Accelerating Sparse Deep Neural Networks

Asit Mishra, Jorge Albericio Latorre, Je Pool, Darko Stosic,
Dusan Stosic, Ganesh Venkatesh, Chong Yu, Paulius Micikevicius, NVIDIA

Presented by: Mohamed Ibrahim

PART 1 · MOTIVATION

What do we want from sparsity?

Four goals — and they don't always come together.

Reduce memory footprint

Compress models so they fit in DRAM, on-chip SRAM, or on edge devices.

Preserve accuracy

Compressed model should match the dense baseline within run-to-run noise.

Reduce computation time

Skip multiplications by zero — only if the hardware exploits the pattern.

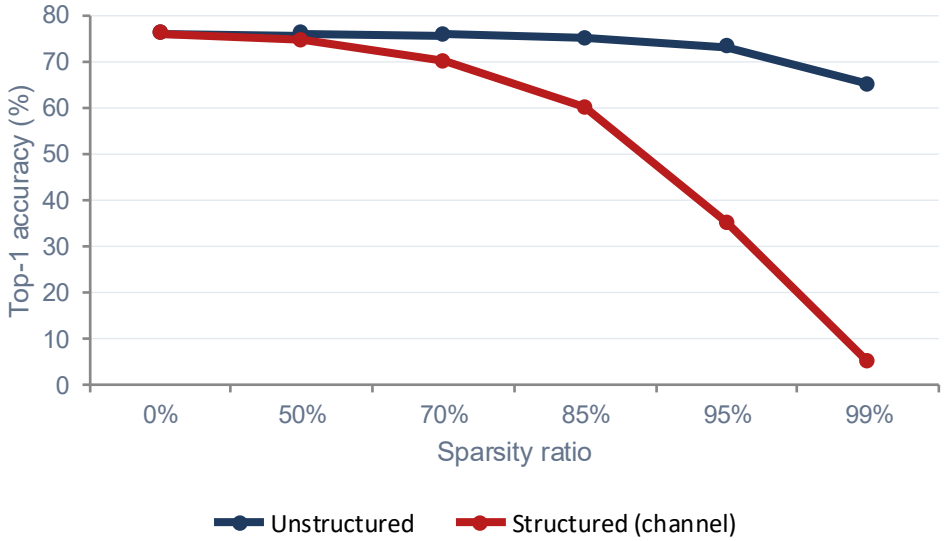
Reduce power consumption

Avoid memory traffic; gate idle logic. The hardware-aware aspect.

Different methods optimize different subsets of these. **The art is hitting all four.**

The classical approach

Maximize sparsity to reduce memory footprint with a modest accuracy penalty.



Illustrative — ResNet-50 on ImageNet

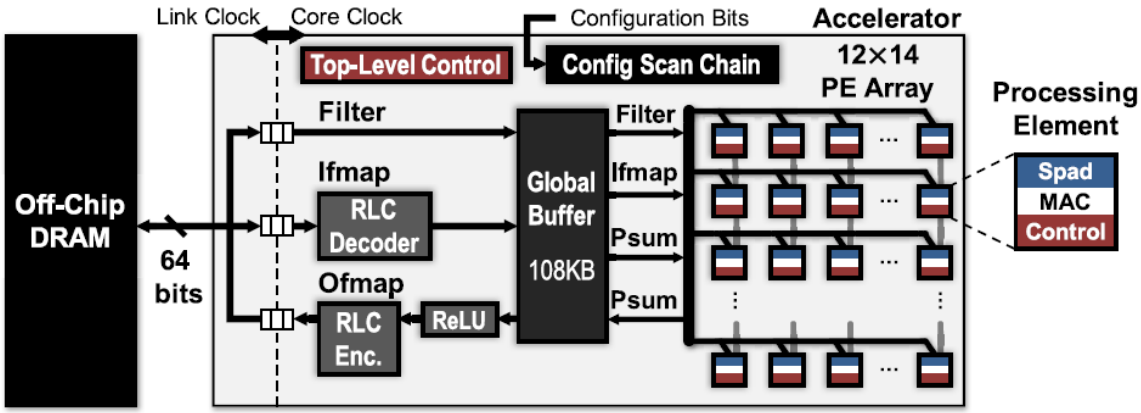
Why unstructured?

- Highest memory compression
- Han et al. 2015: 13× pruning, no accuracy loss
- Best accuracy retention at high sparsity

The catch

- Irregular memory access patterns
- Bad fit for matrix-math hardware

Case study: Eyeriss V1



DRAM
RLC compressed

→

RLC decoder
decompresses

→

PE array
12 × 14, dense compute

Takeway
Compression alone saves bandwidth, not computation time.

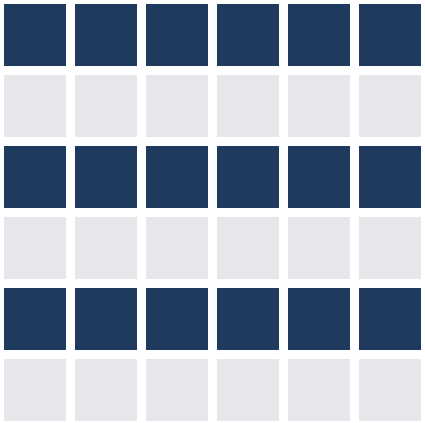
Eyeriss V1 dataflow (Chen et al., ISSCC 2016 / JSSC 2017)

PART 2

Hardware–software co-design

Structured sparsity

Drop whole channels or filters — what's left is just smaller and dense.



Whole rows removed → smaller dense matrix

PROS

- Resulting matrix is dense — no special HW needed
- Real time savings: skip whole columns of MACs
- No metadata, no indirection, no decoder

CONS

- Coarse — easy to drop important weights
- ResNet-50 channel pruning: 2× speedup, ~1.5% accuracy loss

The compromise

Don't push sparsity to the limit — co-design the pattern.

Unstructured @ ~95%	Structured @ ~50%	2:4 fine-grained @ 50%
Memory: maximal	Memory: half	Memory: ~half
Accuracy: held	Accuracy: degrades	Accuracy: held
Speedup: ~none	Speedup: real	Speedup: 2×
Hardware: bad fit	Hardware: Simple	Hardware: Designed with Algorithm in mind

The bet: a milder sparsity ratio with *just enough structure* for hardware to exploit.

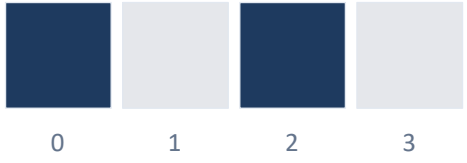
PART 3 · PAPER SECTION 3

The 2:4 pattern

Format, storage, and Sparse Tensor Cores.

The pattern

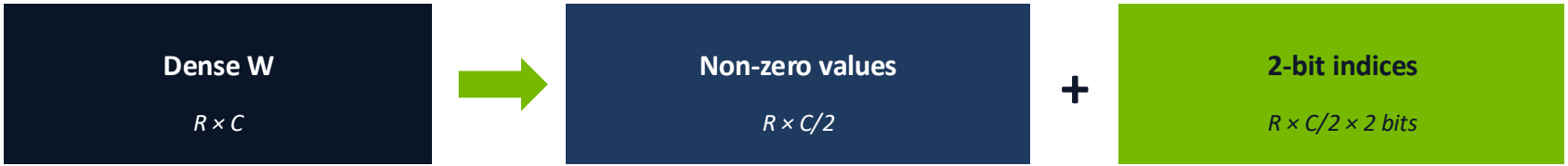
In every group of 4 consecutive weights, at least 2 must be zero.



A group of 4 consecutive weights

- Two non-zero values at known positions (here: 0 and 2)
- Encoded with just 2 bits per non-zero value

Compressed storage format



Half the values stored, plus a small index — no row pointers, no indirection.

PART 3 · THE 2:4 PATTERN

How much do we save?

Worked example: storing one group of 4 weights.

FP16 (16-bit)

~44%
savings

Dense:	`4 × 16 = 64 bits`
Sparse (vals):	`2 × 16 = 32 bits`
Sparse (idx):	`2 × 2 = 4 bits`
Total:	**`36 bits`**

INT8 (8-bit)

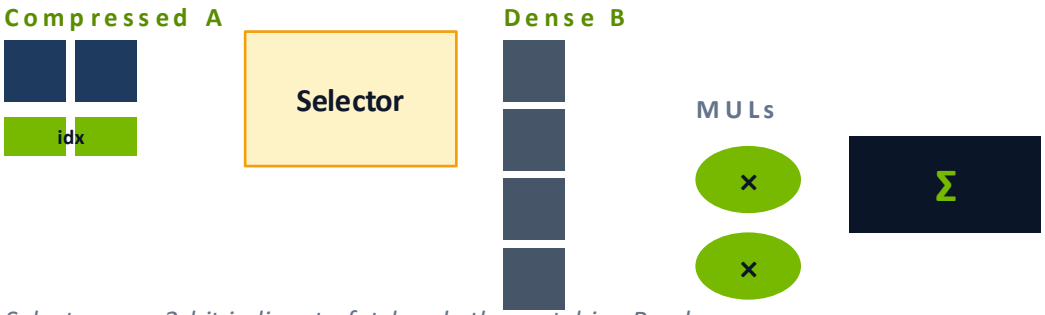
~38%
savings

Dense:	`4 × 8 = 32 bits`
Sparse (vals):	`2 × 8 = 16 bits`
Sparse (idx):	`2 × 2 = 4 bits`
Total:	**`20 bits`**

Compare to CSR for INT8: metadata can reach **200% overhead**. 2:4 caps it at ~25%.

Sparse Tensor Cores

How the silicon turns 50% sparsity into 2× math throughput.



Selector uses 2-bit indices to fetch only the matching B values.

Format	Dense	Sparse
TF32	156	**312**
FP16/BF16	312	**624**
INT8	624	**1248**

A100 Tensor Core peak TOPS

Half the multipliers, fed selectively from a full-width B → **2× math throughput.**

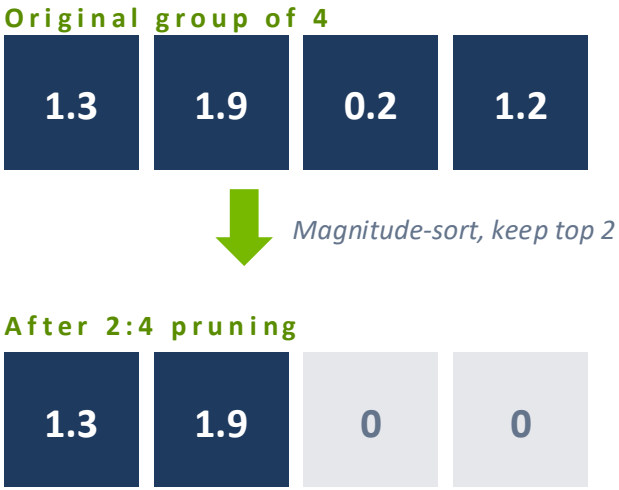
PART 4 · PAPER SECTION 4

Training the network

How to actually produce a 2:4-conformant model.

How do we choose which weights to drop?

Group consecutive weights in 4s; keep the two largest in magnitude.



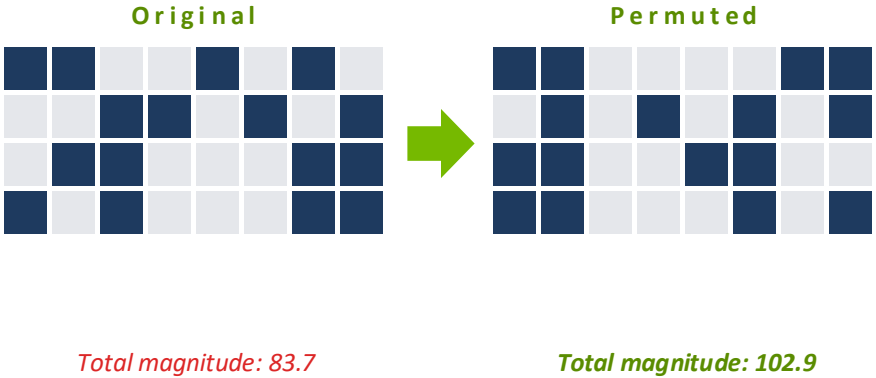
Why magnitude?

- Smallest weights contribute least to the layer's output
- Simple — no extra hyper-parameters or calibration data
- Authors found magnitude was sufficient; gradient/salience also possible

Note: groups are formed in row-major order along the GEMM-K dimension.

Channel permutations

When 2:4 hurts: rearrange columns to spread the large weights out.



The intuition

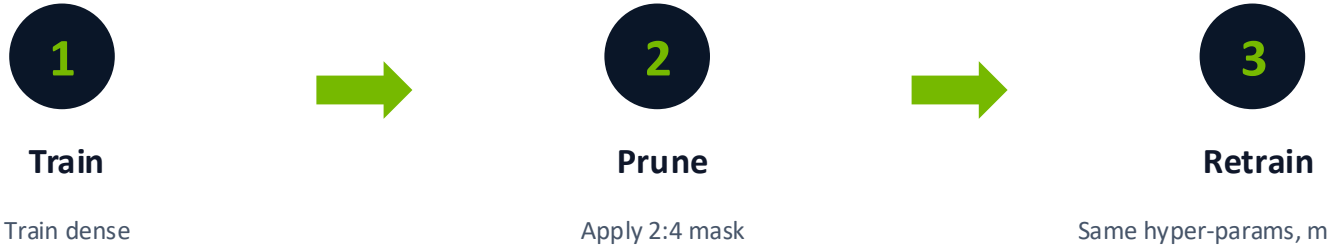
- If large weights cluster in one group of 4, the constraint forces us to drop some
- Permuting columns spreads them across groups → more total magnitude survives
- Inverse permutation absorbs into the previous layer — zero runtime cost

Follow-up: Pool & Yu, *"Channel Permutations for N:M Sparsity,"* NeurIPS 2021

PART 4 · TRAINING

The recipe — and why it scales

Three steps. No per-network tuning. Validated across vision, NLP, GANs.



RESULTS — accuracy held across architectures (selected)

Domain	Network	Metric	Dense	Sparse 2:4
Image cls.	ResNet-50	Top-1 (ImageNet)	76.1	**76.2**
Detection	MaskRCNN-RN50	bbox AP (COCO)	37.9	**37.9**
Translation	Transformer	BLEU (En-De)	28.2	**28.5**
Lang. modeling	BERT-Large	F1 (SQuAD)	91.9	**91.9**

Differences within run-to-run noise across all 30+ models tested.

What does this cost?

The recipe is simple — but it doubles the training budget.

Why it's still worth it:

- Training is a one-time cost; inference runs for the model's deployment lifetime
- No hyper-parameter search across networks, making this a universal recipe.

PART 5 · WHAT'S NEXT

Open directions

Shorter retraining schedules

Today: full retraining. Per-network shortcuts exist; a universal one is open.

Accelerate training itself

Backward pass needs 2:4 along both dims of W — "transposable masks" (Hubara 2021).

Activation sparsity

Multi-head attention has no weights. Pruning activations with 2:4 looks promising.

Beyond 2:4

N:M generalizes (1:4, 2:8, 4:8). Different ratios trade accuracy vs. silicon area.

Active research: **transposable masks, dynamic masks, attention sparsity, LLM-scale post-training pruning.**

CONCLUSION

Sparsity aware hardware-software co-design achieves modest sparsity with high memory footprint savings, computation time decrease and power consumption decrease.

Pattern

2:4 sparsity — fine-grained but structured. 50% sparsity, predictable layout.

Hardware

Sparse Tensor Cores: a selector stage on top of dense Tensor Cores → 2× math throughput.

Algorithm

Train, prune, retrain — same hyper-parameters, no per-network tuning. Doubles training cost only.

Mishra et al. — Accelerating Sparse Deep Neural Networks · arXiv:2104.08378