New Architectural Opportunities for LLM Quantization

Numerical Representation, Architecture, and Computing Paradigm

Cong Guo

ICCAD 2026 Workshop on Memory-Centric Computing for LLM Inference

Background

AI Is Everywhere, and LLMs Keep Growing

900M+

weekly ChatGPT users (OpenAI, Feb 2026)

ChatGPT
Chat assistant

GitHub Copilot
Coding

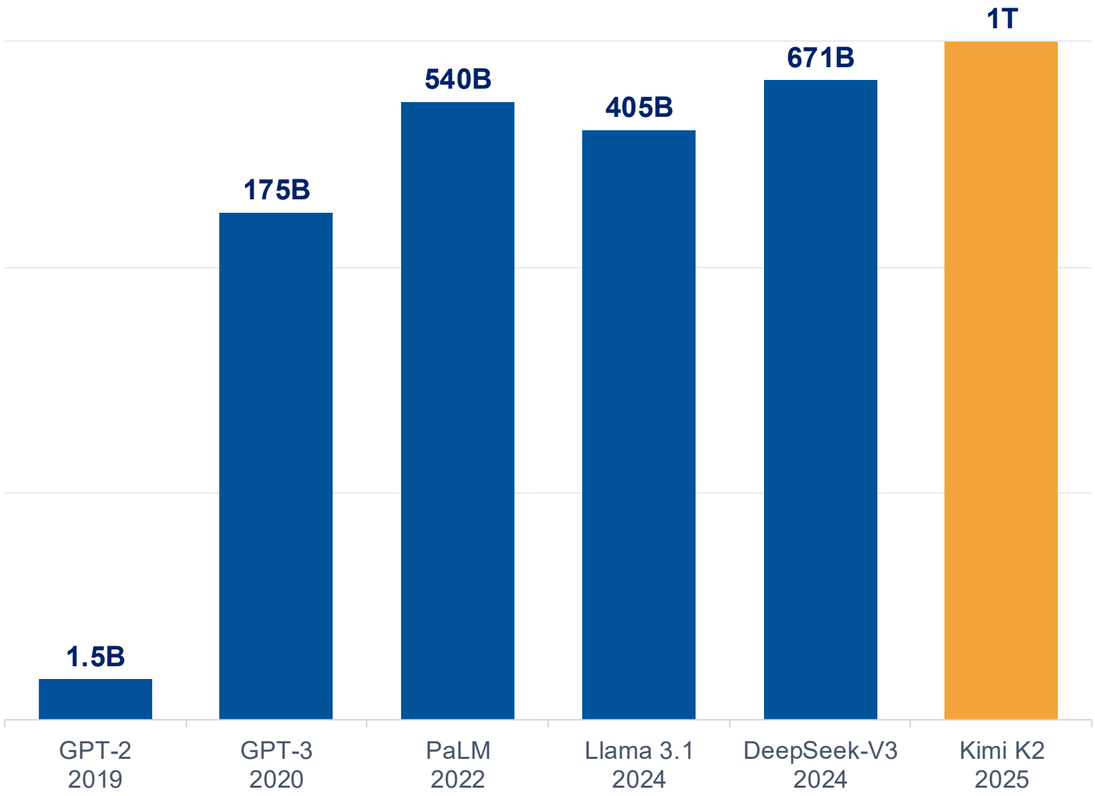
Gemini
Search + assistant

Meta AI
Social apps

Microsoft 365 Copilot
Office work

Midjourney
Image generation

Model parameters (log scale)



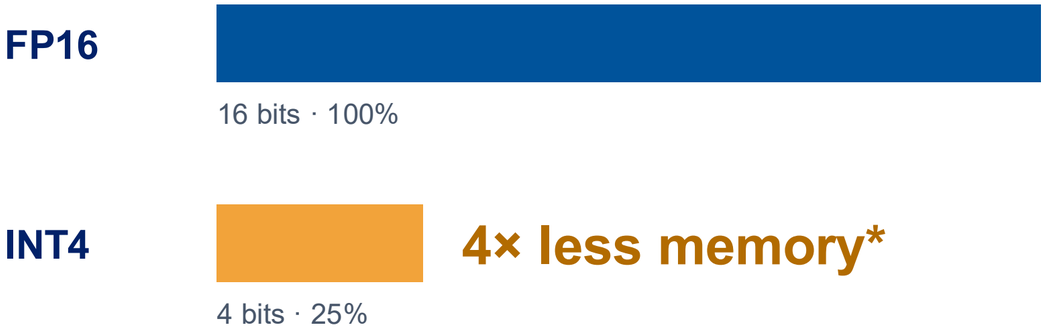
- **Usage is exploding, and every token must stream through ever-larger models.**

Sources: OpenAI (Feb 2026); model papers / release notes. Kimi K2 and DeepSeek-V3 are MoE (total parameters).

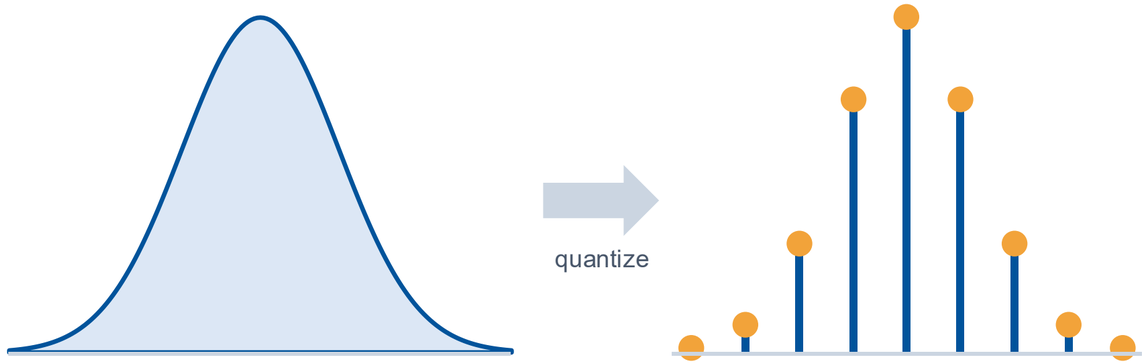
Background

Quantization Makes LLM Inference Cheaper

Fewer bits per weight



Continuous values become discrete levels



Smaller footprint

Fit bigger models in memory

Less data movement

Fewer bytes per token

Faster compute

Low-precision tensor cores

Quantization is now essential, not optional, for LLM inference.

*Ideal storage reduction for weights, excluding scale metadata; not a fixed inference speedup.

Background

Quantization Has Become Infrastructure

Our survey of MICRO, ISCA, ASPLOS, and HPCA, 2018–2026: 744 records screened, 46 quantization-architecture papers.

Quantization-relevant papers per year



6.5%

of papers touched quantization in 2018

38%

peak share of papers, 2025–2026

12×

growth: 3 papers (2018) → 36 (2026)

Open questions

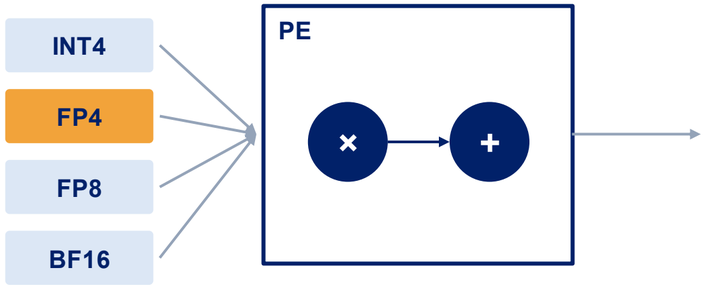
Where does the codebook live?

Who pays for contention?

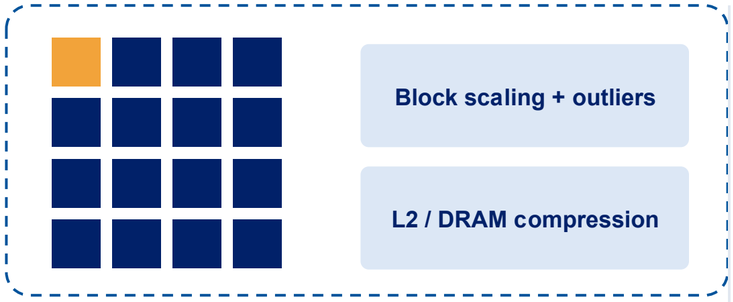
Can codebooks compose with sparsity?

Coverage: 744 official records · 46 quantization-oriented · 137 use quantization as background · 561 excluded.

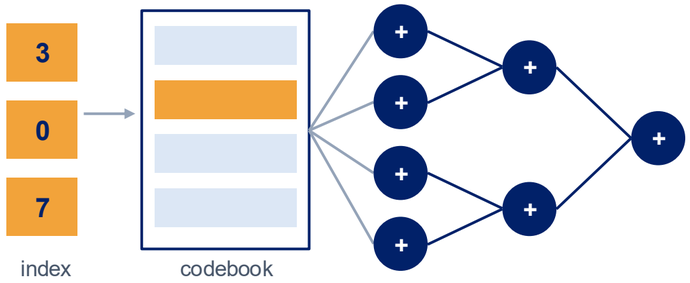
Quantization Co-Design in Three Stages



One PE, many data types



PE array + memory system



Adder tree, no multiplier

Stage 1

Numerics

New data types (custom INT / FP)
Exponent / mantissa allocation
Lightweight PE logic

Stage 2

Architecture

Block scaling (NVFP4, MX)
Outlier handling
Tensor-block dataflow

Stage 3

Computing Paradigm

Non-analytic formats (VQ, codebooks)
Lookup replaces multiplication
Data / hardware co-design

Our Work

MICRO 2022	ISCA 2023	HPCA 2025	ISCA 2025	ISCA 2026	ASPLOS 2027
ANT	OliVe	MANT	Transitive Array	EVA	Vortex

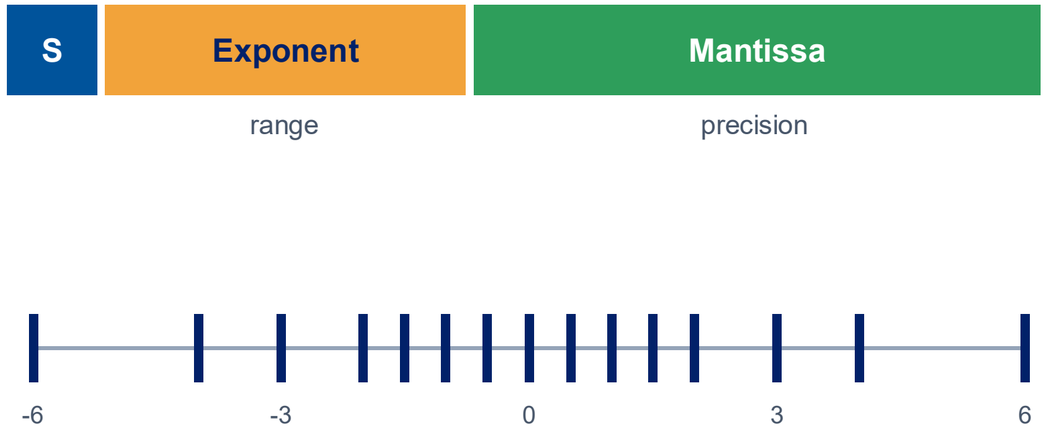
01

Stage 1: Numerics

New data types inside the processing element

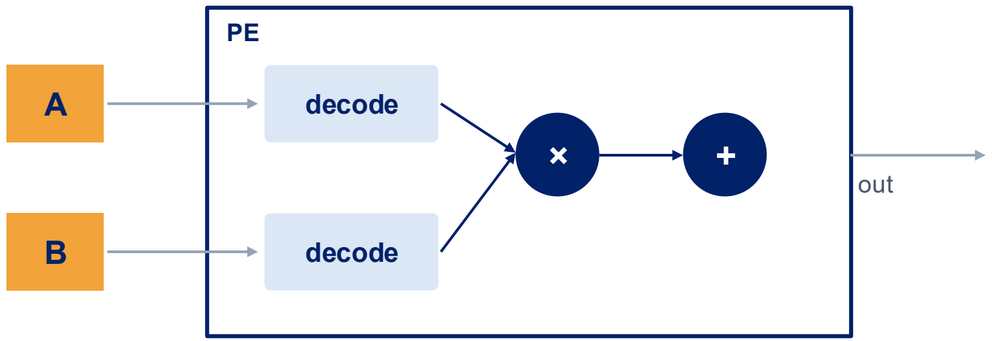
A New Data Type and Its PE

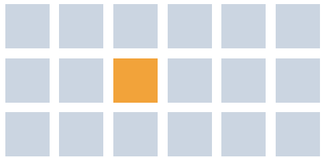
1. Choose the bits



Representable values (FP4 E2M1): dense near zero, sparse in the tails

2. Build the PE for it



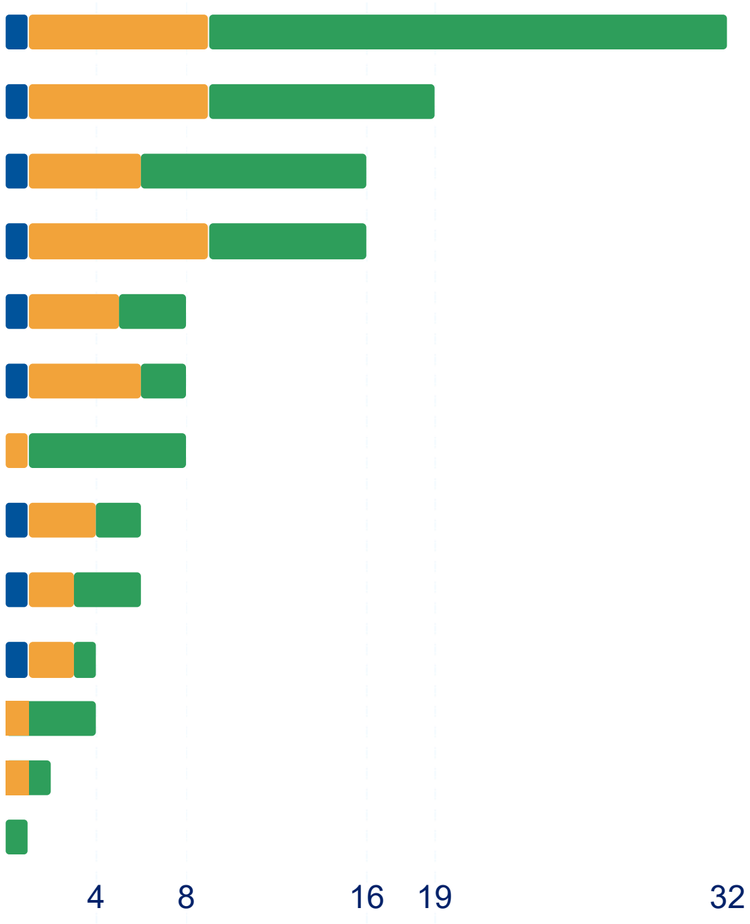


Only this PE changes.
The array and memory stay the same.

- **Stage 1 changes what a number is, and only the PE that computes it.**

Scalar Data Types in Commercial Silicon

First hardware recorded in the supplied reference. NVIDIA narrow-float arithmetic uses Tensor Cores.



Format	Encoding	Bits	First Hardware (Reference)
FP32	1·8·23	32	— (not identified in the reference)
TF32	1·8·10	19	NVIDIA A100 (**2020**)
FP16	1·5·10	16	NVIDIA V100 Tensor Core (**2017**)
BF16	1·8·7	16	Google TPUv2 (**2017**)
FP8 E4M3	1·4·3	8	NVIDIA H100 (**2022**)
FP8 E5M2	1·5·2	8	NVIDIA H100 (**2022**)
INT8	2's comp.	8	Google TPUv1 (**2016**)
FP6 E3M2	1·3·2	6	NVIDIA GB200 (**2024**; MX)
FP6 E2M3	1·2·3	6	NVIDIA GB200 (**2024**; MX)
FP4 E2M1	1·2·1	4	Microsoft Maia 100 (**2023**; MXFP4)
INT4	2's comp.	4	NVIDIA T4 Tensor Core (**2018**)
INT2	integer	2	IBM NorthPole (**2023**)
INT1	binary	1	NVIDIA T4 Tensor Core (**2018**)

NVIDIA INT8: V100 DP4A (**2017**) ≠ T4 Tensor Core (**2018**). INT4 / INT1 dropped at Hopper.

FP6 / FP4 ship with block scales: MXFP6 6.25 b; MXFP4 4.25 b; NVFP4 4.5 b/value (scale included).

02

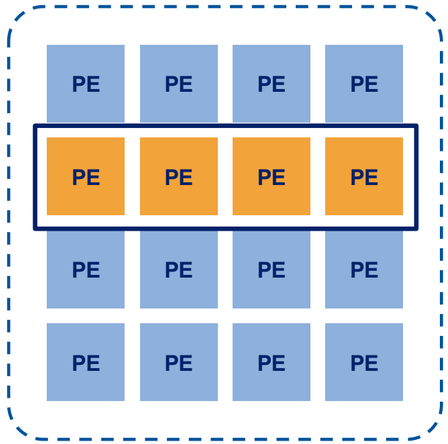
Stage 2: Architecture

Beyond the PE: granularity, mixed formats, outliers

Granularity + Mixed Formats = Outlier Handling

Quantization granularity

Tensor → channel → block granularity



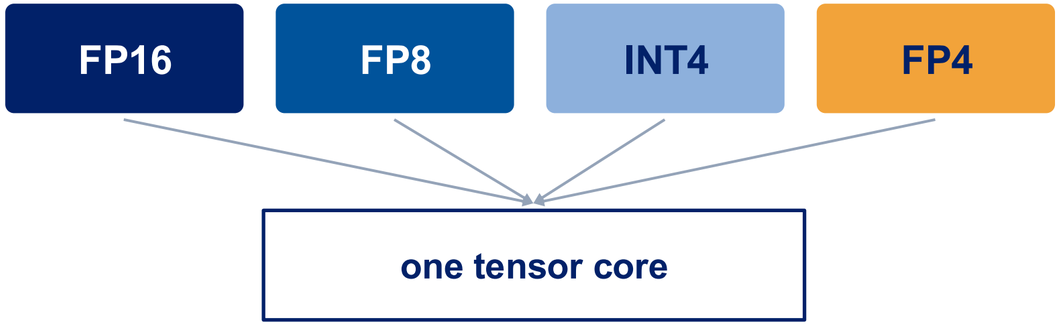
PE Array (4 × 4)

group size = 4

one scale per group

Mixed formats

Different types and bit widths, one tensor core



Push to element-wise

+ a different precision / data type per element



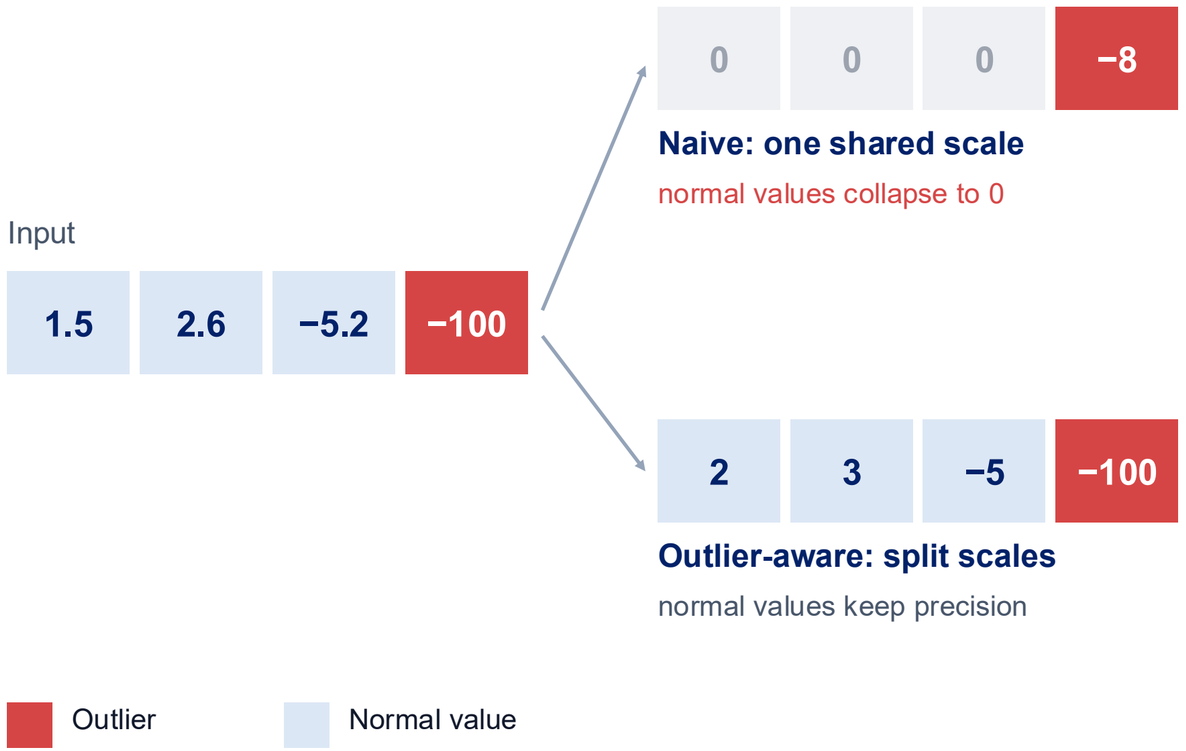
group size = 1: each element picks its own type

= Outlier handling

Normal values low-bit, outliers high-precision

Outlier Issues for Quantization

Quantize to INT4 (range −8 … 7)



One outlier ruins the shared scale

−100 sets the INT4 step to 12.5: 1.5, 2.6, −5.2 all become 0.

Keeping outliers needs a separate path

Normal values stay low-bit; outliers keep their range.

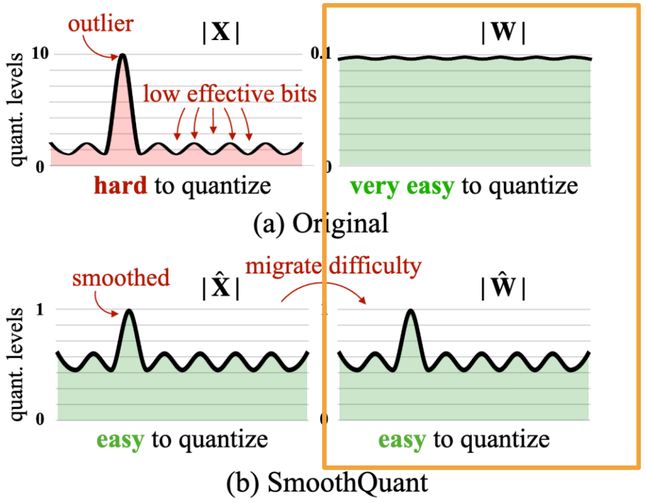
- **One outlier can wipe out every normal value in its group.**

Two Answers to Outliers: A Software–Hardware Trade-off

1 Algorithm

Adapt to today's hardware

Suppress / smooth outliers, e.g. SmoothQuant



Difficulty migrates from activations into the weights

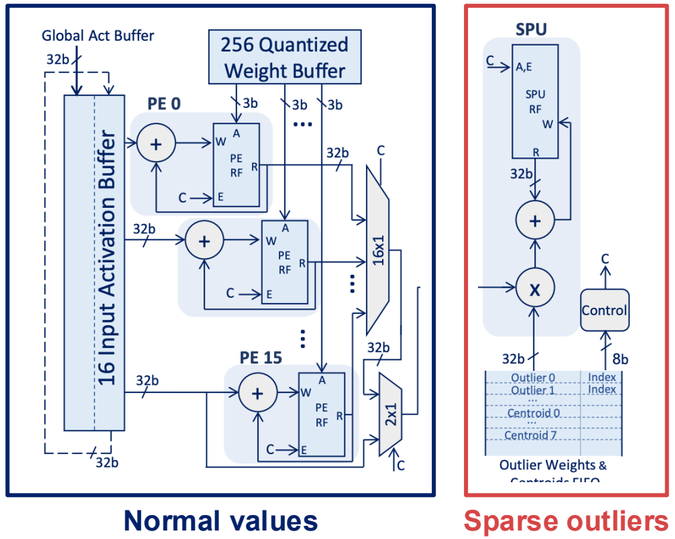
✓ **Runs on existing GPUs**

✗ **Accuracy drops at low bits (results later)**

2 Architecture

Build new hardware for outliers

Dedicated outlier path, e.g. GOBO



Normal values | **Sparse outliers**

~1%

of values are outliers

55%

of PE-array area spent on them

Cheap hardware ⟷ **High accuracy**

- **A classic software–hardware trade-off. Can we get both?**

Our Solution: Outlier-Victim Pair (OliVe, ISCA 2023)

1

Step 1

Identify outliers

Values beyond 3σ are outliers.

2

Step 2

Pair with a neighbor

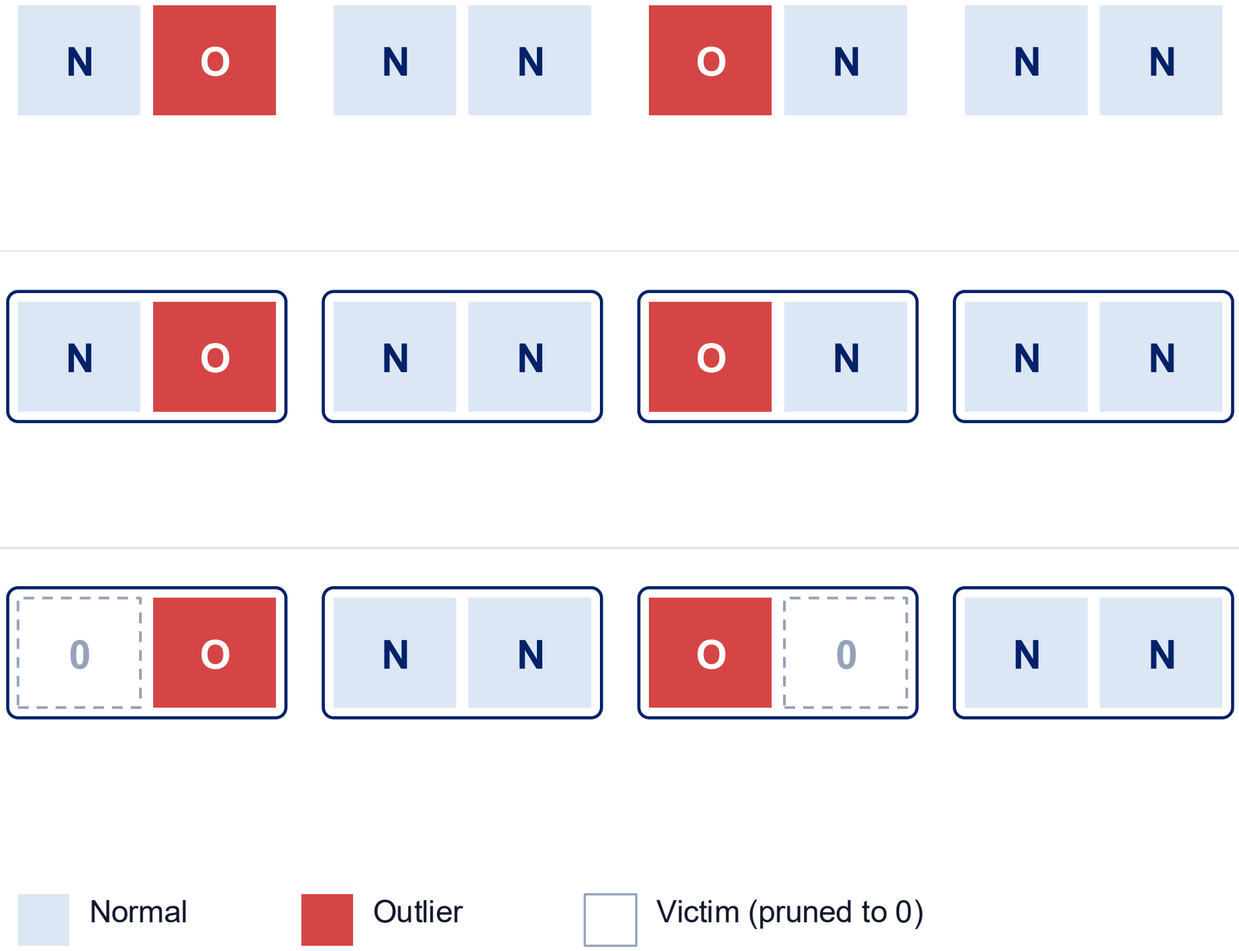
Group adjacent values into pairs.

3

Step 3

Prune the victim

Set the outlier's neighbor to 0; keep the outlier.



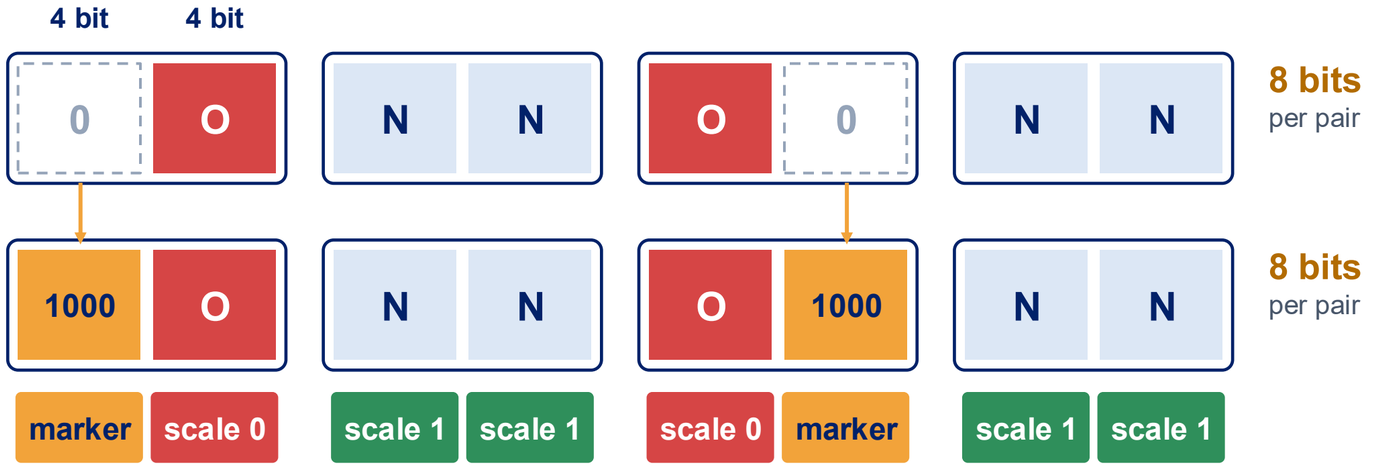
Encoding: Mark the Victim with 1000

After pruning

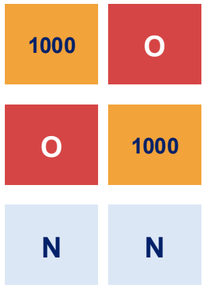
Victim is 0

Encoded

Victim slot stores 1000



The decoder reads the marker



right is an outlier → FP4, scale factor 0

left is an outlier → FP4, scale factor 0

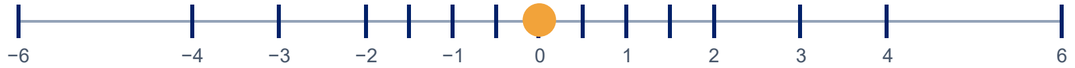
no marker → INT4/FP4, scale factor 1

Why 1000? It is almost free

INT4: $1000_2 = -8$ give up one value of the range



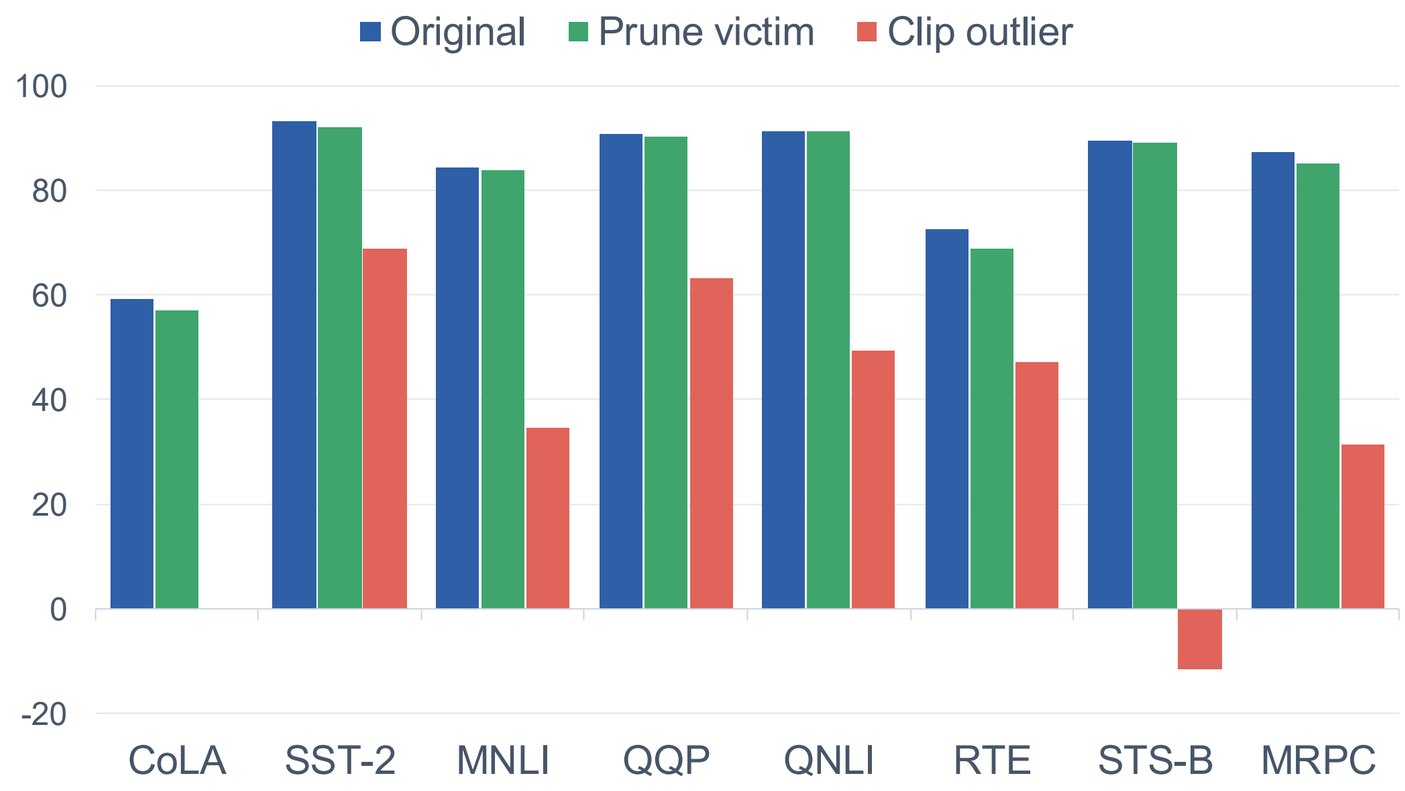
FP4 (E2M1): $1000_2 = -0$ redundant with +0



- **The marker’s position tells the decoder where the outlier is, with no extra bits.**

Victims Are Rare and Cheap to Prune

Accuracy on GLUE (BERT-Base, W+A quantized)



Accuracy barely changes

−1.3

pts avg. when pruning victims

−48

pts avg. when clipping outliers

Very few values are pruned

< 1.2%

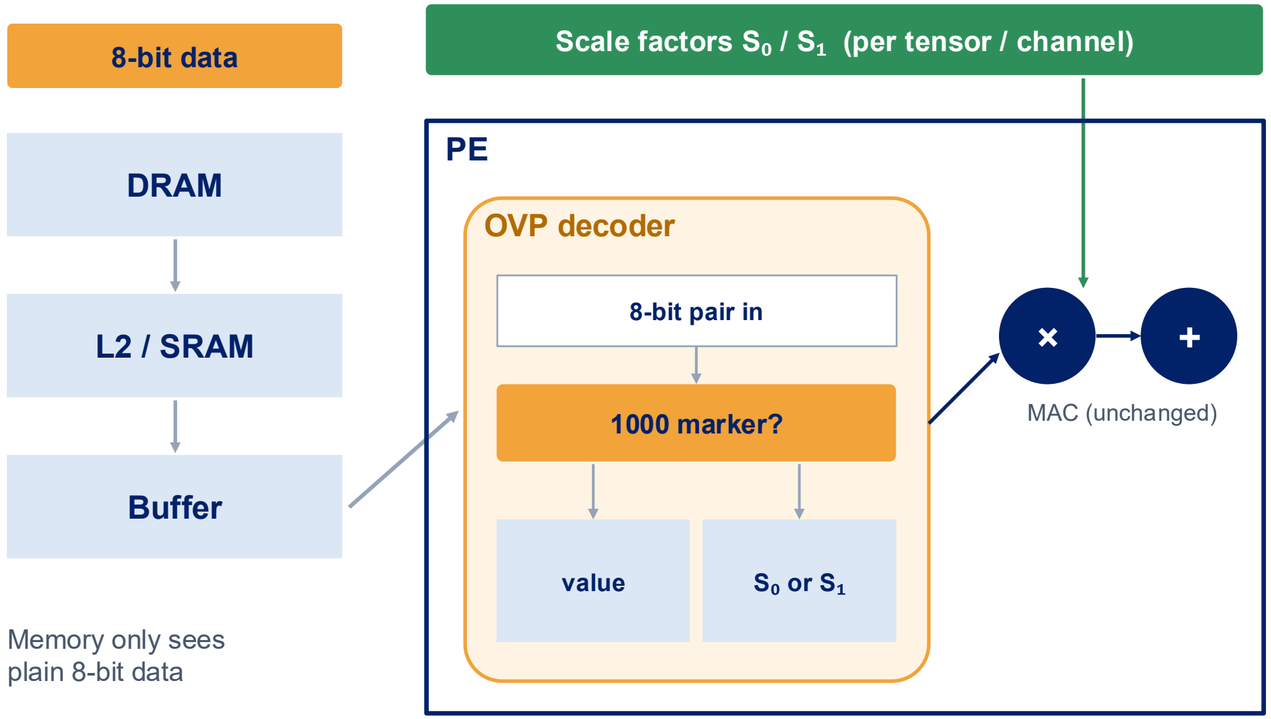
of pairs contain a victim (outlier–normal)

BERT-Base	**0.84%**
BERT-Large	**0.71%**
GPT-2 XL	**1.14%**
OPT-6.7B	**0.64%**

- **Pruning the victim frees bits for the outlier at almost no accuracy cost.**

Outlier: > 3σ, tensor-wise. Accuracy values read from the OliVe (ISCA 2023) figure; pair ratios from its table.

Outlier Decoder: Global Scales, Local Decoding



Global scale, local decoding

Scales are shared per tensor / channel; each pair decodes itself inside the PE.

Transparent to memory

DRAM, caches, and buffers only see aligned 8-bit data: no index, no layout change.

0.25%

decoder area of the GPU die

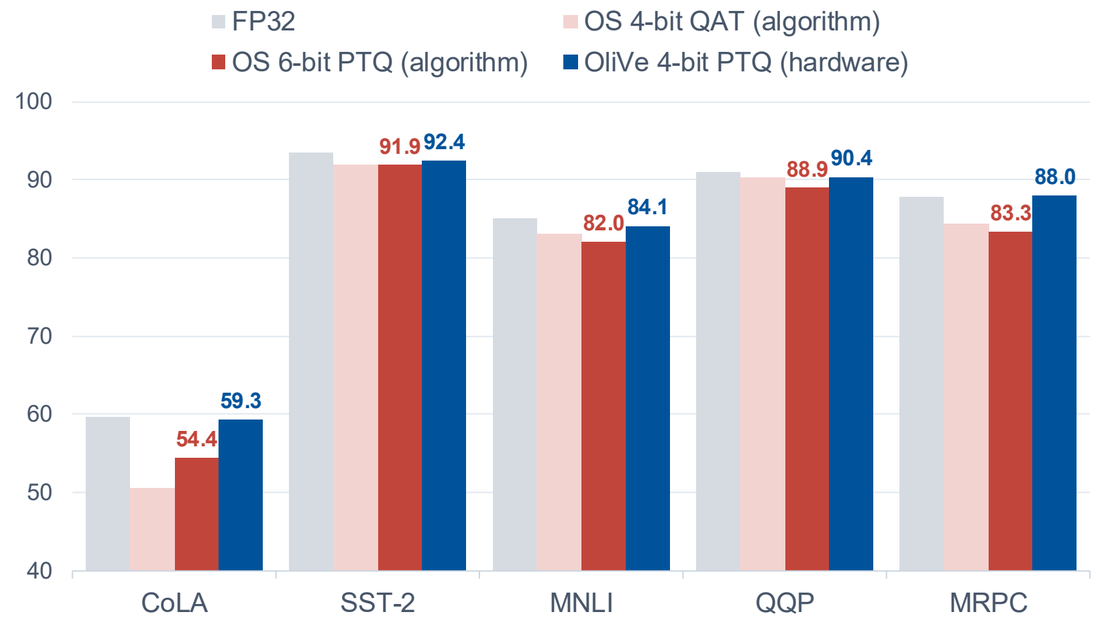
- **A tiny decoder in the PE; the rest of the system just sees 8-bit data.**

Area: OliVe (ISCA 2023), 4-bit decoders on an RTX 2080 Ti die (12 nm).

Hardware Support Beats Software-Only Fixes

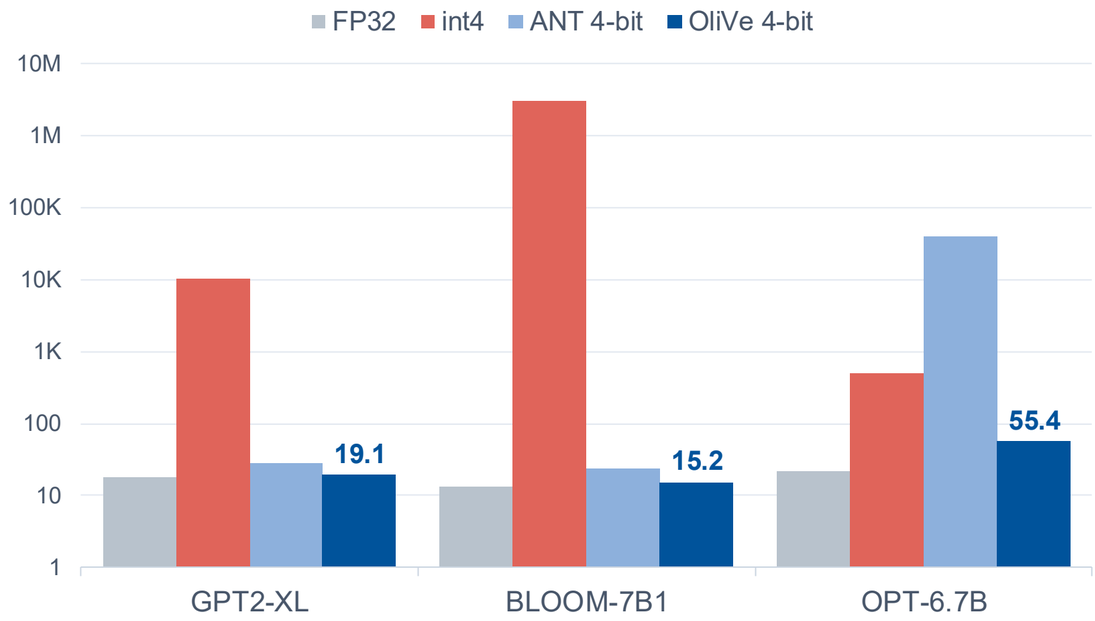
BERT-Base on GLUE (accuracy, higher is better)

OS = Outlier Suppression (NeurIPS 2022)
representative algorithm-only outlier fix, before SmoothQuant



4-bit OliVe beats 6-bit OS on every task
+0.6 to +4.9 points, with 2 fewer bits and no retraining

LLM perplexity on WikiText, 4-bit (lower is better)



Plain int4 collapses; OliVe stays near FP32
BLOOM-7B1: 15.2 vs. 3×10⁶ (int4), FP32 13.1

- **Architecture reaches what algorithms alone cannot: 4-bit OliVe beats 6-bit Outlier Suppression.**

OS: Wei et al., NeurIPS 2022. Data: OliVe (ISCA 2023), GLUE and LLM perplexity tables.

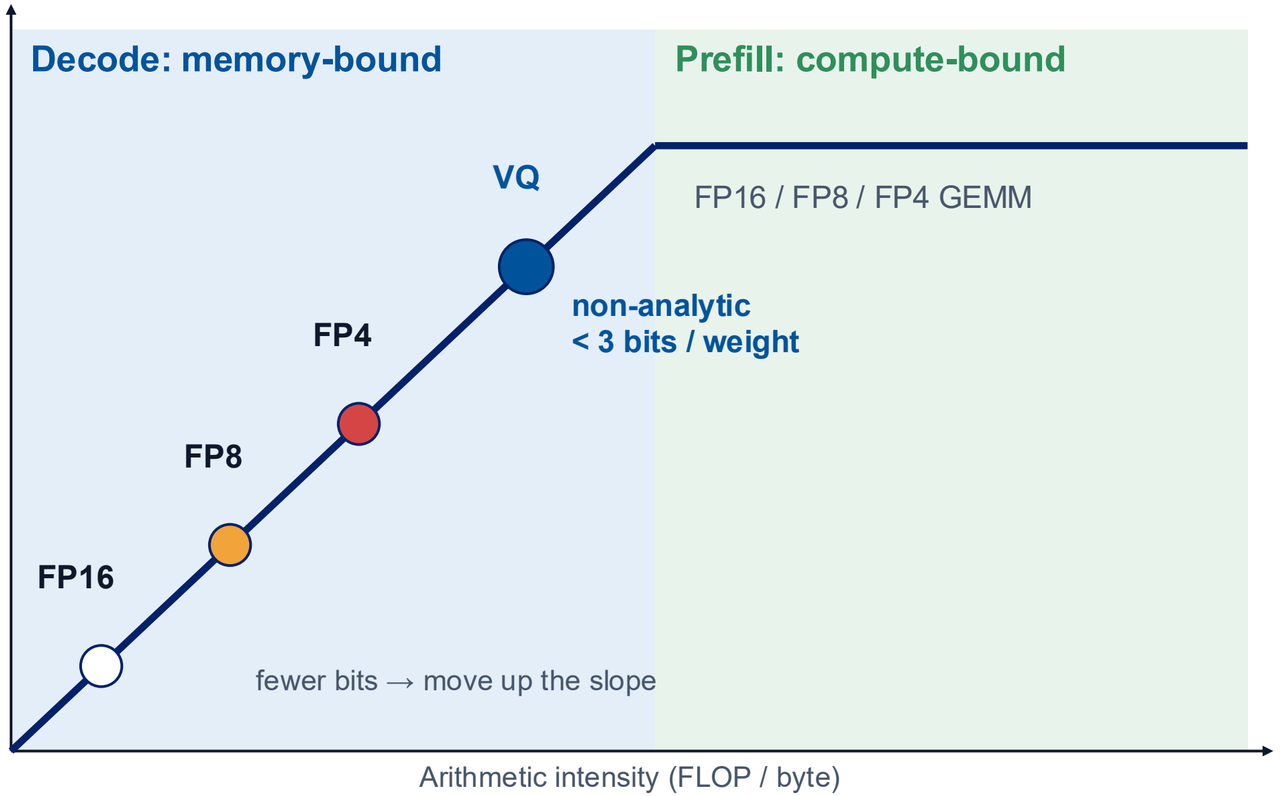
03

Stage 3: Computing Paradigm

From analytic to non-analytic quantization

Decode Is Memory-Bound: Push It to Non-Analytic

Batch-1 decode is GEMV: every weight byte is used once per token.



1 GPU utilization ≈ 0.4%

Batch-1 decode leaves almost all compute idle

2 Speed ∝ bytes per weight

Decode needs memory-efficient weights, not faster math

3 Free to change the format

Prefill and decode run apart, so decode weights can be non-analytic

- **Push decode to non-analytic quantization: the fewest bits per weight wins.**

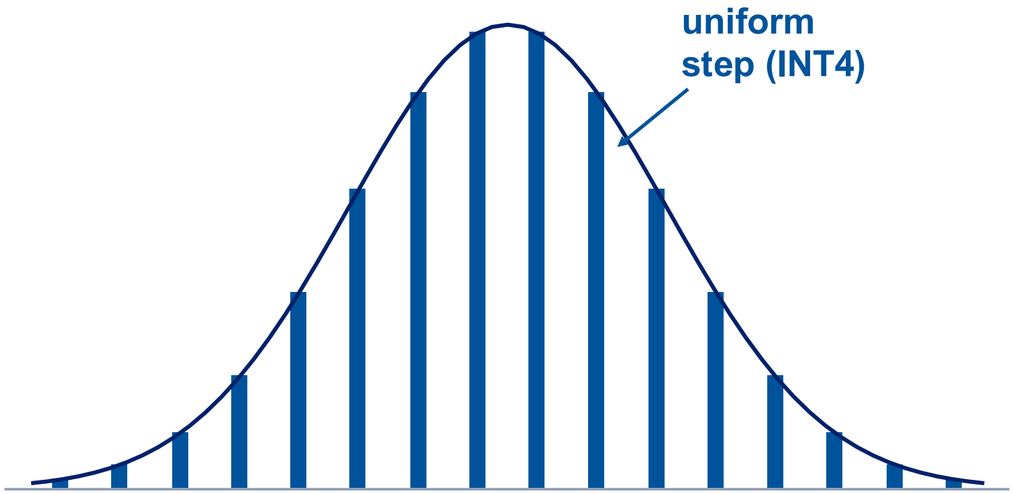
From Analytic to Non-Analytic Quantization

Analytic quantization

- Explicit arithmetic rule

✗ Limited flexibility

✓ Compute directly



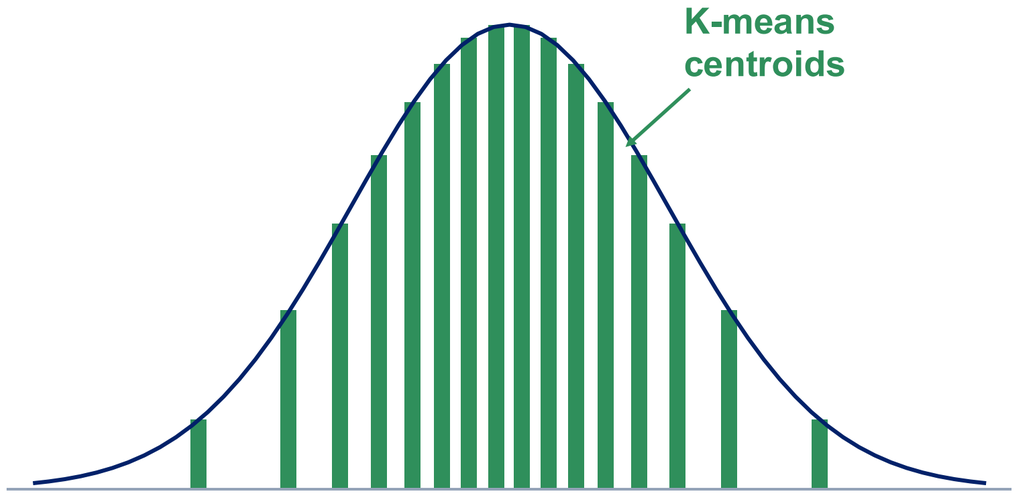
MSE = **1.58**

Non-analytic quantization

- Mapping learned from data (K-means)

✓ Lower quantization error

✗ Needs a codebook to reconstruct



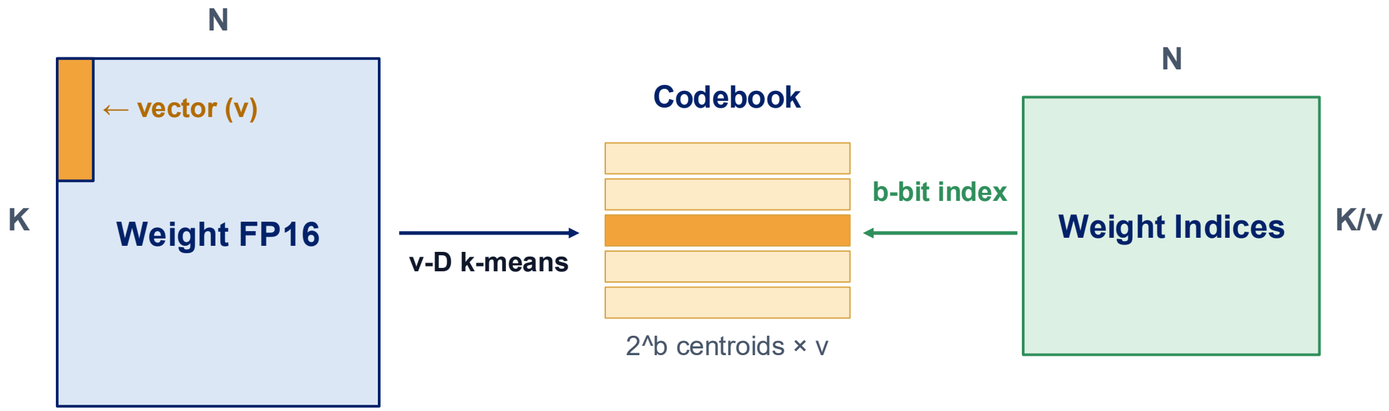
MSE = **0.49**

Learn the levels from data: ~3× lower error at the same 4 bits.

Illustration: 4-bit (16-level) quantization of a Gaussian tensor.

Vector Quantization Pushes Further

Go to higher dimensions: quantize v weights jointly with one index.



v-D k-means codebook

Learn 2^b centroids of dimension v

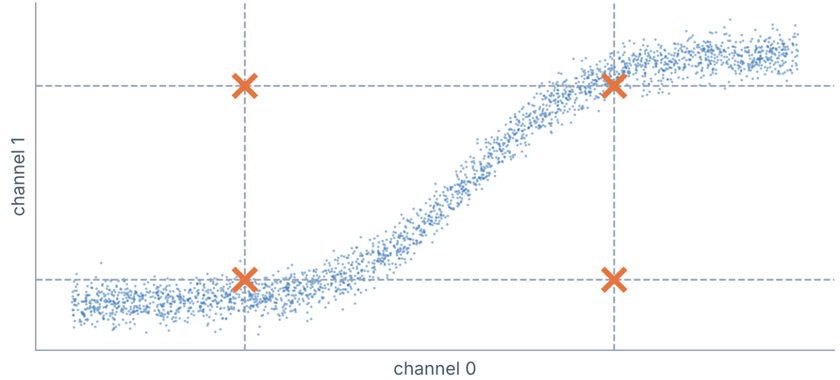
b / v bits per weight

e.g. 8-bit index, v = 8 → ~1 bit / weight

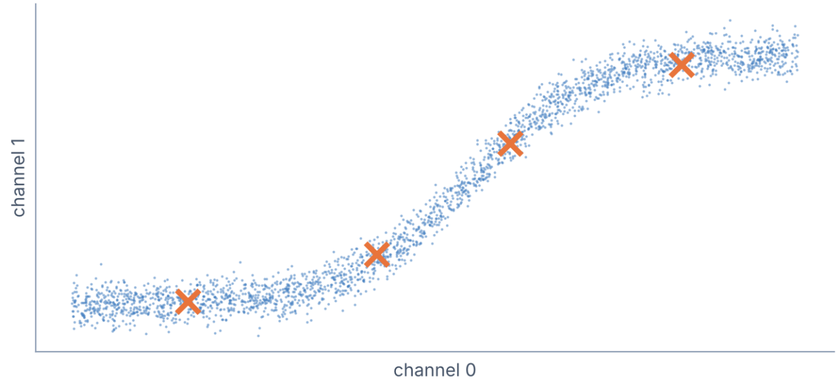
✗ Cost: a much larger codebook

Scalar 3-bit: **8 entries** vs. VQ 8-bit index: **256 entries × v values**

1-D (scalar), MSE = 0.049



2-D VQ, MSE = 0.015



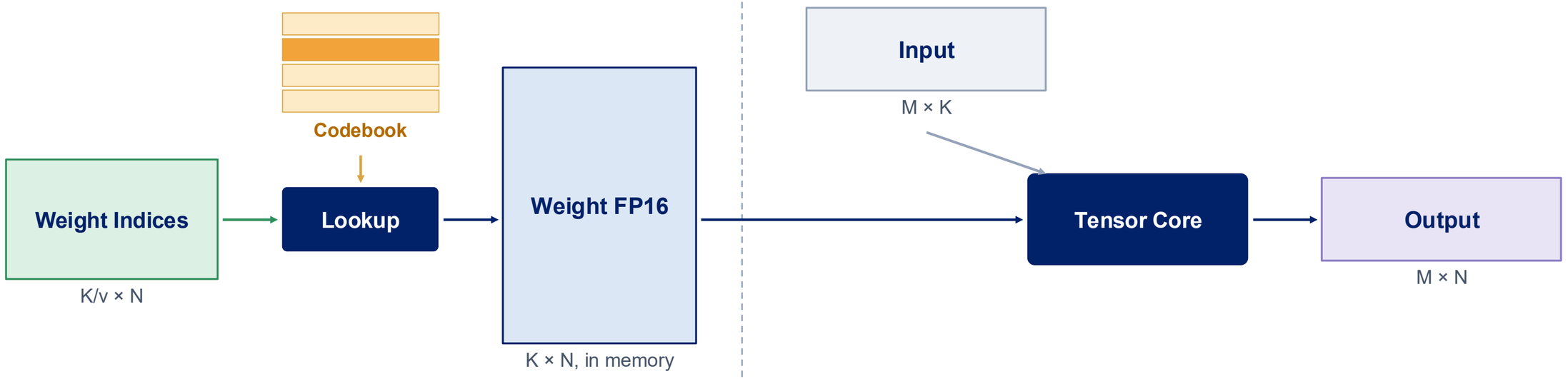
- **Higher dimensions capture the joint distribution: VQ pushes compression to ~2 bits.**

Large Batch: Vector Lookup

Lookup → FP16 weights → Tensor Core GEMM

① Vector lookup (decode weights)

② FP16 GEMM on Tensor Core



Reconstruct once

Decode indices to FP16 weights before the GEMM

Reuse across many tokens

High weight reuse amortizes the lookup cost

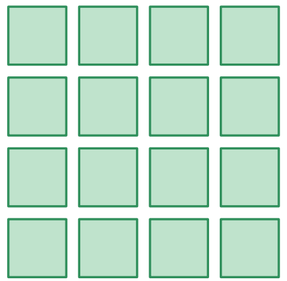
- **Large batch: rebuild the weights, then let the Tensor Core do the GEMM.**

Lookup Before Compute Still Has Two Problems

Conventional VQ decodes weights first, then runs a normal GEMM / GEMV.

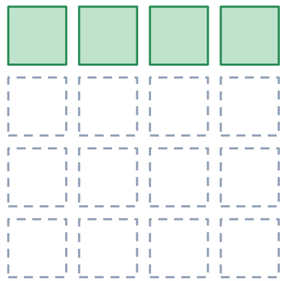
① GEMV underutilization

Prefill (M ≫ 1)



✓ full

Decode (M = 1)



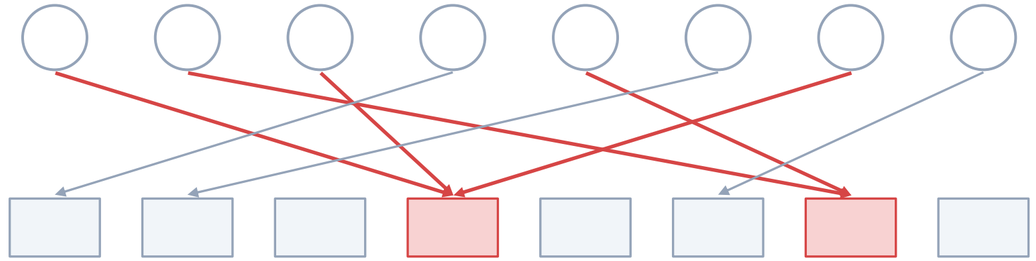
✗ mostly idle

1/16

of a Tensor Core MMA tile is used at M = 1; decoded FP16 weights still feed a memory-bound GEMV

② Bank conflicts in lookups

Lanes (8 of 32 shown)



Banks: set by the weight index → lanes collide at random

2.78

cycles per warp lookup, random indices

1.00

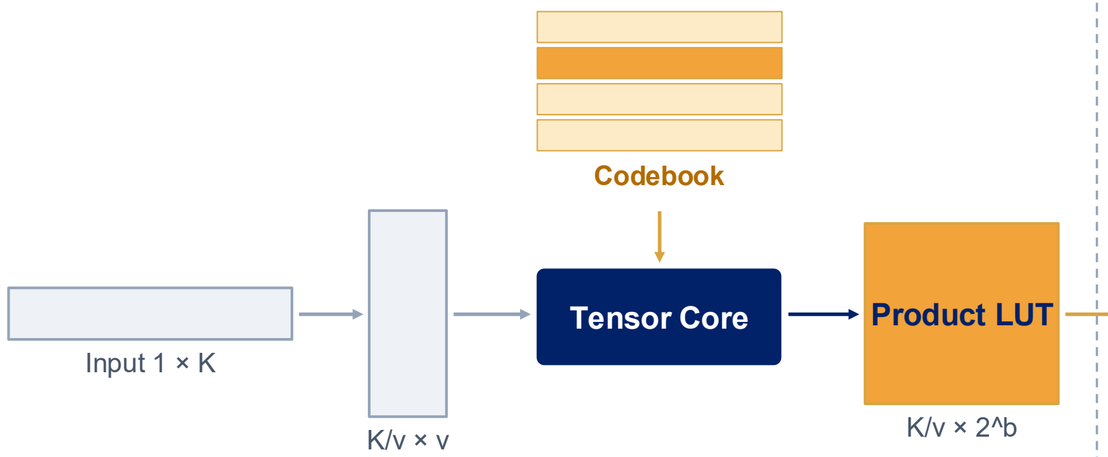
cycles if conflict-free

- **Lookup-before-compute still wastes compute at small batch and stalls on bank conflicts.**

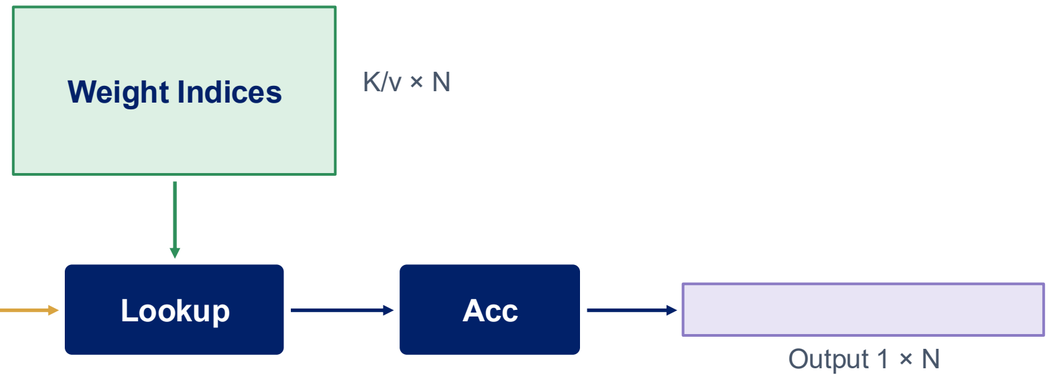
Small Batch: Product Lookup

Multiply first. Look up products, not weights.

① Build product LUT (once per token)



② Look up products and accumulate



No multiplies here: only lookups and adds

P[g,e] = x_g · z_e

Build an input-dependent product LUT per token

y_j = Σ_g P[g, q_gj]

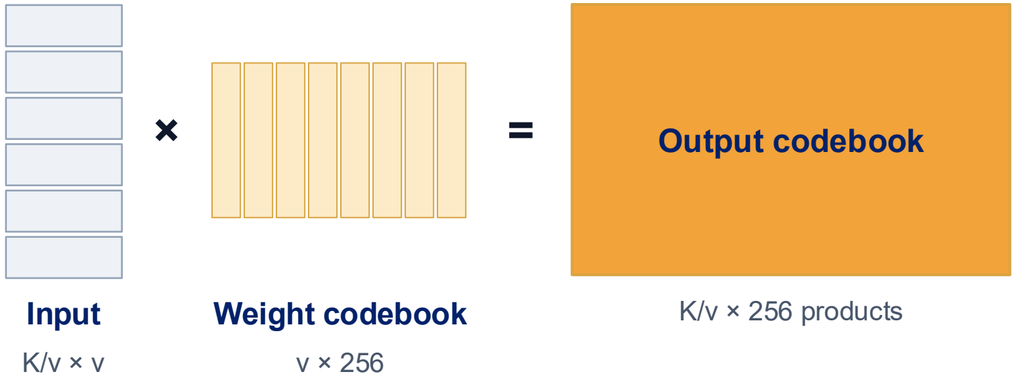
Look up precomputed products and accumulate

- **Small batch (BS 1–2): compute products once per token. This is the idea behind EVA.**

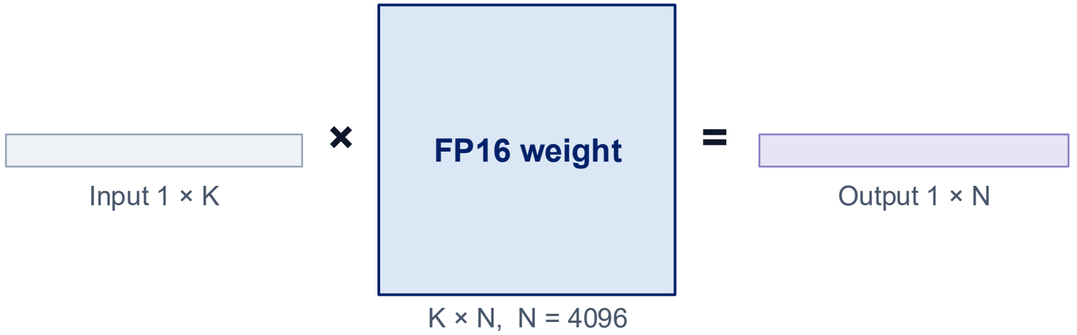
EVA: Compute Before Lookup Fixes Both

Key insight: multiply the input with the codebook first, then look up the results.

EVA: input × codebook (GEMM)



Conventional: input × FP16 weight (GEMV)



16× **fewer multiplications**

Multiply by 256 codebook entries instead of N = 4096 columns

8× **higher utilization (v = 8)**

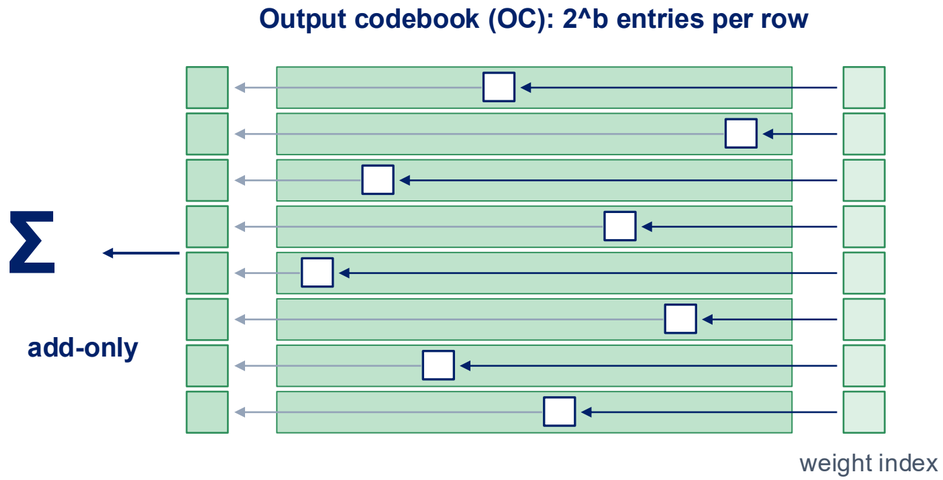
GEMV (M = 1) becomes GEMM (M = K/v): 1/16 → 8/16 of a Tensor Core tile

- **Compute before lookup: fewer multiplies, and a GEMM the hardware can fill.**

EVA: Conflict-Free Lookup, Vector Epilogue

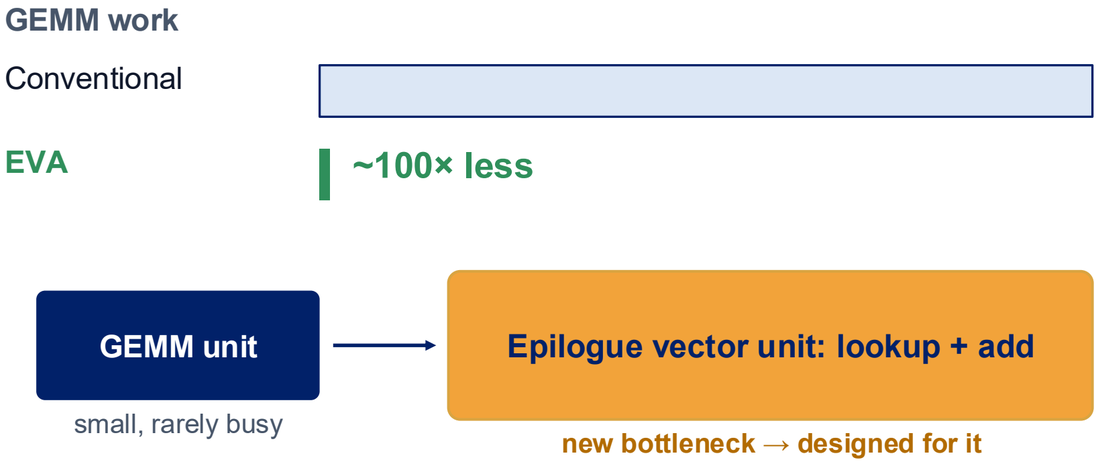
Lookups stay inside one row, and the work moves from GEMM to a vector unit.

③ Conflict-free lookup



- Lookup moves from the weight codebook to the OC
- Each lane reads only its own row → no bank conflicts (1.00 cycle)

④ GEMM is no longer the bottleneck

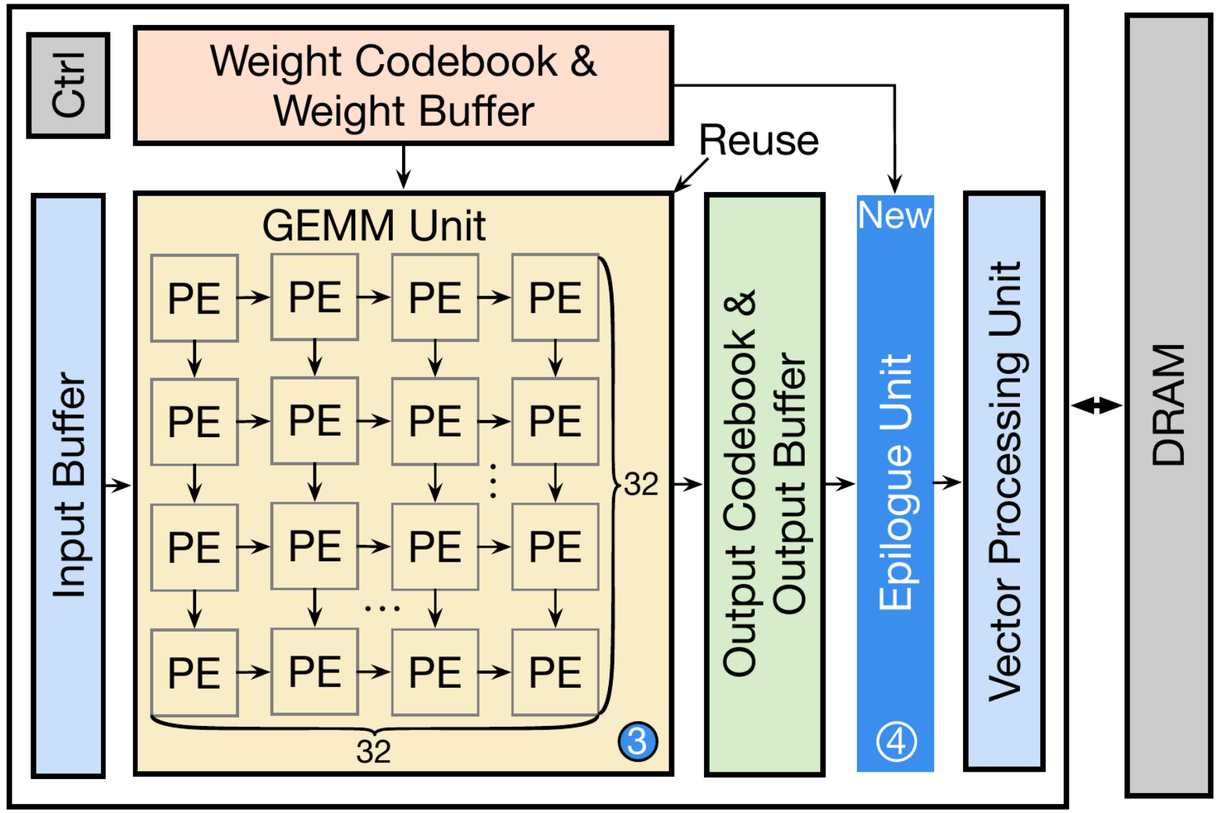


- Bottleneck shifts from GEMM to a vector unit
- A dedicated epilogue unit makes decode much faster

Conflict-free lookups plus a dedicated vector epilogue: fast VQ decode.

EVA Architecture: Systolic-Array GEMM + New Epilogue Unit

We model and simulate the full architecture (ISCA 2026); details skipped here.



What we model

GEMM unit

Systolic array computes input × codebook

Output codebook buffer

Holds the products for lookup

Epilogue unit

Conflict-free lookup + accumulate

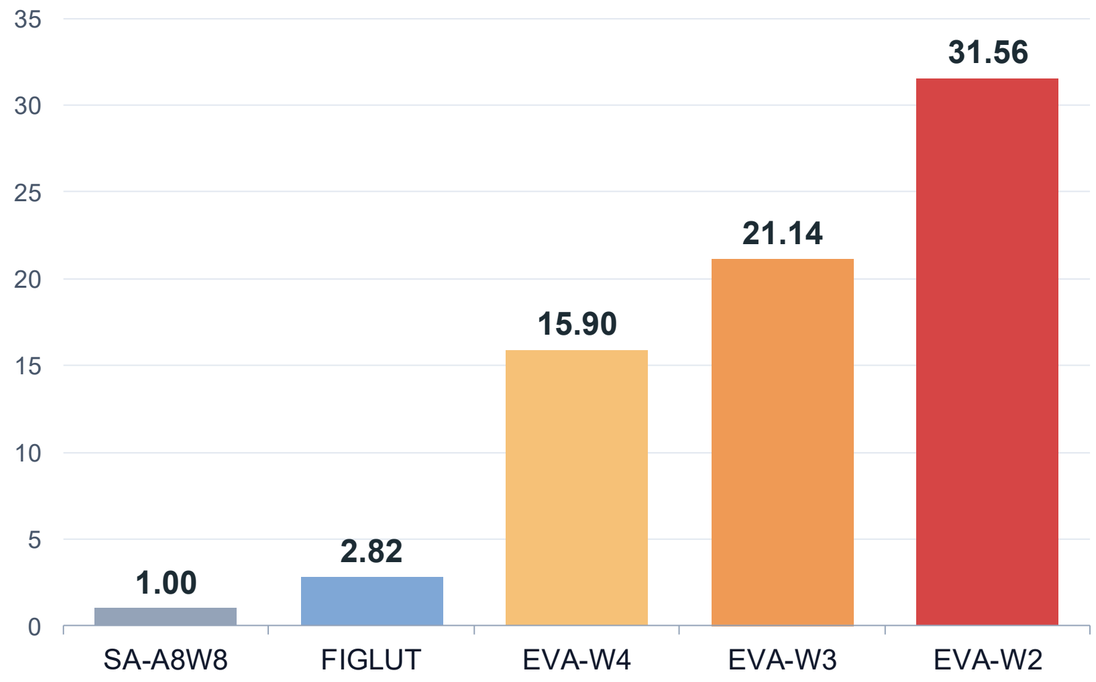
Full-system simulation

End to end, including DRAM and buffers

- **Putting it together: GEMM unit + conflict-free epilogue, simulated end to end.**

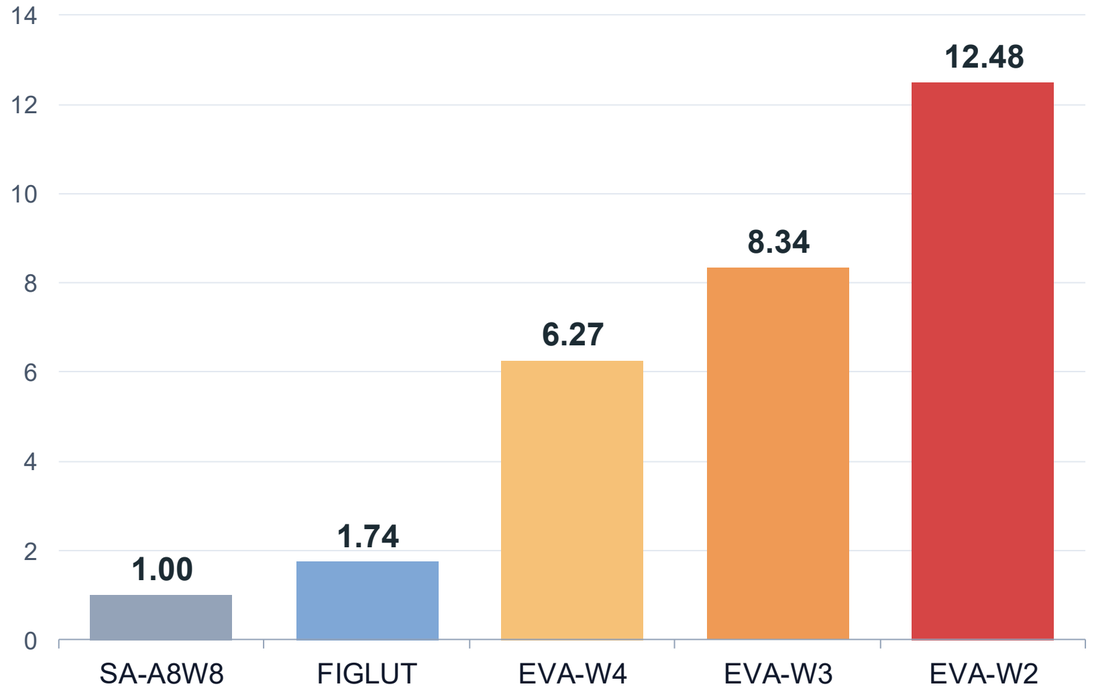
EVA: Up to 31.6× Speedup, 12.5× Energy Efficiency

Speedup (geomean, normalized to SA-A8W8)



5.69× vs. FIGLUT at the same 4 bits

Energy efficiency (geomean, normalized to SA-A8W8)



3.60× vs. FIGLUT at the same 4 bits

- **Lookup-based VQ turns memory-bound decode into a large, real speedup.**

Geomean over LLaMA-1/2/3 models. ANT and FIGNA are close to the baseline and omitted. Source: EVA (ISCA 2026).

Stage 3 · Computing Paradigm

Industry Is Heading the Same Way

NVIDIA Vera Rubin 3-bit scalar LUT inside the Tensor Core

3-bit index

3 | 0 | 7 | 2 | 5 | 1

compressed matrix B

Lookup table

v0	v1	v2	v3
v4	v5	v6	v7

8 representative values

Tensor Core

inline decode + MMA

Where it goes next

Today

Scalar LUT

3-bit, 8 entries · NVIDIA Rubin

Our work

Vector quantization

Compute before lookup · EVA

Next

Richer quantization

Higher-dim VQ, lattice, learned codes

- **Our design aligns with, and runs ahead of, where GPUs are heading.**

NVIDIA Technical Blog · Rubin GPU architecture (2026)

Conclusion

Three Stages of Quantization Co-Design

01 Numerics

Match the data type to the distribution

ANT MICRO 2022
SQuant ICLR 2022
M2XFP ASPLOS 2026
Tensor-Core Numerics HPCA 2027

02 Architecture

Co-design granularity, formats, and outliers

OliVe ISCA 2023
Ecco ISCA 2025
Gossamer MICRO 2026
MANT HPCA 2025

03 Computing paradigm

Compute efficiently from compact codes

EVA ISCA 2026
Vortex ASPLOS 2027
Transitive Array ISCA 2025
Platinum ASP-DAC 2026
VQ-LLM HPCA 2025

Bold: first / corresponding author **Gray:** collaborations

Our survey: it brings dozens of hardware–software co-design works into one map, and we hope it sparks ideas for future architectures.

Where does quantization architecture go next?

Non-analytic quantization: lookup-based compute from compact codes.

Thank You

Questions & Discussion