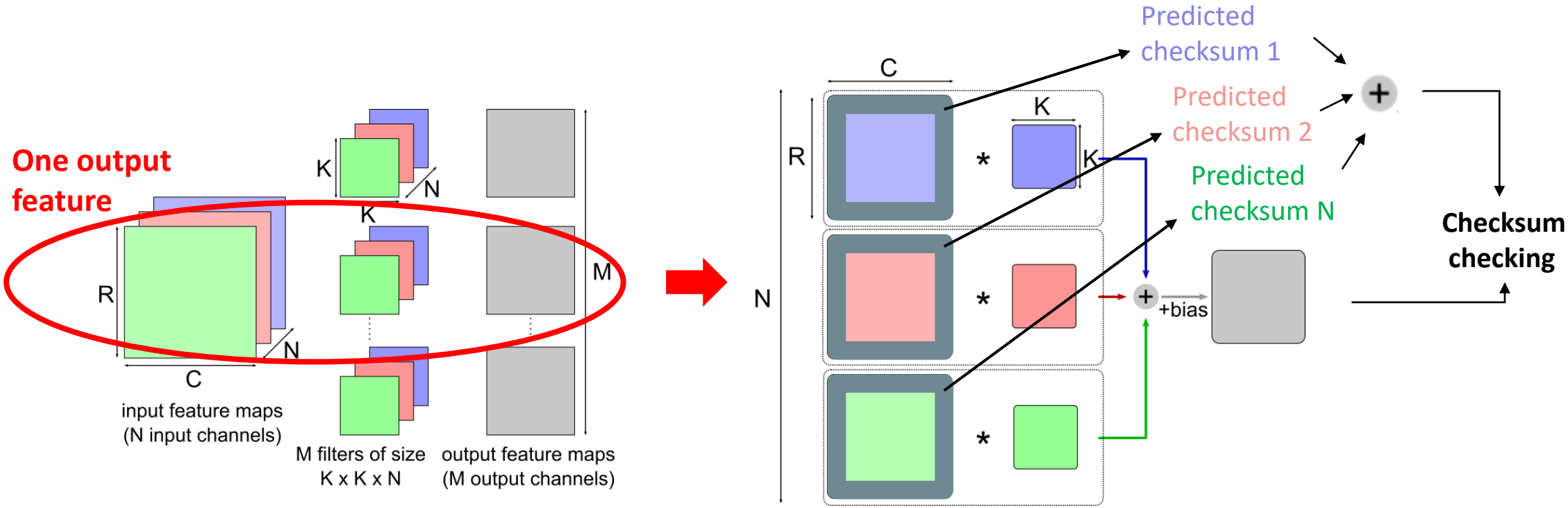
The efficiency of ConvGuard was quantified through extended simulation.

- Faults that affect the engine do not affect the checker
- The minimum number of registers reduces the chance for potential bit flips in the checker

Faults Injected	Image	Fault Categories			
		Detected	Silent	FP	FN
2	14 × 14	91.36%	3.98%	4.62%	0.01%
	28 × 28	95.95%	2.03%	2.01%	0.01%
	56 × 56	97.96%	1.45%	0.59%	0.00%
	112 × 112	98.58%	1.17%	0.25%	0.00%
4	14 × 14	99.54%	0.23%	0.23%	0.00%
	28 × 28	99.85%	0.12%	0.02%	0.00%
	56 × 56	99.93%	0.07%	0.00%	0.00%
	112 × 112	99.94%	0.06%	0.00%	0.00%



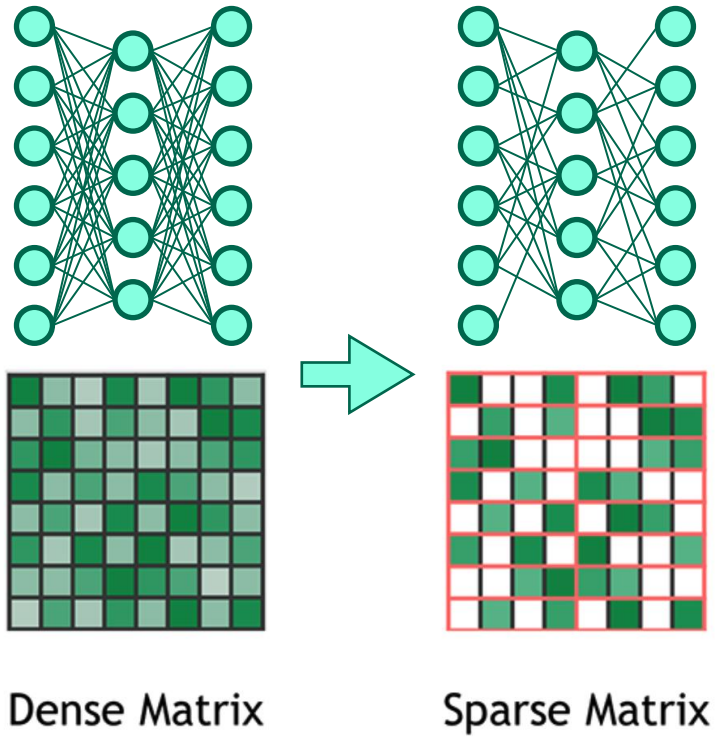
Generalize ConvGuard for CNN layers



The prediction of the checksum is performed in parallel to each input channel and after all channels have been computed the checking is performed.

Weight pruning for model-size reduction

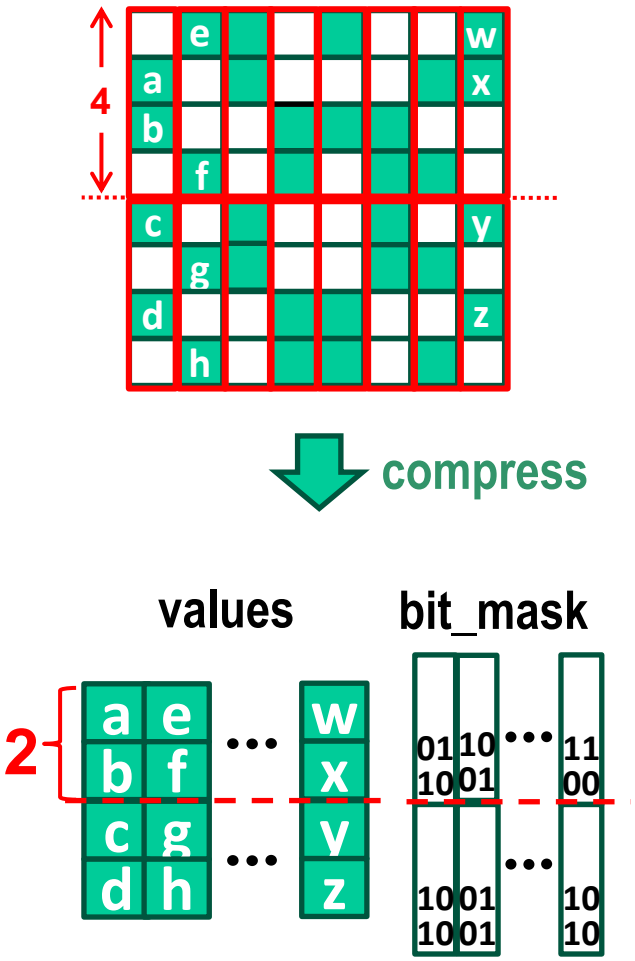
- A simple way to reduce neural network's size is **weight pruning**
  - **The less important weights are removed**
  - **Easier to fit in edge devices with limited resources**
- Pruning results in sparse weights
- Reduce the number of required computations
  - **Significantly improve inference speed**
  - **Allow for l**ower computation cost during training



- **To further reduce model size → combined with other compression techniques**

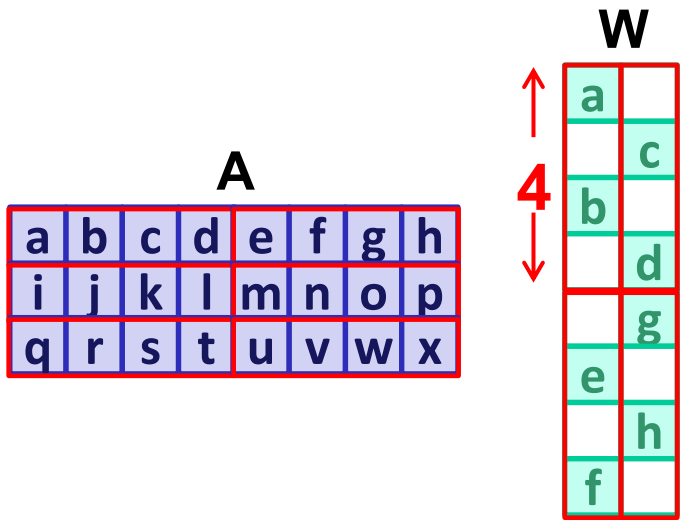
Structured Block Sparsity

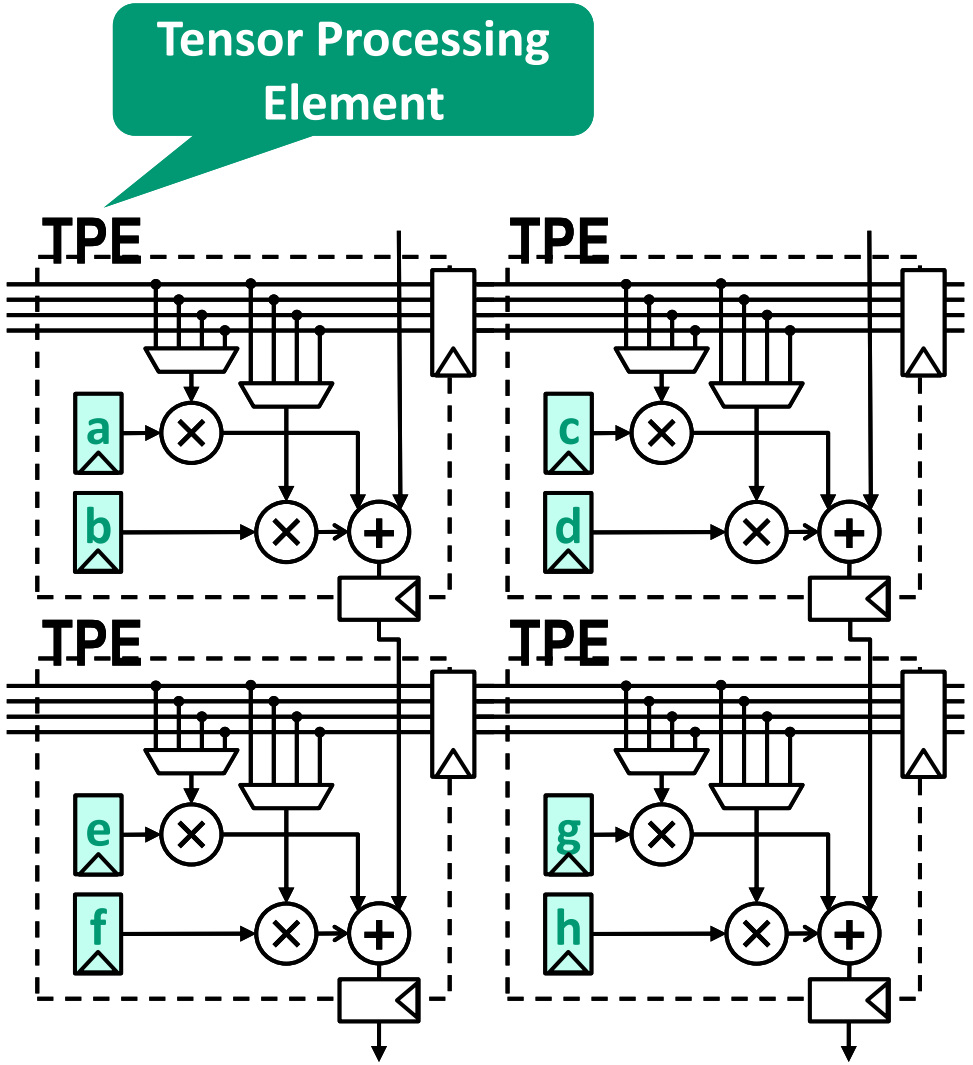
- N:M structured block sparsity
  - Up to N non-zero elements out of M
  - M is small in practice
- Indexing in each block → bit masks
- Sparsification needs retraining/fine tuning for acceptable accuracy



Sparse Systolic Tensor Arrays

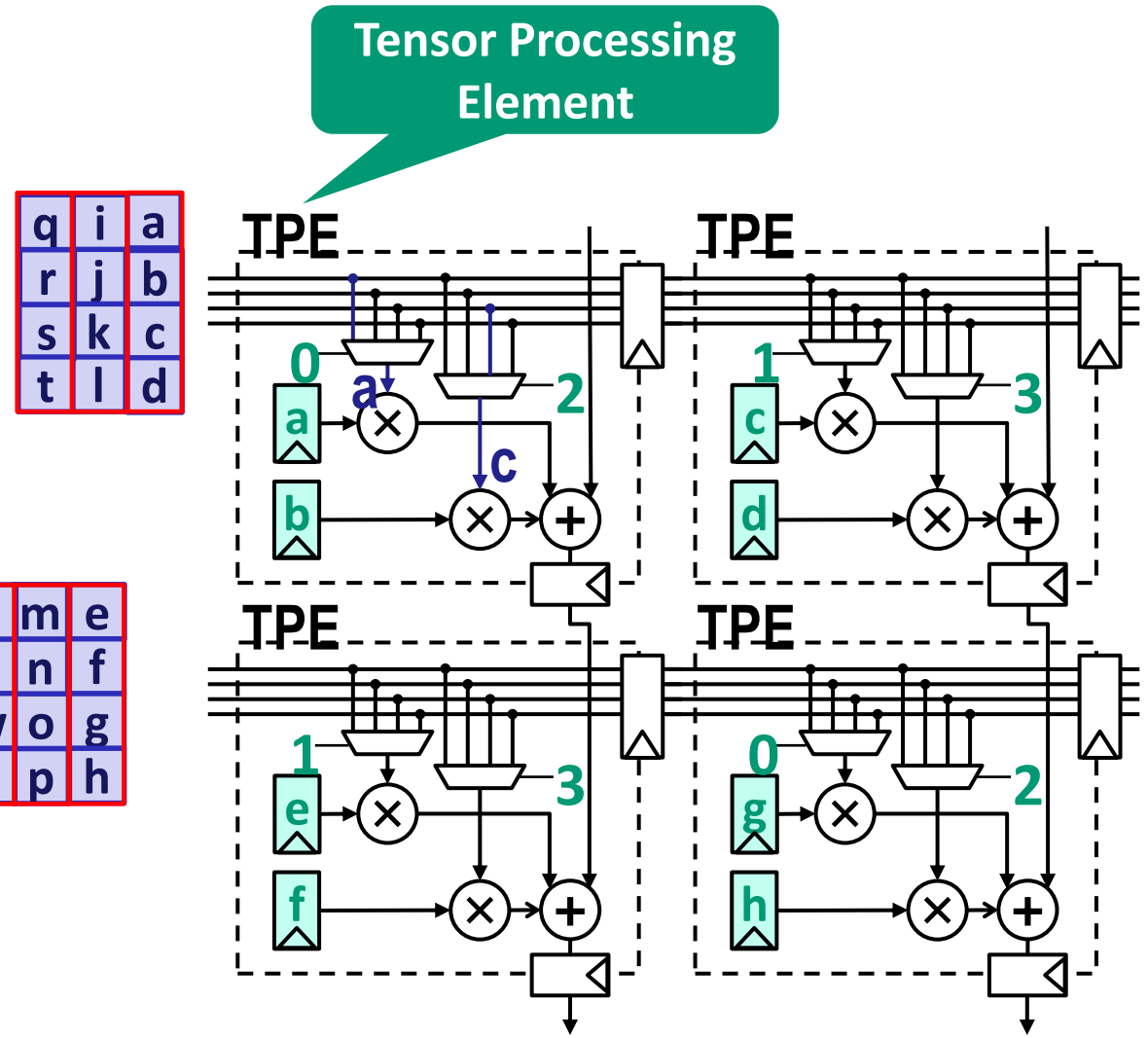
TPE consists of multiple MAC units instead of scalar PEs





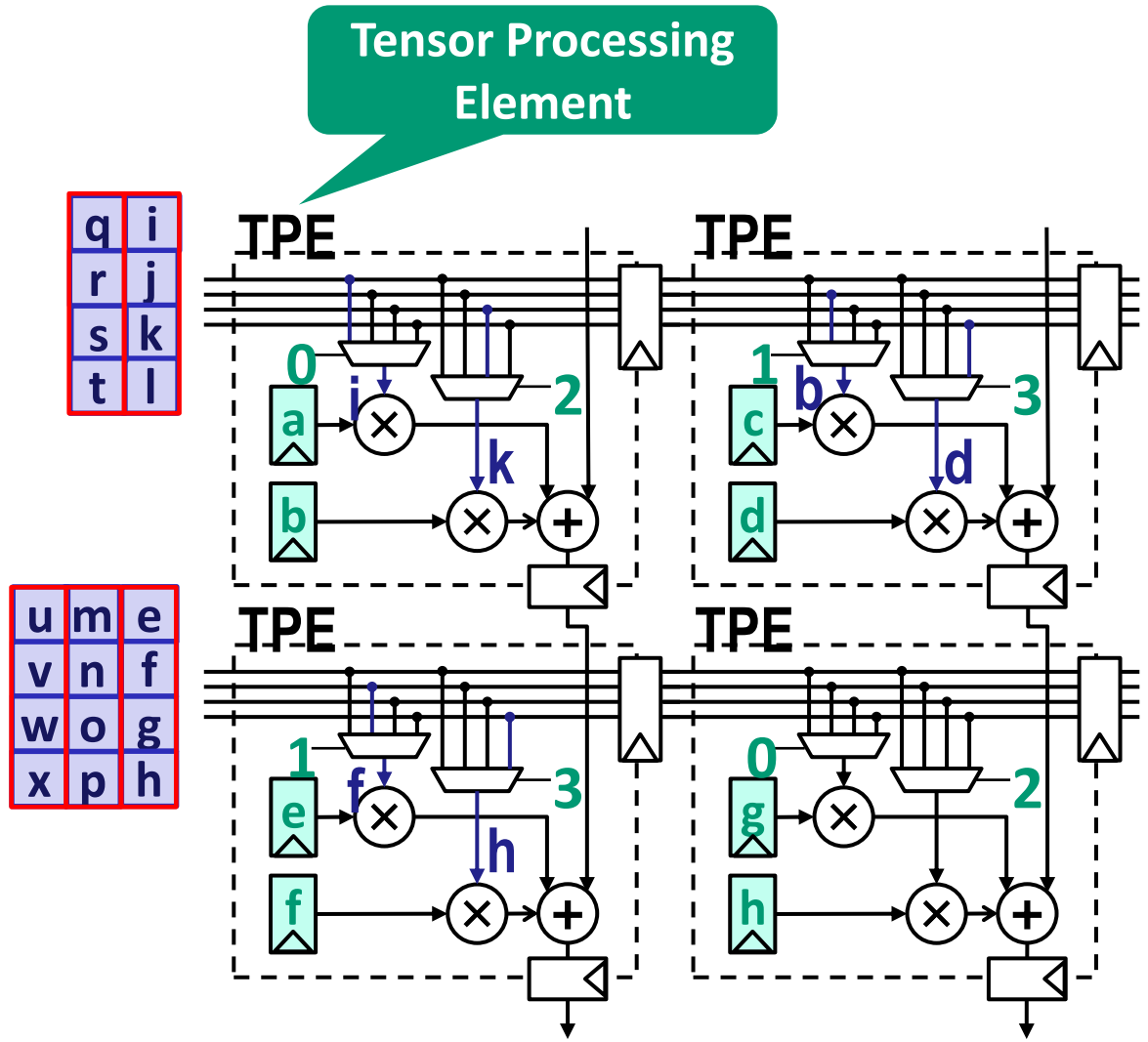
Sparse Systolic Tensor Arrays

- A block of inputs flow through each row of the systolic
- Index of each weight selects the corresponding input to be multiplied
- Could be configured for either 2:4 or 1:4 block sparsity



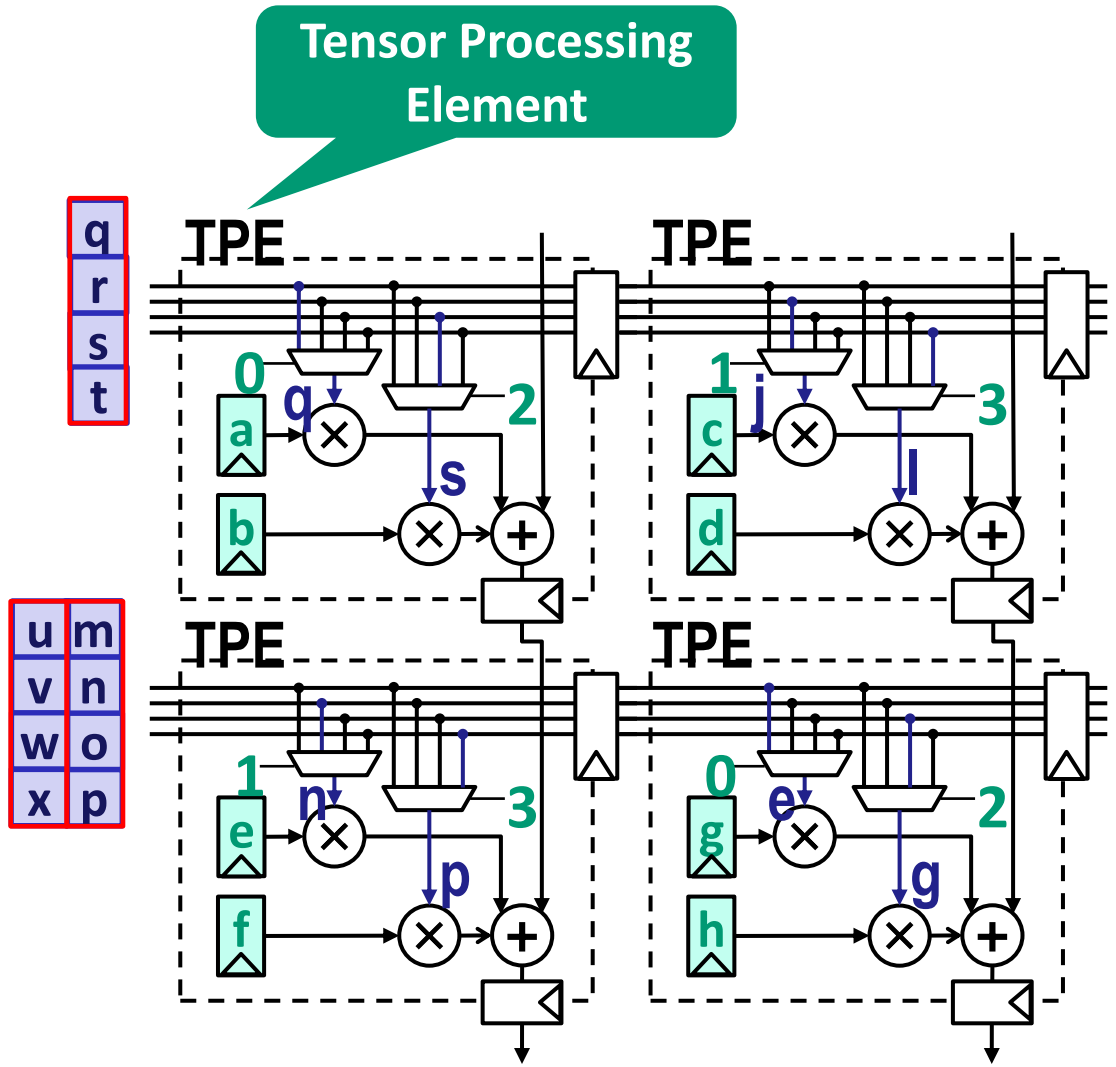
Sparse Systolic Tensor Arrays

- A block of inputs flow through each row of the systolic
- Index of each weight selects the corresponding input to be multiplied
- Could be configured for either 2:4 or 1:4 block sparsity



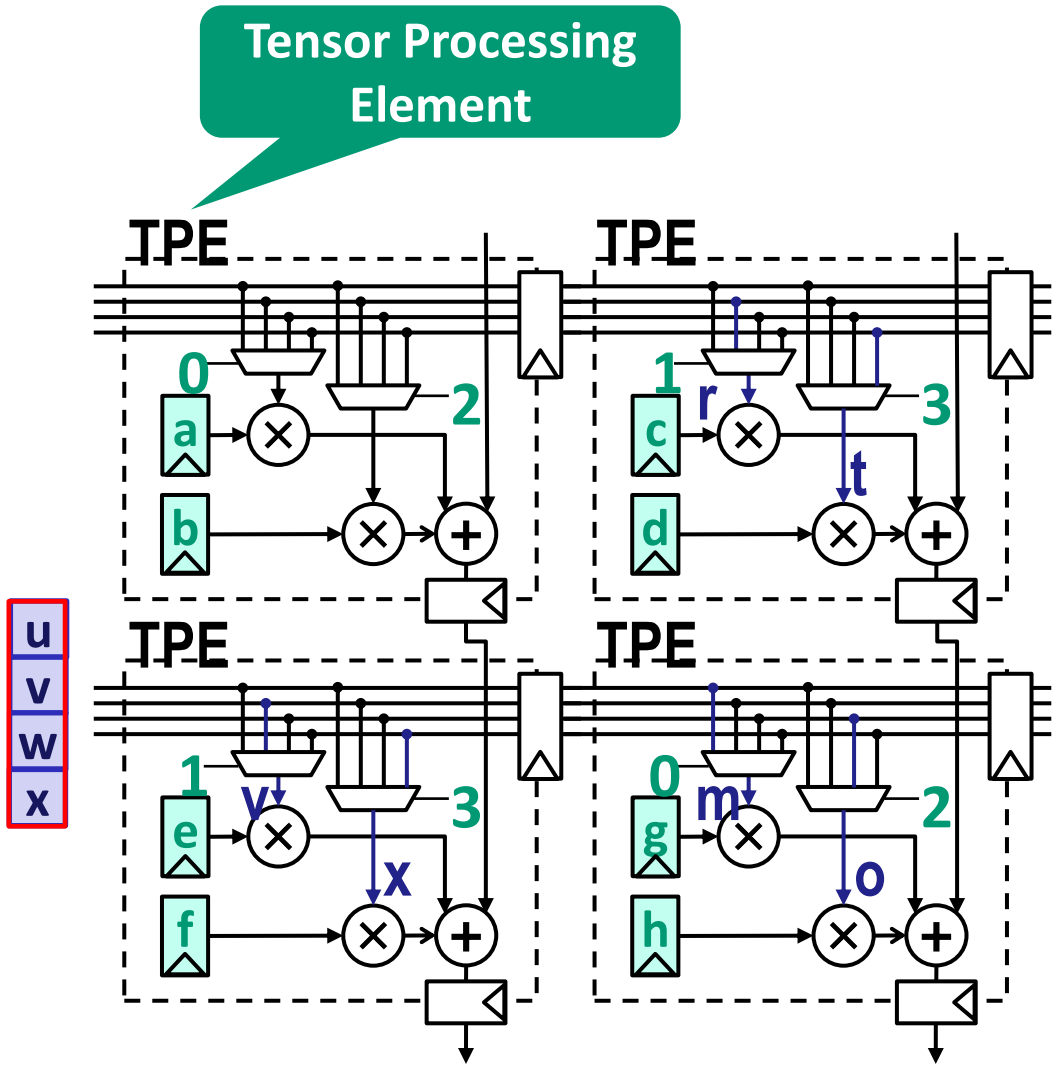
Sparse Systolic Tensor Arrays

- A block of inputs flow through each row of the systolic
- Index of each weight selects the corresponding input to be multiplied
- Could be configured for either 2:4 or 1:4 block sparsity



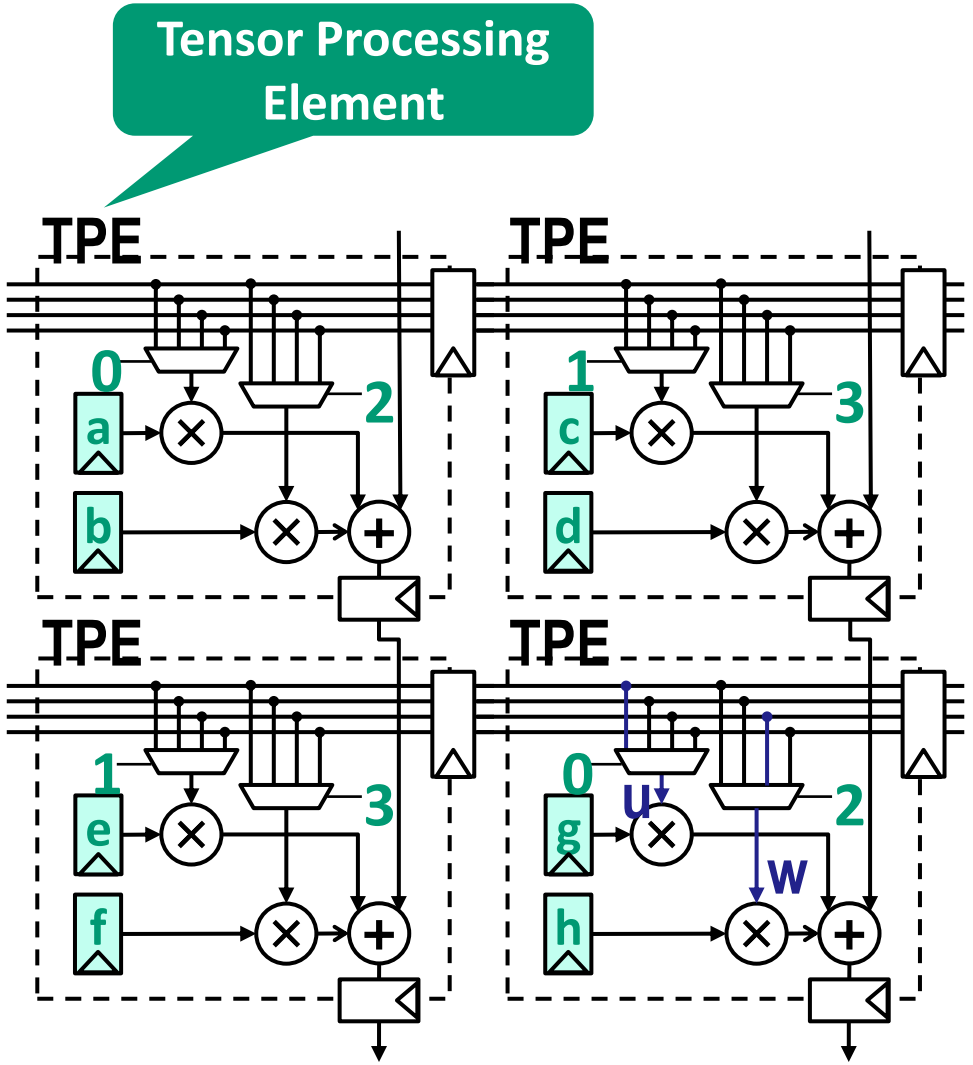
Sparse Systolic Tensor Arrays

- A block of inputs flow through each row of the systolic
- Index of each weight selects the corresponding input to be multiplied
- Could be configured for either 2:4 or 1:4 block sparsity

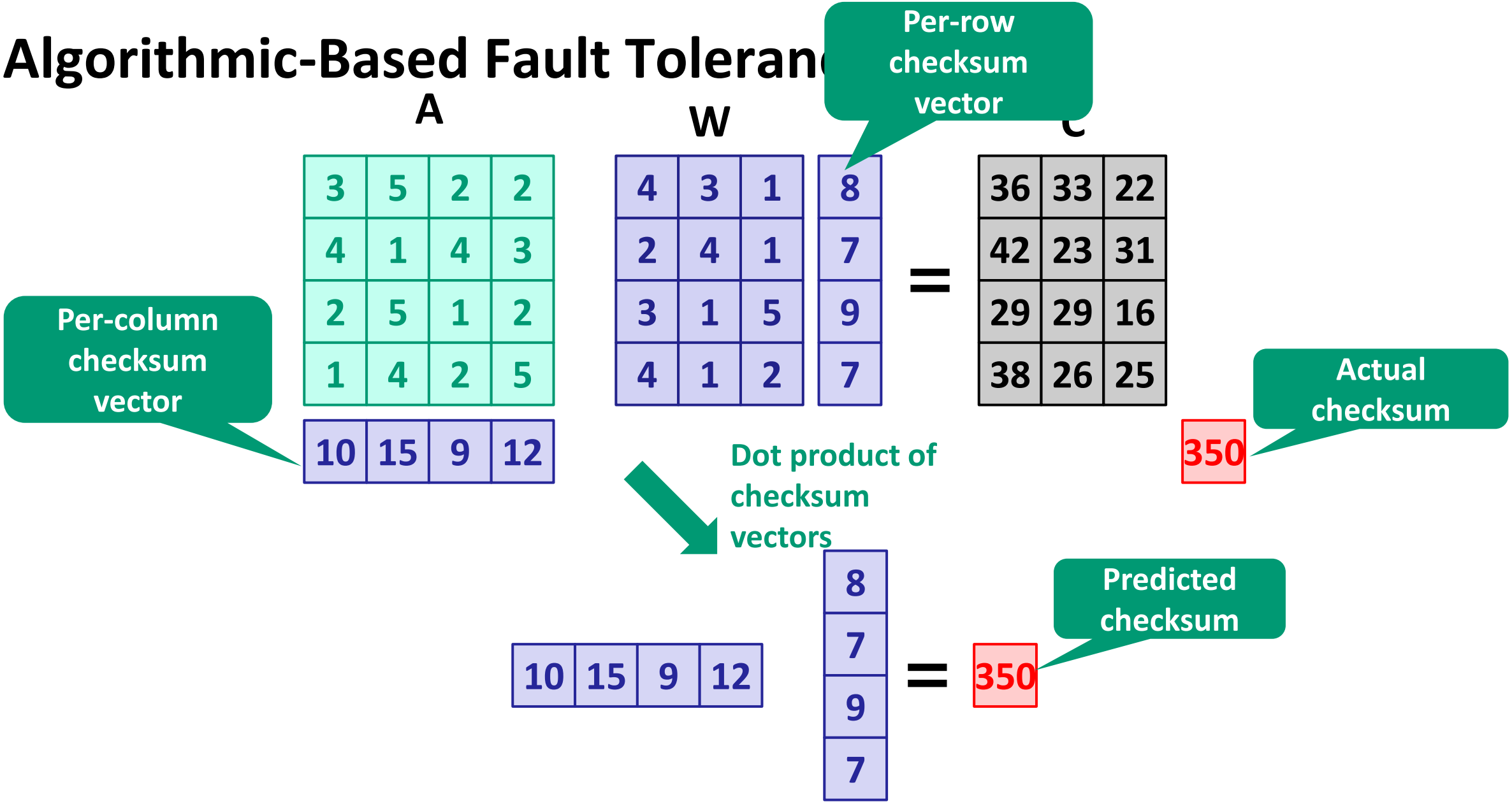


Sparse Systolic Tensor Arrays

- A block of inputs flow through each row of the systolic
- Index of each weight selects the corresponding input to be multiplied
- Could be configured for either 2:4 or 1:4 block sparsity



Algorithmic-Based Fault Toleran



Equivalent Algorithmic-Based Fault Tolerance

Reuse matrix W → Reuse weight stationary hardware

A

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5
10	15	9	12

W

4	3	1
2	4	1
3	1	5
4	1	2

=

36	33	22
42	23	31
29	29	16
38	26	25

To predict checksum multiply per-column checksum vector with W

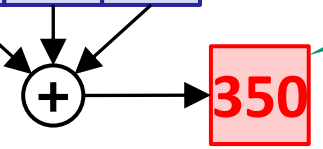
Per-column checksum vector

10	15	9	12

4	3	1
2	4	1
3	1	5
4	1	2

=

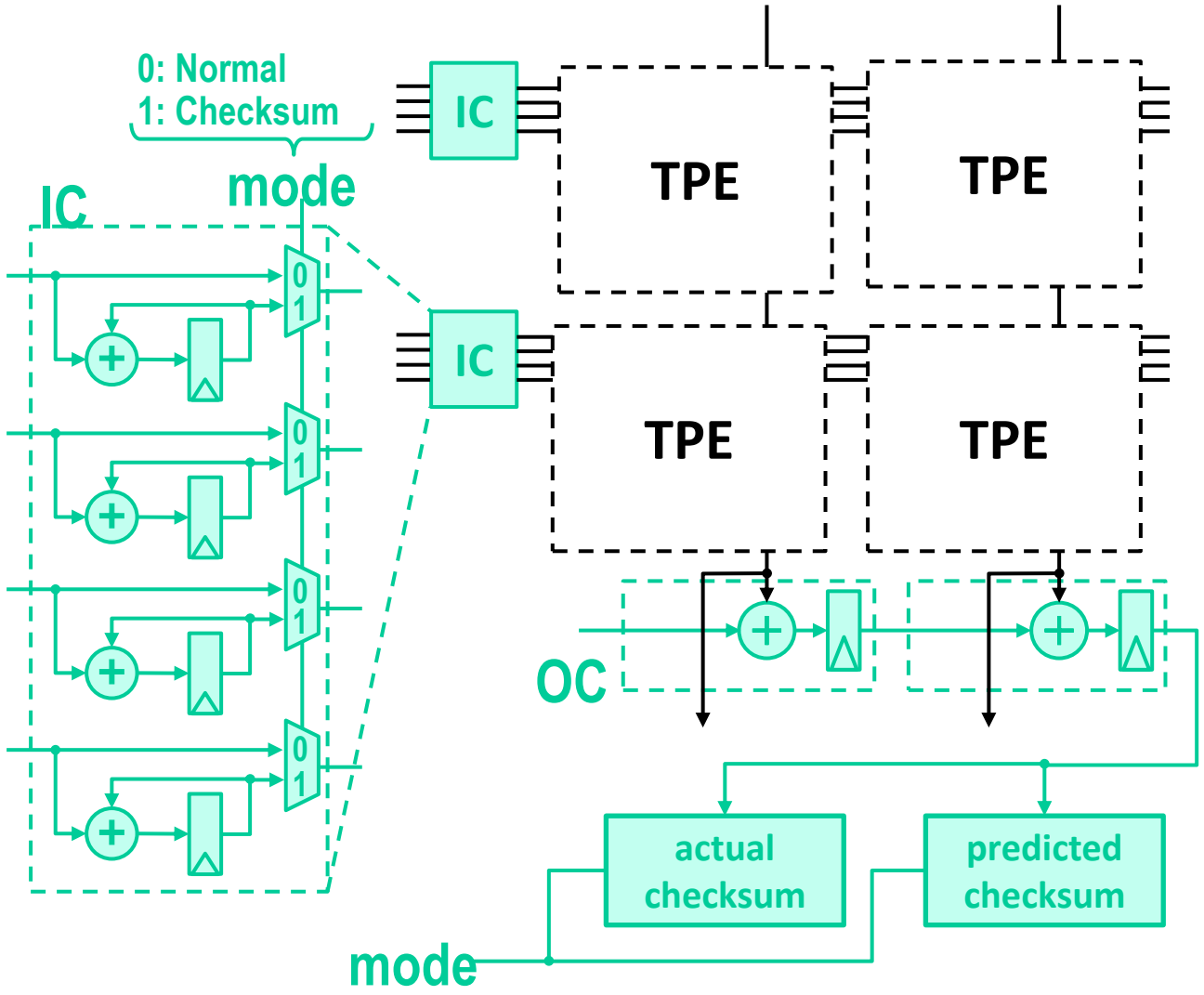
145	111	94



Predicted checksum

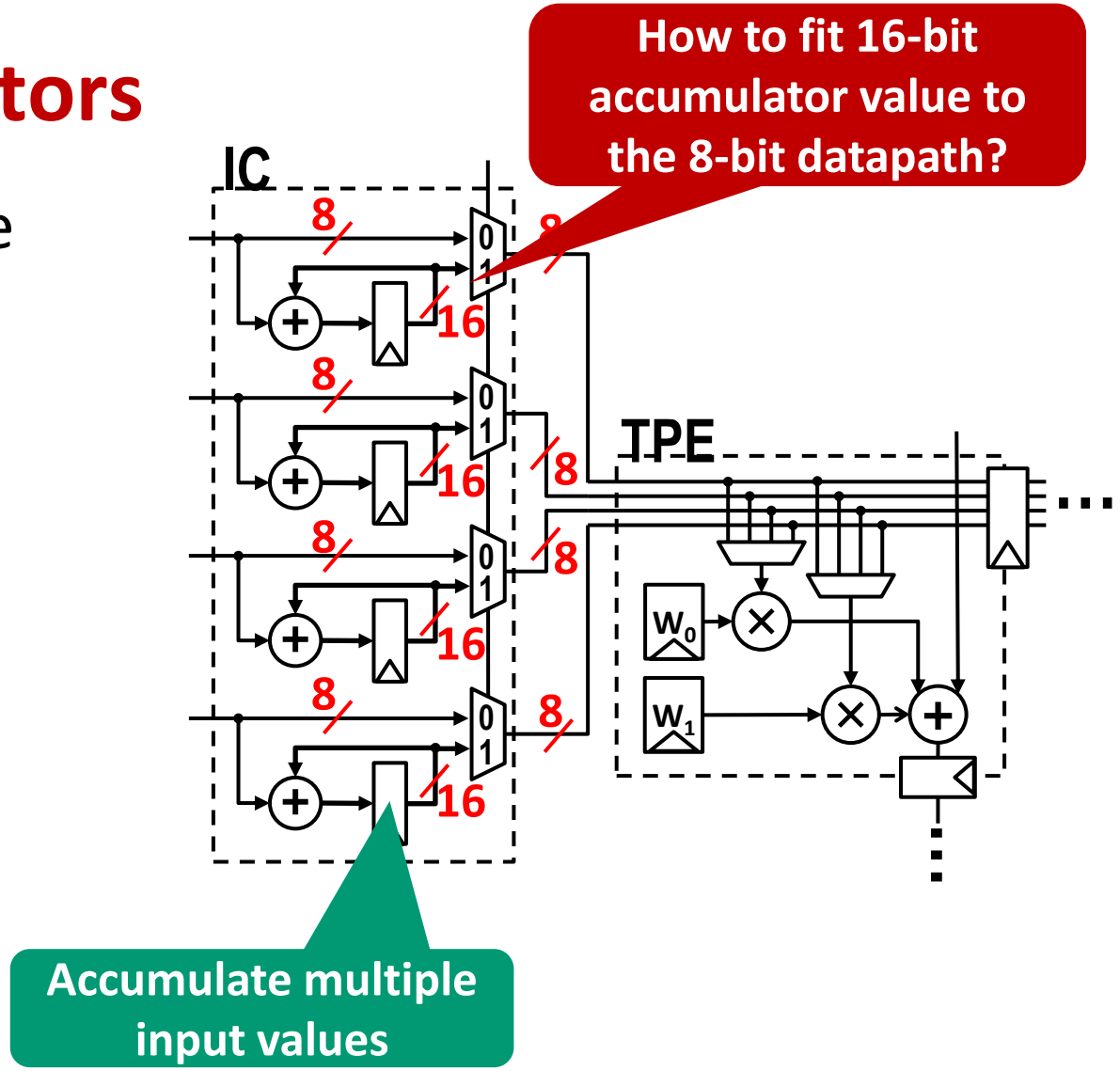
Online ABFT on Sparse Systolic Tensor Arrays

- Add IC and OC modules in the periphery of the sparse tensor array to implement ABFT
- Per-column checksum vector accumulated in ICs and then fed to the SA
- Predicted checksum computed in OC modules



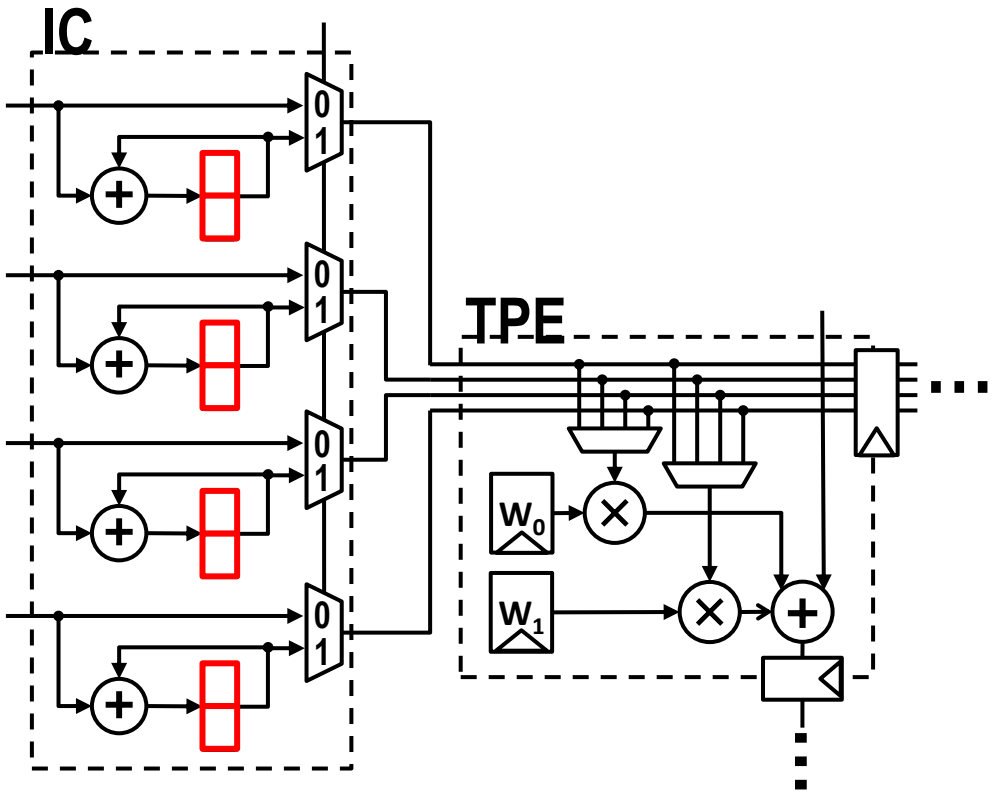
Increasing Width of Accumulators

- IC accumulators accumulate multiple rows of *A* →larger than input width
- TPEs follow input values' bit-width
- Per-column checksum values accumulated in ICs could not fit in utilized MAC units



Increasing Width of Accumulators

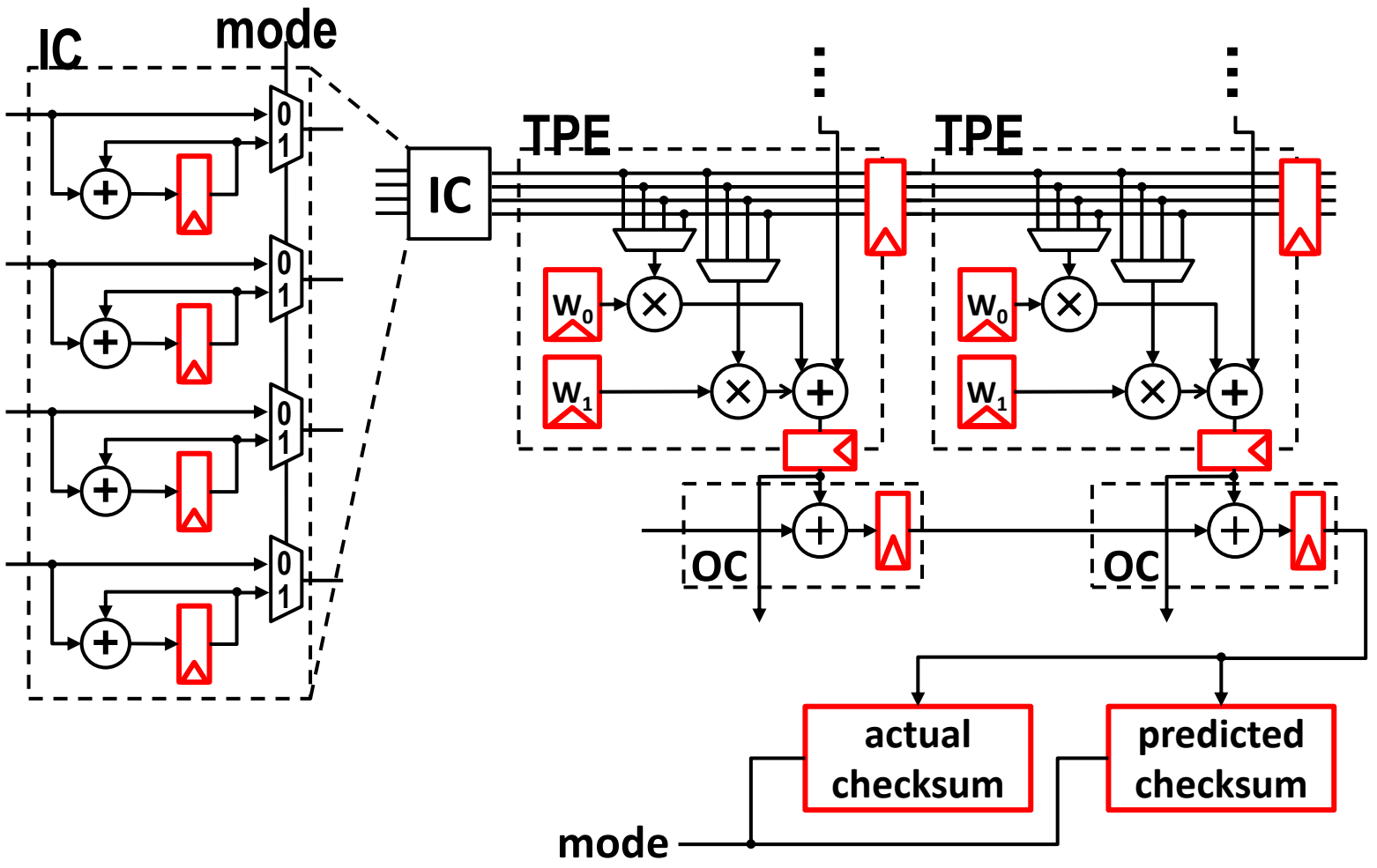
- IC accumulators accumulate multiple rows of *A* →larger than input width
- TPEs follow input values' bit-width
- Per-column checksum values accumulated in ICs could not fit in utilized MAC units



Breakdown per-column checksum values to digits → equal size with input values

Experimental Setup

- Assume that faults may occur in storage elements
- Faults may occur in both sparse tensor array and checking logic
- Probability of error is proportional to the number and size of storage elements



Fault Detection

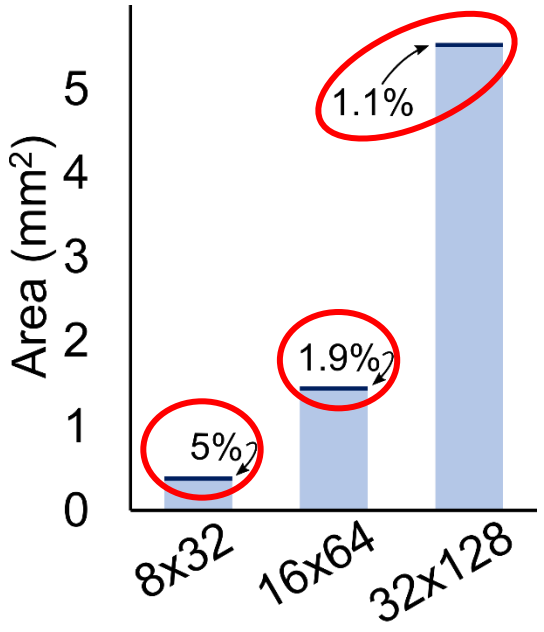
# of Faults	1 fault		1 – 5 faults	
Block Sparsity	**2:4**	**1:4**	**2:4**	**1:4**
Detected	81.11%	74.15%	95.61%	93.34%
Silent	12.46%	20.75%	2.15%	4.95%
False Positive	2.36%	2.58%	0.58%	0.69%
False Negative	4.07%	2.51%	1.65%	1.02%

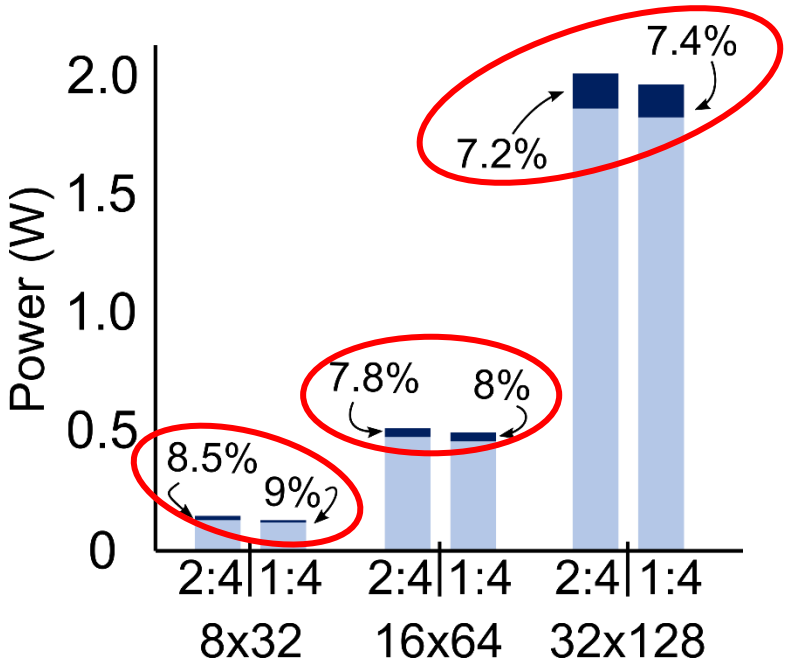
- Performed 15000 independent fault-injection campaigns
- Four possible scenarios of fault campaigns:
  1. Detected: An error occurred, and the checker detect it
  2. Silent: The fault was masked although the checker remained fault free
  3. False Positive: The checker flagged a fault detection, but no fault occurred
  4. False Negative: An error occurred, but the checker did not detect it

Hardware Overhead

- Inputs and weights are 8-bit quantized integers
- Area overhead reduced with the size of the sparse tensor array
- Sparser weights → lower power consumption in each TPE
- Power consumption in ICs and OCs independent of sparsity

Greater power overhead for 1:4 block sparsity



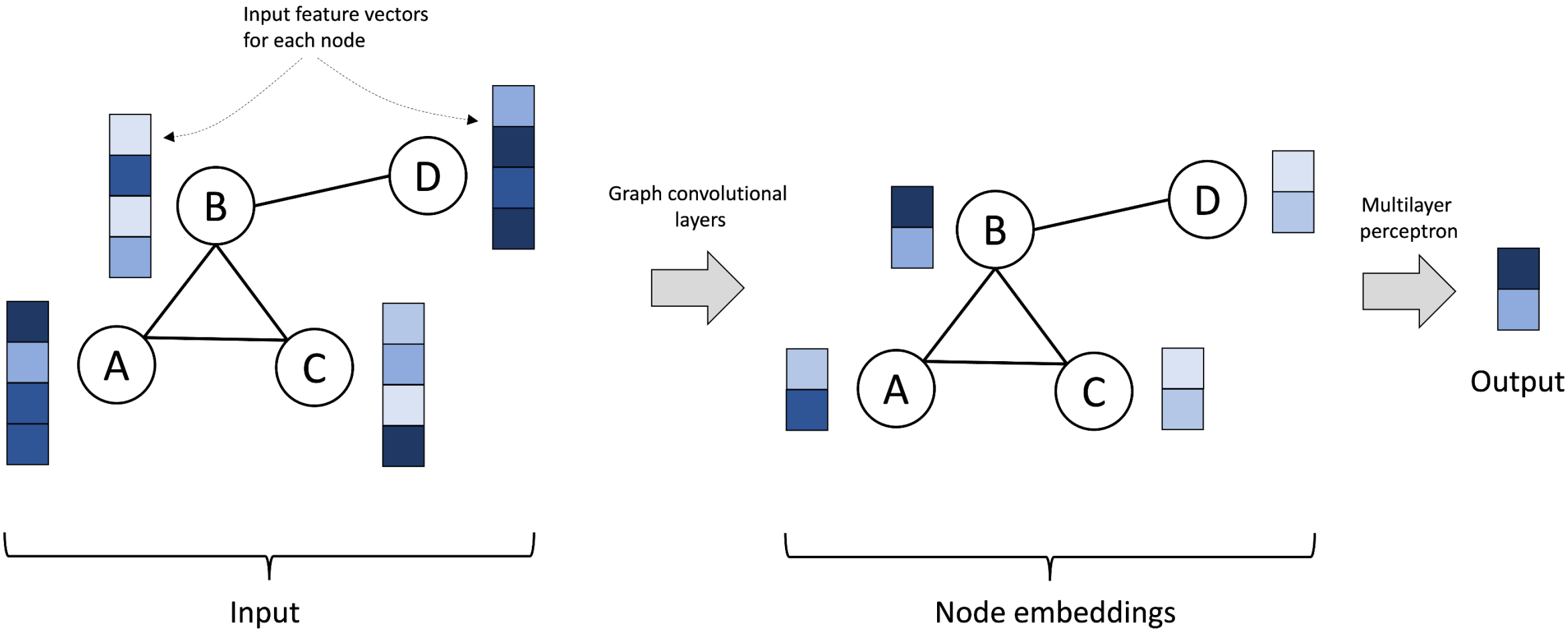


Part 2

GRAPH CONVOLUTIONAL NETWORKS

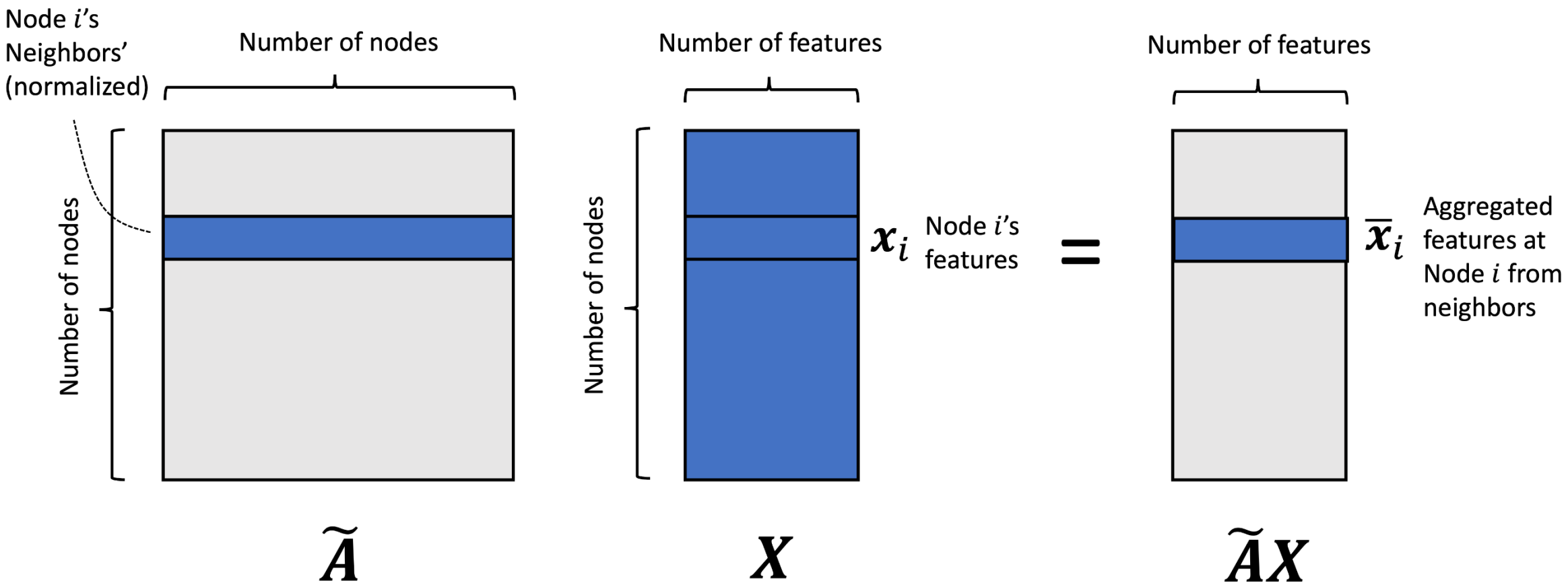
Graph Convolutional Networks

Graph convolutional layer pools the vectors from each node's neighbors



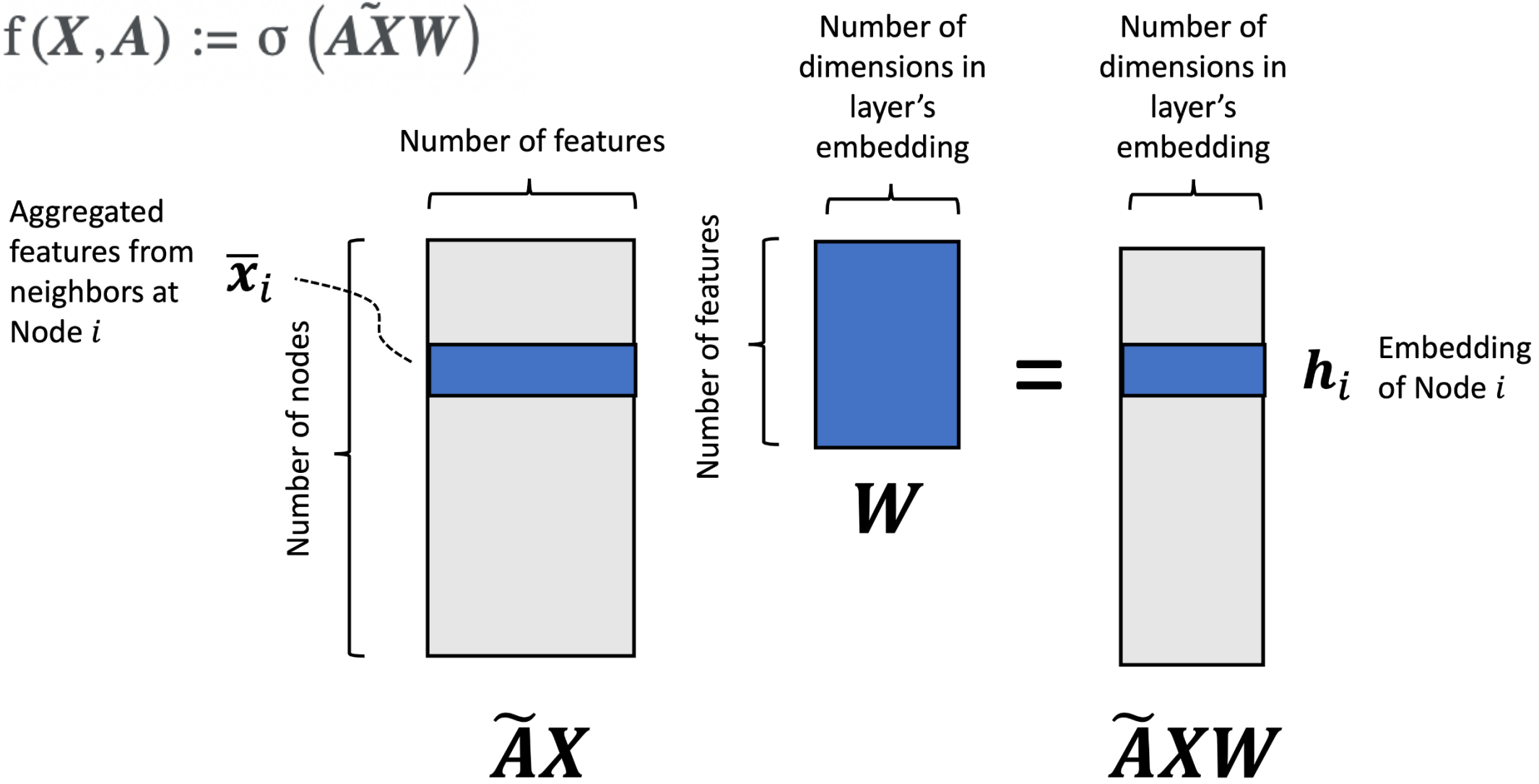
GCNs computation: Aggregation

$$f(X,A) := \sigma\left(A\tilde{X}W\right)$$



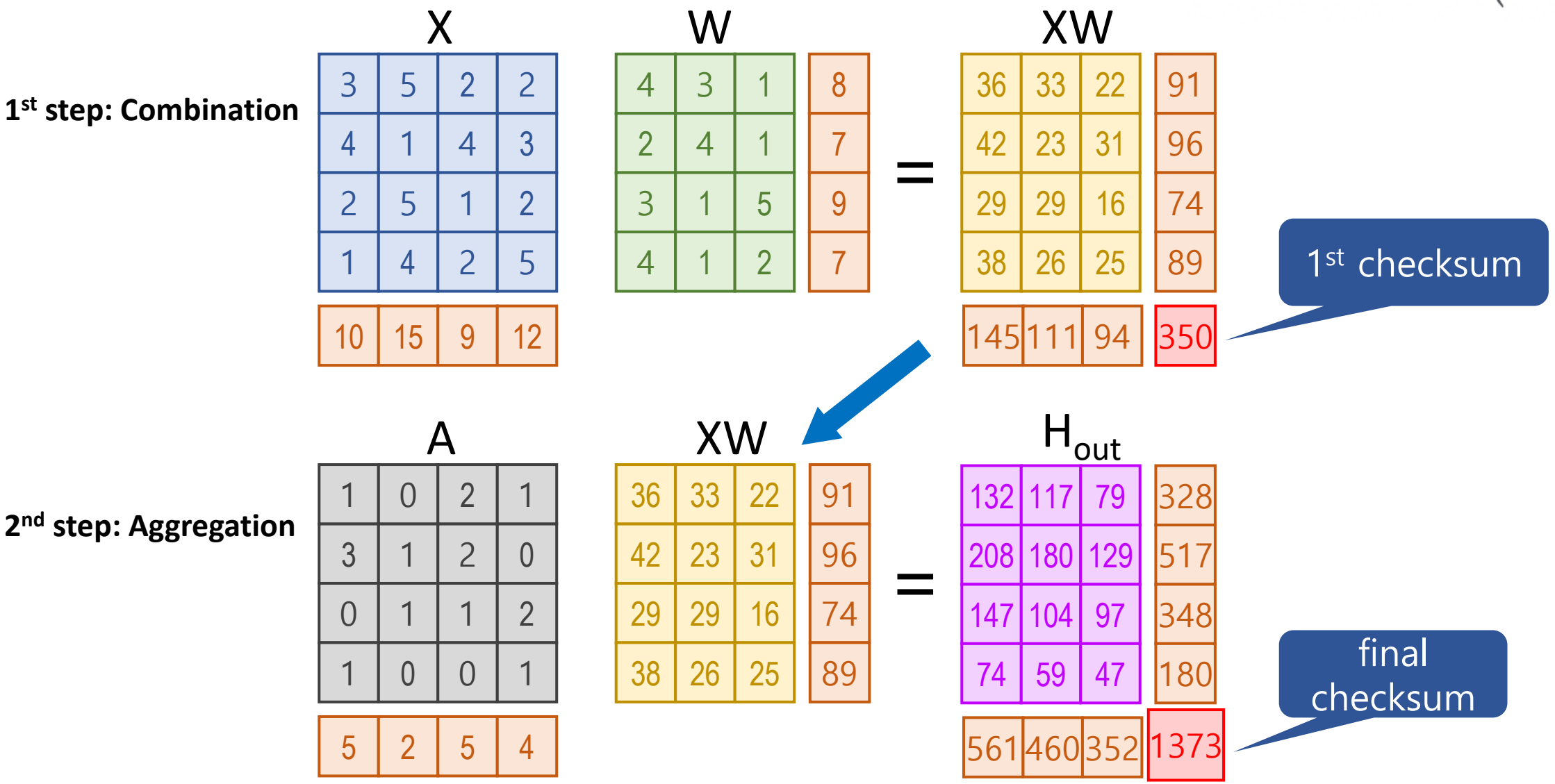
GCN computation: Combination

$$f(X, A) := \sigma\left(A\tilde{X}W\right)$$



Algorithmic-Based Fault Tolerance in GCNs

$$f(X,A) := \sigma\left(A\tilde{X}W\right)$$



GCN-ABFT

$$f(X,A) := \sigma\left(A\tilde{X}W\right)$$

1st step: Combination

X

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5
10	15	9	12

W

4	3	1	8
2	4	1	7
3	1	5	9
4	1	2	7

=

XW

36	33	22	91
42	23	31	96
29	29	16	74
38	26	25	89
145	111	94	350

1st checksum

2nd step: Aggregation

A

1	0	2	1
3	1	2	0
0	1	1	2
1	0	0	1
5	2	5	4

XW

36	33	22	91
42	23	31	96
29	29	16	74
38	26	25	89

=

H_{out}

132	117	79	328
208	180	129	517
147	104	97	348
74	59	47	180
561	460	352	1373

final checksum

GCN-ABFT

$$f(X,A) := \sigma\left(A\tilde{X}W\right)$$

Could be pre-computed

1st step: Combination

X

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

W

4	3	1	8
2	4	1	7
3	1	5	9
4	1	2	7

=

XW

36	33	22	91
42	23	31	96
29	29	16	74
38	26	25	89

Compute only per-row checksum of XW at the 1st step

2nd step: Aggregation

A

1	0	2	1
3	1	2	0
0	1	1	2
1	0	0	1
5	2	5	4

XW

36	33	22	91
42	23	31	96
29	29	16	74
38	26	25	89

=

H_{out}

132	117	79	328
208	180	129	517
147	104	97	348
74	59	47	180
561	460	352	1373

Could be pre-computed

final checksum

GCN-ABFT: Fault Detection Properties

- Single-bit flips injected randomly into MACs or checksum accumulation using FP arithmetic
  - FP32 for MACs
  - FP64 for checksum
- Setting the checksum error bound to 10^{-7} → eliminates silent faults in all GCNs
- Multi-fault campaigns show 100% detection in both ABFT and GCN-ABFT

GCN	Critical Faults	Avg. Nodes Affected		Split	GCN-ABFT
Cora	96.92%	68.61%	**Detected**	95.80%	96.66%
			False Pos	4.20%	3.34%
			Silent	0.00%	0.00%
Citeseer	92.60%	33.70%	**Detected**	95.44%	97.06%
			False Pos	4.56%	2.94%
			Silent	0.00%	0.00%
PubMed	96.10%	28.32%	**Detected**	96.38%	97.42%
			False Pos	3.62%	2.58%
			Silent	0.00%	0.00%
Nell	95.32%	61.06%	**Detected**	96.90%	97.82%
			False Pos	3.10%	2.18%
			Silent	0.00%	0.00%

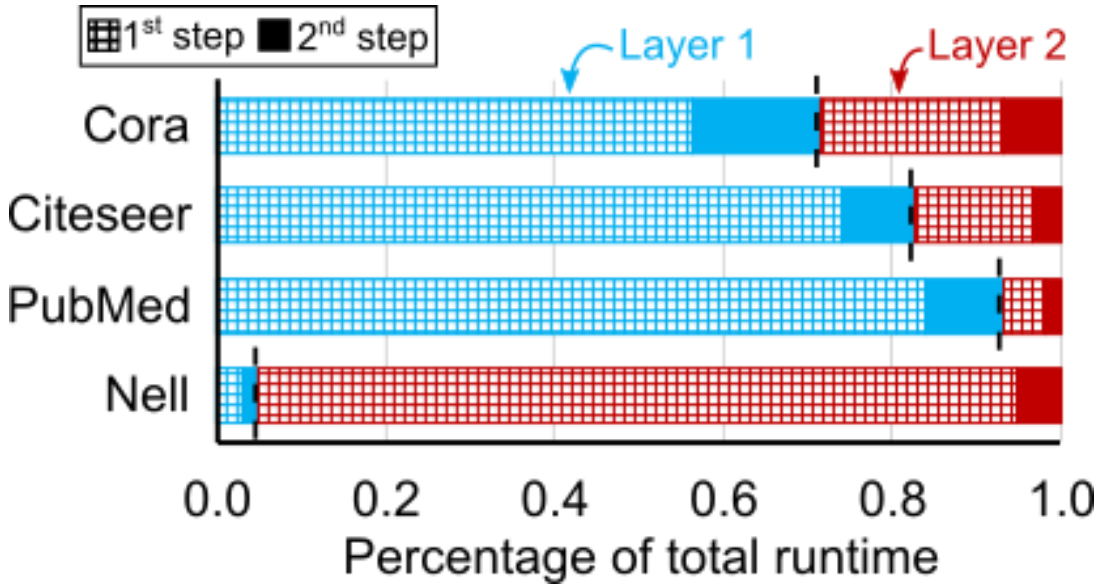
Computation Savings

GCN-ABFT validates each GCN layer with a *single fused checksum* → requires fewer total operations than the baseline split-checksum ABFT

GCN	True Outputs (M)	Split		GCN-ABFT		Savings	
		Check (M)	Total (M)	Check (M)	Total (M)	Check	Total
Cora	2.8	0.55	3.35	0.44	3.24	20.0%	3.3%
Citeseer	4.6	0.80	5.40	0.60	5.20	25.0%	3.7%
PubMed	37.6	4.60	42.20	4.04	41.64	12.2%	1.3%
Nell	1745.9	84.3	1830.2	59.9	1805.8	28.9%	1.3%

Error Detection Latency Gap

- GCN-ABFT detects errors reliably at the end of each GCN layer, regardless of when the error occurs.
- The first multiplication step dominates runtime making early detection less impactful.
  - >90% in PubMed
  - ~95% in Nell



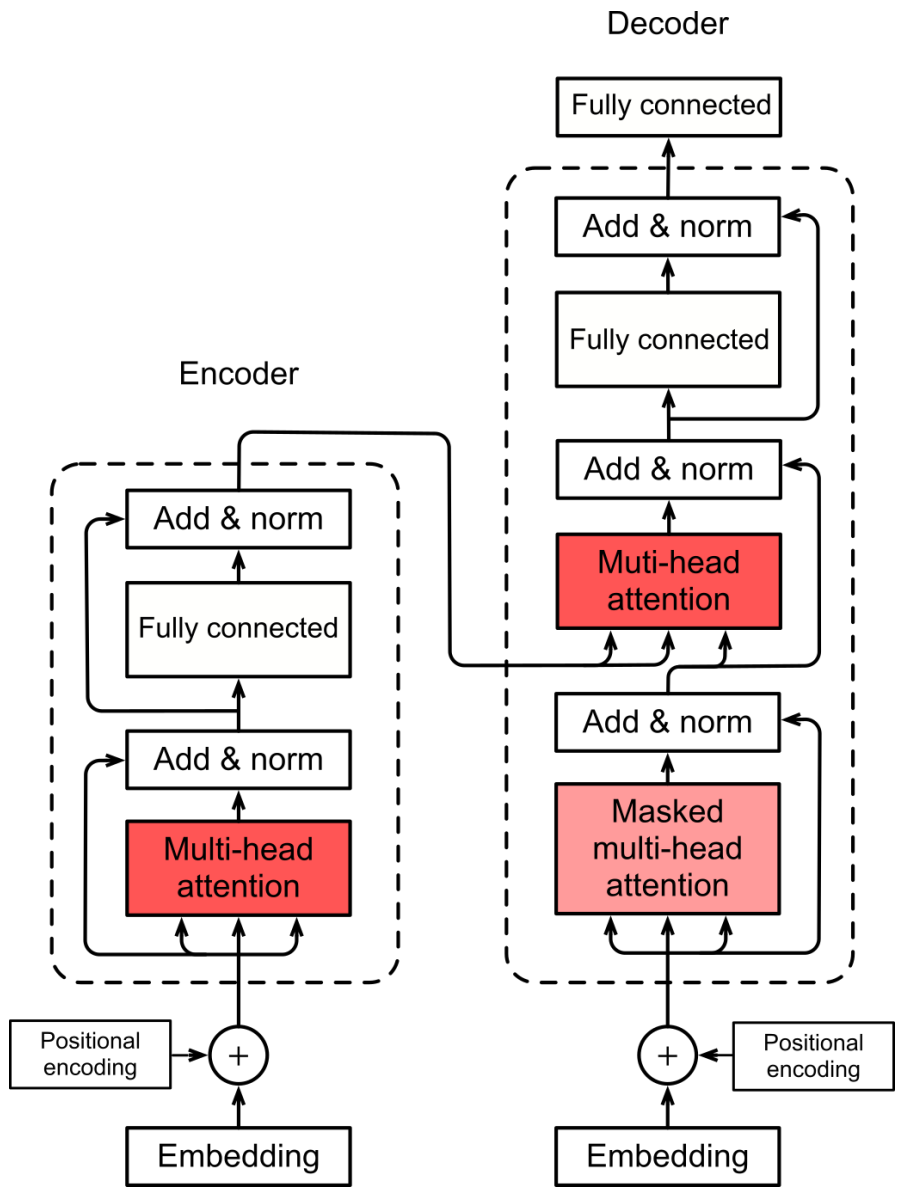
- The delay in error notification compared to baseline ABFT is negligible, since waiting for the second multiplication adds minimal overhead.

Part 3

TRANSFORMERS

Transformers are taking over…

- **Originally developed for NLP**
  - Used in image processing and decision-making Agentic systems
- **Different Transformer types**
  - Encoder (BERT)
  - Decoder (GPT-2 and LLMs)
  - Both (T5)
- **Attention is the core mechanism**
  - Transformer's decision-making capabilities are attributed to attention



The Attention Mechanism

- **The input tokens are transformed to Query (Q), Key (K) and Value (V) matrices**
  - Each matrix contains N vectors $\vec{q}$, $\vec{k}$, $\vec{v}$ respectively
- **Attention uses the Q, K and V matrices to compute its output**

$$Attn(Q, K, V) = \text{softmax}(QK^T)V$$

- **To compute a single output vector, it considers a query vector** $\vec{q}$ and the entire K and V matrices

$$s_i = \text{dot}(\vec{q}, \vec{k}_i)$$

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \frac{e^{s_i}}{\sum_j e^{s_j}} \vec{v}_i \qquad \vec{v}_i = \frac{\text{softmax}(s_i)}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

ABFT for Attention

$$Attn(Q,K,V) = \text{softmax}(QK^T)V$$

Per row checksum

$$\begin{bmatrix} \frac{e^{s_{11}}}{\sum_j e^{s_{1j}}} & \frac{e^{s_{12}}}{\sum_j e^{s_{1j}}} & \frac{e^{s_{13}}}{\sum_j e^{s_{1j}}} \\ \frac{e^{s_{21}}}{\sum_j e^{s_{2j}}} & \frac{e^{s_{22}}}{\sum_j e^{s_{2j}}} & \frac{e^{s_{23}}}{\sum_j e^{s_{2j}}} \\ \frac{e^{s_{31}}}{\sum_j e^{s_{3j}}} & \frac{e^{s_{32}}}{\sum_j e^{s_{3j}}} & \frac{e^{s_{33}}}{\sum_j e^{s_{3j}}} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix} \begin{matrix} \sum_i V_{1i} \\ \sum_i V_{2i} \\ \sum_i V_{3i} \end{matrix}$$

ABFT for Attention

$$Attn(Q,K,V) = \text{softmax}(QK^T)V$$

Per row checksum

$$\begin{matrix} \frac{1}{\sum_j e^{s_{1j}}} \\ \frac{1}{\sum_j e^{s_{2j}}} \\ \frac{1}{\sum_j e^{s_{3j}}} \end{matrix} \begin{bmatrix} e^{s_{11}} & e^{s_{12}} & e^{s_{13}} \\ e^{s_{21}} & e^{s_{22}} & e^{s_{23}} \\ e^{s_{31}} & e^{s_{32}} & e^{s_{33}} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix} \begin{matrix} \sum_i V_{1i} \\ \sum_i V_{2i} \\ \sum_i V_{3i} \end{matrix}$$

ABFT for Attention

$$\frac{1}{\sum_j e^{s_{1j}}} \left[e^{s_{11}} \quad e^{s_{12}} \quad e^{s_{13}} \right] \left[\sum_i V_{1i} \right]$$
$$\frac{1}{\sum_j e^{s_{2j}}} \left[e^{s_{21}} \quad e^{s_{22}} \quad e^{s_{23}} \right] \left[\sum_i V_{2i} \right]$$
$$\frac{1}{\sum_j e^{s_{3j}}} \left[e^{s_{31}} \quad e^{s_{32}} \quad e^{s_{33}} \right] \left[\sum_i V_{3i} \right]$$

$$\frac{1}{\sum_j e^{s_{1j}}} \left(e^{s_{11}} \cdot \sum_i V_{1i} + e^{s_{12}} \cdot \sum_i V_{2i} + e^{s_{13}} \cdot \sum_i V_{3i} \right) +$$
$$\frac{1}{\sum_j e^{s_{2j}}} \left(e^{s_{21}} \cdot \sum_i V_{1i} + e^{s_{22}} \cdot \sum_i V_{2i} + e^{s_{23}} \cdot \sum_i V_{3i} \right) +$$
$$\frac{1}{\sum_j e^{s_{3j}}} \left(e^{s_{31}} \cdot \sum_i V_{1i} + e^{s_{32}} \cdot \sum_i V_{2i} + e^{s_{33}} \cdot \sum_i V_{3i} \right)$$

$$\boldsymbol{check}\big(Attn(Q,K,V)\big) = \sum_i \frac{1}{\sum_j e^{s_{ij}}} \left(\sum_k e^{s_{ik}} \cdot \sum_i V_{ki} \right)$$

Re-used from attention computation

Attention output checksum with shared logic

$$\boldsymbol{check}\big(Attn(Q,K,V)\big) = \sum_i \frac{1}{\sum_j e^{s_{ij}}} \left(\sum_{\boldsymbol{k}} e^{s_{i\boldsymbol{k}}} \cdot \sum_i V_{\boldsymbol{k}i} \right)$$

Check for one query $\quad \boldsymbol{check}(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \left(\sum_{i=1\ldots N} e^{s_i} \cdot \sum_i \vec{v}_i \right)$

Attention for one query $\quad Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i}\, \vec{v}_i$

The Attention Mechanism: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

=

Attn

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

$sum(\vec{v}_i)$

8
7
9
7

=

Attn

				$\mathrm{chk}(\vec{q})$

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

$sum(\vec{v}_i)$

8
7
9
7

=

Attn

2	4	1	

$\mathrm{chk}(\vec{q})$

7

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

$sum(\vec{v}_i)$

8
7
9
7

=

Attn

2	4	1
4	2	3

$\mathrm{chk}(\vec{q})$

7
9

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

$sum(\vec{v}_i)$

8
7
9
7

=

Attn

2	4	1
4	2	3
2	4	1

$\mathrm{chk}(\vec{q})$

7
9
7

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$

Score (QK^T)

3	5	2	2
4	1	4	3
2	5	1	2
1	4	2	5

Value

4	3	1
2	4	1
3	1	5
4	1	2

$sum(\vec{v}_i)$

8
7
9
7

=

Attn

2	4	1
4	2	3
2	4	1
3	1	2

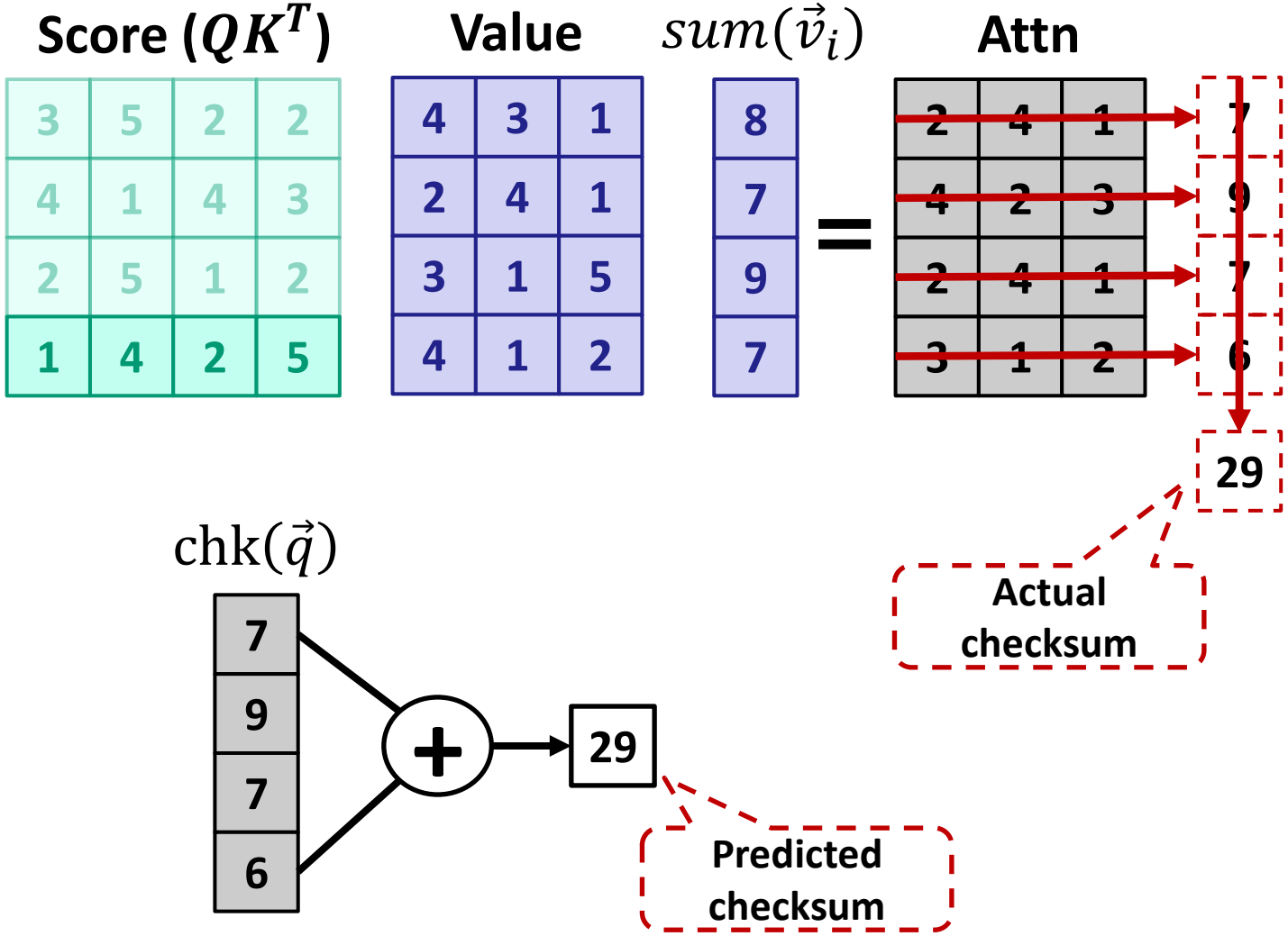
$\mathrm{chk}(\vec{q})$

7
9
7
6

The Attention Mechanism + ABFT: Example

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot sum\,(\vec{v}_i)$$



Attention with lazy softmax division

- **Normalize attention scores to avoid numerical instabilities**
  - This is done by subtracting the maximum score from all scores
  - Softmax function characteristics are preserved

$$f_i = \frac{e^{s_i}}{\sum_j e^{s_j}} \quad \xrightarrow{\text{Normalize}} \quad f_i = \frac{e^{s_i - \max}}{\sum_j e^{s_j - \max}}$$

- **Perform division once at the end**

$$s_i = \text{dot}(\vec{q}, \vec{k}_i) \qquad f_i = e^{s_i - \max}$$

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{\sum_i f_i \vec{v}_i}{\sum_j e^{s_j - \max}}$$

Serial dependency due to m_N calculation

Algorithm 1 Attention with lazy softmax division

1: **for each** query $\vec{q}$ **do**
2: **for** $i = 1 : N$ **do**
3: $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
4: $m_i \leftarrow \max(m_{i-1}, s_i)$
5: **end for**
6: $\ell_0 \leftarrow 0$
7: **for** $i = 1 : N$ **do**
8: $\vec{o}_i \leftarrow \vec{o}_{i-1} + e^{s_i - m_N} \cdot \vec{v}_i$
9: $\ell_i \leftarrow \ell_{i-1} + e^{s_i - m_N}$
10: **end for**
11: $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
12: **end for**

Normalize scores

Division performed once

FlashAttenion-2

- **The two dependent loops are merged into one**
  - The sum and output accumulations are now performed using a 'local' maximum score m_i
- **To maintain a correct result, we need to correct the previous accumulated values**

Algorithm 1 Attention with lazy softmax division

1: **for each** query $\vec{q}$ **do**
2: **for** $i = 1 : N$ **do**
3: $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
4: $m_i \leftarrow \max(m_{i-1}, s_i)$
5: **end for**
6: $\ell_0 \leftarrow 0$
7: **for** $i = 1 : N$ **do**
8: $\vec{o}_i \leftarrow \vec{o}_{i-1} + e^{s_i - m_N} \cdot \vec{v}_i$
9: $\ell_i \leftarrow \ell_{i-1} + e^{s_i - m_N}$
10: **end for**
11: $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
12: **end for**

Algorithm 2 FlashAttention-2

1: **for each** query $\vec{q}$ **do**
2: **for** $i = 1 : N$ **do**
3: $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
4: $m_i \leftarrow \max(m_{i-1}, s_i)$
5: $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
6: $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i e^{s_i - m_i}$
7: **end for**
8: $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
9: **end for**

Division performed once

The Attention Mechanism + ABFT

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

$$\mathrm{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot rowsum\,(\vec{v}_i)$$

Algorithm 2 FlashAttention-2

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \mathrm{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i e^{s_i - m_i}$
  end for
  $\mathrm{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
end for

- **Max values guarantee the numerical stability of e^x**

Algorithm 3 FlashAttention-2 with online checksum computation

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \mathrm{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i \cdot e^{s_i - m_i}$
    $c_i \leftarrow c_{i-1} e^{m_{i-1} - m_i} + sumrow_i(V) \cdot e^{s_i - m_i}$
  end for
  $\mathrm{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
  $check(\vec{q}) = c_N / \ell_N$
  $check \leftarrow check + check(\vec{q})$
end for

The Attention Mechanism + ABFT

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

Algorithm 2 FlashAttention-2

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
end for

- **Max values guarantee the numerical stability of e^x**

$$\text{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot rowsum\,(\vec{v}_i)$$

Algorithm 3 FlashAttention-2 with online checksum computation

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i \cdot e^{s_i - m_i}$
    $c_i \leftarrow c_{i-1} e^{m_{i-1} - m_i} + sumrow_i(V) \cdot e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
  $check(\vec{q}) = c_N / \ell_N$
  $check \leftarrow check + check(\vec{q})$
end for

The Attention Mechanism + ABFT

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

Algorithm 2 FlashAttention-2

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
end for

- **Max values guarantee the numerical stability of e^x**

$$\text{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot rowsum\,(\vec{v}_i)$$

Algorithm 3 FlashAttention-2 with online checksum computation

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i \cdot e^{s_i - m_i}$
    $c_i \leftarrow c_{i-1} e^{m_{i-1} - m_i} + sumrow_i(V) \cdot e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
  $check(\vec{q}) = c_N / \ell_N$
  $check \leftarrow check + check(\vec{q})$
end for

The Attention Mechanism + ABFT

$$Attn(\vec{q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \vec{v}_i$$

Algorithm 2 FlashAttention-2

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
end for

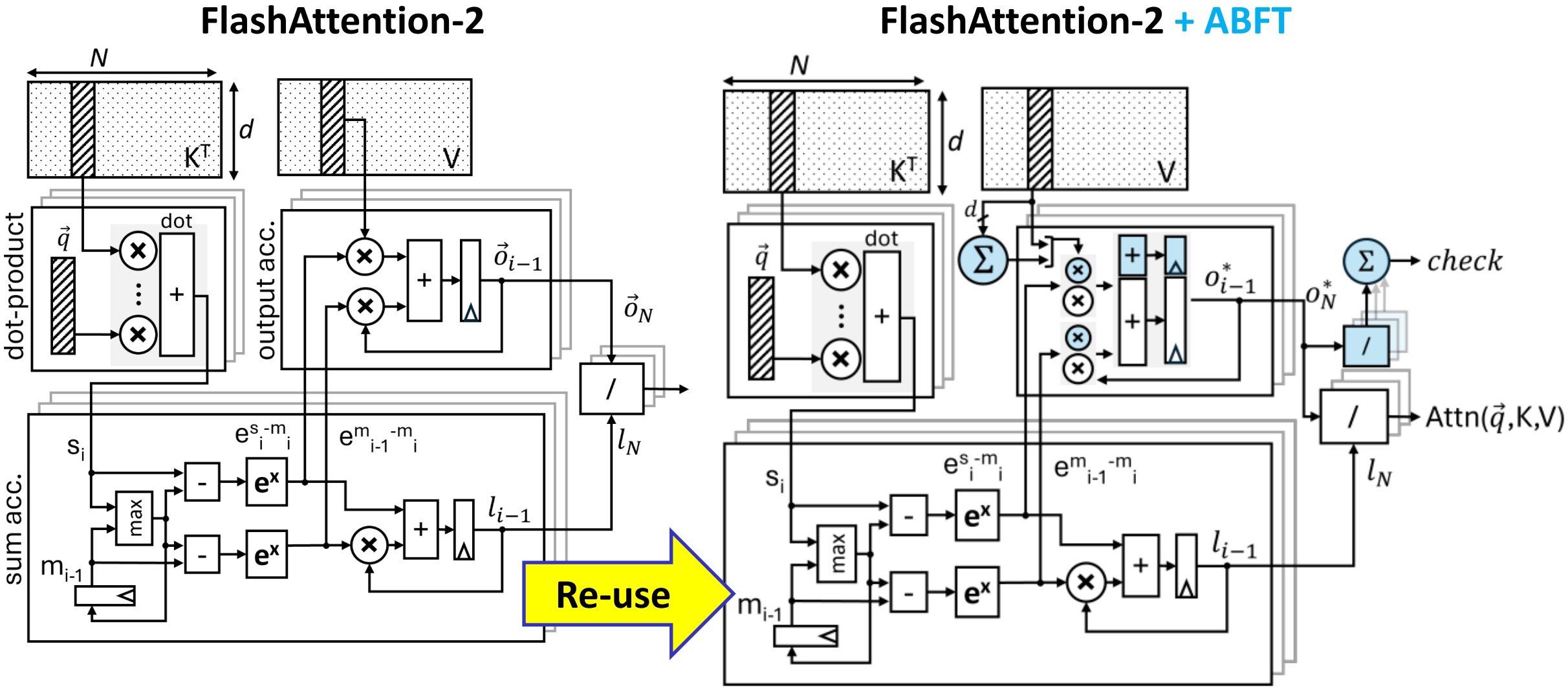
- **Max values guarantee the numerical stability of e^x**

$$\text{chk}(\vec{q}) = \frac{1}{\sum_j e^{s_j}} \sum_{i=1\ldots N} e^{s_i} \cdot rowsum\,(\vec{v}_i)$$

Algorithm 3 FlashAttention-2 with online checksum computation

for each query $\vec{q}$ **do**
  for $i = 1 : N$ **do**
    $s_i \leftarrow \text{dot}(\vec{q}, \vec{k}_i)$
    $m_i \leftarrow \max(m_{i-1}, s_i)$
    $\ell_i \leftarrow \ell_{i-1} e^{m_{i-1} - m_i} + e^{s_i - m_i}$
    $\vec{o}_i \leftarrow \vec{o}_{i-1} e^{m_{i-1} - m_i} + \vec{v}_i \cdot e^{s_i - m_i}$
    $c_i \leftarrow c_{i-1} e^{m_{i-1} - m_i} + sumrow_i(V) \cdot e^{s_i - m_i}$
  end for
  $\text{attn}(\vec{q}, K, V) \leftarrow \vec{o}_N / \ell_N$
  $check(\vec{q}) = c_N / \ell_N$
  $check \leftarrow check + check(\vec{q})$
end for

Hardware architecture overview



Experimental results: Detection analysis

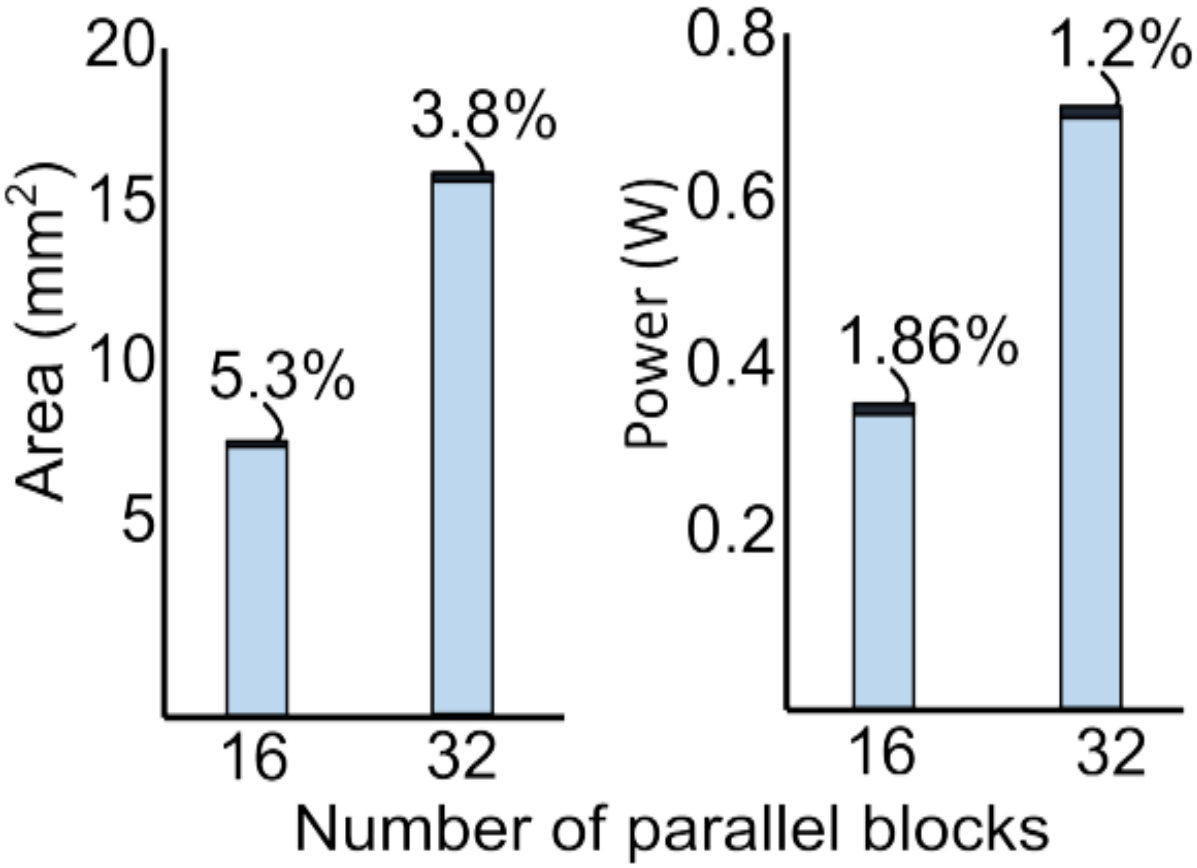
FAULT DETECTION ACCURACY FOR A SINGLE INJECTED FAULT USING AN ERROR BOUND OF 10^{-6} .

Fault injection behavior	Sequence length=256			
	d=64	d=96	d=128	d=256
Detected	96.94%	97.56%	98.45%	98.87%
False Positive	2.66%	1.99%	1.25%	0.62%
Silent	0.4%	0.45%	0.3%	0.51%

- Error injected by random bit flips during attention inference
- Detection rate is greater than 96.9% for various head dimension sizes
- 3% False positives and ~0.5% undetected in all examined cases

Experimental results: Hardware overhead

- **Results are reported for 16 and 32 parallel queries**
  - Size of a query is 128 elements
- **Area overhead less than 5.3%**
  - Increasing the number of parallel blocks reduces area overhead
- **Power overhead less than 2%**
  - Increasing the number of parallel blocks reduces power overhead
- **Parallel queries↑ ⇒ HW overhead↓**
  - Logic for reducing value vectors shared between parallel blocks



Key takeaways

- ABFT is a versatile technique for online detection of random hardware faults.
- **Algorithm-tuned ABFT reduces checking overhead** and **improves energy efficiency**
  - Smaller checking logic minimizes false-alarm vulnerability while keeping protection lightweight.
- **Math-enabled LLMs can accelerate ABFT design by automatically deriving algorithm-specific optimizations with appropriate guidance**
  - Our experiments with Gemini verified that, with the right guidance, it can rediscover the techniques I presented today, which we originally developed manually over the past three years.
  - The key requirement is effective mathematical exploration: rapidly interpreting derivations and iterating on ideas.