Is Unstructured Sparsity Free on a Wafer? A Cycle-Accurate Study of Sparse Decode GEMV on the Cerebras Wafer-Scale Engine

Ali Asaria
Transformer Lab

Tony Salomone
Transformer Lab

Deep Gandhi*
Transformer Lab

Abstract

Wafer-scale engines keep model weights resident in on-chip SRAM, which suggests that the memory-bound decode matrix-vector product (GEMV) at the heart of autoregressive inference might turn compute-bound, and that unstructured weight sparsity might translate into proportional cycle savings without the accuracy tax of structured pruning. We test three such ideas with cycle-accurate measurements on the Cerebras SDK fabric simulator, using small processing-element (PE) rectangles and a single format-consistent observable: cycles per kernel. First, unstructured sparsity is nearly free, but only with the right kernel: a compressed kernel that iterates over non-zeros makes cycles fall almost linearly with sparsity (a per-non-zero cost within 2.5% of the ideal, about 8× fewer cycles at 90% sparsity relative to the same kernel at full density), clearly beating the analytic GPU curve (about 3.6×), whereas a naive per-element zero-skip loop does not linearize because it still pays loop and branch overhead on every position. Second, at the single-PE cycle level, structured 2:4 sparsity buys nothing: at matched density the per-non-zero cost is identical across patterns (to within 0.02%), so the accuracy cost of forcing an N:M pattern is unjustified there. Third, the distributed mesh version does not scale as hoped: at fixed problem size, adding processing elements reduces cycles only sub-linearly (a strong-scaling exponent near 0.25 against an ideal of 1.0), consistent with a collective-reduction cost that grows with the mesh, so decode GEMV on the mesh appears communication-bound. A per-PE SRAM ceiling, which we observe directly when a large tile fails to compile, is what forces large problems onto more PEs in the first place. The sparse-linearity and single-PE pattern-independence results hold cleanly; the wafer's roofline advantage is argued architecturally rather than measured, and near-ideal distributed scaling does not hold.

1 Introduction

Autoregressive decoding is dominated by matrix-vector products (GEMV): each generated token multiplies a $[1\times d]$ activation by weight matrices $[d\times d]$. On GPUs this operation is deeply memory-bound [2]. Its arithmetic intensity is only 0.5 to 1.0 floating-point operations per byte, far below the roofline ridge points of modern accelerators (about 153 for an A100, 295 for an H100), so the hardware spends most of its time waiting on High Bandwidth Memory (HBM) rather than computing. Two ideas follow naturally from this fact. First, if the weights lived in on-chip memory

*Corresponding author: deep@lab.cloud

instead of HBM, the operation would no longer pay the bandwidth wall. Second, if a large fraction of the weights are zero, and the hardware can skip those zeros cheaply, then cycles should fall in proportion to sparsity, and the arrangement of the non-zeros should not matter.

The Cerebras Wafer-Scale Engine (WSE) makes the first idea concrete: it holds weights in per-processing-element (per-PE) SRAM, roughly 48 KB per core, with very high on-chip bandwidth, so weight-stationary decode has no HBM traffic [6]. Whether the second idea holds is an empirical question that, to our knowledge, no prior work answers with cycle-accurate measurements. The sparse-accelerator literature repeatedly *assumes* efficient per-PE zero-skip without isolating its per-cycle cost [17, 15], and reports throughput or wall-clock rather than cycles [6, 7]. A finer observable is what settles whether skipping a zero is genuinely cheap.

We study three claims on the WSE using the Cerebras SDK fabric simulator, which reports exact cycle counts. We deliberately use small PE rectangles and a single format-consistent metric, cycles per kernel, and we verify every kernel output against a numerical reference. Because our strongest evidence is at small, single-PE scale, we are careful throughout to separate what is measured from what is argued. The claims are:

1. **Sparse linearity.** We test whether unstructured-sparse decode GEMV cycles fall about linearly with sparsity on the WSE, and whether the slope is steeper than the analytic GPU curve. See §5.1.

2. **Wafer roofline and distributed scaling.** We test whether the decode-GEMV roofline ridge sits to the left of the GPU ridge (a structural argument from resident weights), and how distributed cycles fall with processing-element count. See §5.3.

3. **Pattern independence.** We test whether, at equal density, structured N:M sparsity gives any cycle advantage over unstructured sparsity. See §5.2.

Our contributions are the measurements themselves and an honest, split verdict. Sparse linearity and single-PE pattern independence hold cleanly. The wafer-roofline claim splits: the ridge-left-of-GPU half is an architectural argument we do not measure, and the distributed-scaling half fails, because the mesh kernel scales sub-linearly. We report the mechanism behind each result, including the negative one, since the contrast between a kernel that linearizes and one that does not is itself the finding.

2 Related Work

Weight-stationary wafer-scale execution. The load-bearing mechanism for both the sparse-linearity and roofline claims is keeping weights resident in PE-local SRAM. WaferLLM shows a decode GEMV that is memory-bound on an A100 becoming compute-bound on the WSE-2, with weights held in 48 KB per core, and reports large speedups over a single-GPU GEMV [6]. Persistent-state dataflow on FPGAs is the clean structural analog: hold recurrent state on chip to cross the memory-to-compute boundary [4], and FRED frames the same weight-stationary premise at wafer scale [13].

Unstructured-sparse GEMM and the load-as-compressed pattern. The field's answer to unstructured sparsity is to load a compressed tile and compute densely within it. MUSTAFAR uses a bitmap format [7], MACKO uses delta-encoded indices and approaches memory-bandwidth peak

[11], and Flash-LLM reconstructs tiles from a sparse representation [16]. A complementary line builds streaming dataflows where the cycle count tracks non-zeros directly, on FPGA and spatial fabrics [8, 17, 3]. On a wafer, the decompression step should be free because the weights are already PE-local, which is precisely the sparse-linearity hypothesis we test.

Roofline analysis for decode. Classifying a kernel by arithmetic intensity against the hardware ridge point is the standard tool for the memory-versus-compute question. RooflineBench catalogues ridge points across devices and shows decode workloads trapped in the memory-bound regime [2]; roofline-based scaling analyses and the prefill-versus-decode SRAM ceiling give the $T = \max(F/\pi, M/\beta)$ formulation and the memory-bound decode baseline we aim to overturn [14, 1, 18].

Structured sparsity and its accuracy cost. GPU structured 2:4 speedups are sublinear even in the best case, roughly 1.5 to 1.8× against a 2× ideal, from index and gather overhead, and shrink further at small batch [10, 5, 9]. Structured pruning also carries a real accuracy cost, for example a sharp drop on a hard reasoning benchmark under strict 2:4 [9], and the literature often conflates equal-density with equal-accuracy comparisons [12]. This grounds the premise of our pattern-independence claim: if structure buys no cycles on the wafer, its accuracy cost is unjustified there. For realistic comparisons, masks derived from magnitude-and-activation (Wanda-style) pruning, rather than purely random masks, are the field standard [15].

Distributed GEMV on a mesh. The reduction across a PE mesh, via a tree allreduce or a mesh collective, is the relevant primitive for the strong-scaling study, following prior wafer-scale designs [6], and its cost grows with mesh dimension; we observe that growth in aggregate (§5.3), though we do not isolate the collective from the compute. Prior work reports this primitive at full-wafer scale; our contribution is a cycle-accurate strong-scaling sweep at small, reproducible sizes, including the point at which per-PE memory forces the problem wider.

3 Method

Problem and metric. We measure the decode GEMV $y = Ax + b$ with $A \in \mathbb{R}^{d\times d}$, a single observable throughout: the exact cycle count reported by the Cerebras SDK fabric simulator. The simulator is cycle-accurate and bit-deterministic by construction, so a given kernel and input produce the same count on every run; no repeated runs are needed to stabilize a cycle count. All host-to-device transfers are 32-bit, so activations, weights, and the packed cycle timestamps are all `float32`. Every kernel output is checked against a NumPy reference. We use an absolute tolerance of 10^{-2}, which is loose relative to `float32` precision but is a coarse correctness gate (catching a wrong gather index or a dropped accumulate), not an accuracy measurement; the reported quantity is cycles, not error.

Single-PE kernels. We compare two ways to exploit weight sparsity on one PE, which turn out to be the crux of the sparse-linearity result:

- **Approach A, per-element zero-skip.** A dense $K\times N$ loop that tests each weight and skips the multiply-accumulate when the weight is zero. The loop body still executes for every position.

- **Approach B, compressed (compressed sparse column, CSC).** The weights are stored in a compressed column format, and the inner loop runs exactly once per non-zero. Positions that are zero are never visited.

A dense kernel (no skipping) is the baseline, and a no-skip control confirms that, absent skipping, cycles are independent of sparsity. Both single-PE kernels run with weights resident in the PE, so the A-versus-B contrast isolates the effect of the kernel, not of residency; residency is what removes the reload cost that would otherwise be present on an HBM-backed device, an argument we make rather than measure here.

Distributed mesh kernel. For the strong-scaling study we port a two-dimensional collectives GEMV to a $P \times P$ PE rectangle (a *mesh*): the activation is scattered across the top row, broadcast down each column, multiplied against the local weight tile, reduced across each row, and gathered. We add per-PE timestamping around the whole pipeline and report end-to-end latency as the span from the earliest start to the latest finish across all PEs. This yields a total cycle count per configuration, not a compute-versus-collective breakdown. Each PE holds a $d/P \times d/P$ sub-block of A (its *tile*), so the tile footprint is $(d/P)^2$ `float32`, which must fit within the PE's local SRAM.

GPU roofline baseline. We compare against an analytic GPU roofline rather than GPU hardware, following standard practice for this class of study [2]. Under the roofline model $T = \max(F/\pi, M/\beta)$, the decode GEMV (arithmetic intensity 0.5 to 1.0 FLOP/byte, well below every GPU ridge point we consider) is memory-bound, so its runtime scales with bytes moved; the analytic unstructured sparse speedup is then bounded by $1/(1-s)$ at sparsity s and reduced by kernel efficiency. The published curve we use (for example about 3.6× at 90% sparsity, well under the ideal 10×, reflecting index and gather overhead) is the reference against which the WSE curve is judged. This is a slope comparison against an idealized memory-bound GPU model, not a like-for-like measured-kernel comparison.

4 Experimental Setup

Substrate. Experiments run on the Cerebras SDK (version 2.10.0) fabric simulator, hosted on a cloud instance; the simulator is CPU-bound, so the host GPU sits idle and is irrelevant to the cycle counts. The kernels are compiled with `cslc` for the WSE-2 architecture and simulated with `cs_python`.

Sizes and memory. A WSE-2 PE holds roughly 48 KB of local SRAM. A single-PE dense tile therefore caps at 64^2 `float32` (16 KB); a 128^2 tile (64 KB) does not fit, which is why 256^2 and larger problems are mesh territory. The single-PE sparsity study uses $d = 64$; the mesh study uses $d \in \{512, 1024, 2048\}$ over PE rectangles from 8×8 to 32×32, with a 2×2 validation configuration at $d = 64$.

Sparsity masks and the sparsity basis. We use two mask families for the sparsity sweep: purely random (unstructured) and a magnitude-and-activation (Wanda-style) pruning proxy [15], which reflects realistic LLM weight statistics. Both are single seeded draws; we report their agreement below but do not interpret sub-percent differences between the two families, since each is one realization and the non-zero count varies with the draw. Throughout, we report sparsity by *measured* non-zero

Table 1: Sparse decode GEMV cycles versus sparsity, compressed kernel (approach B), random mask, $d = 64$, single PE. All points lie exactly on cycles $= 4{,}083 + 39.0\,\text{nnz}$ ($R^2 = 1.00$).

Sparsity (measured)	Non-zeros (nnz)	Cycles
0%	4096	163,827
26.0%	3029	122,214
50.1%	2042	83,721
75.9%	988	42,615
90.0%	411	20,112

count, not nominal target: the nominal $\{25, 50, 75\}$% levels correspond to actual sparsities of 26.0%, 50.1%, and 75.9% at $d = 64$. For the pattern-independence study we additionally construct an exact 2:4 structured mask and a constant-fan-in mask at matched density (to within one mask draw).

Slope definition. We define the sparse-linearity *slope* as the marginal cycle cost per additional non-zero (the coefficient of a linear cycles-versus-nnz fit over the five sparsity points), normalized by the kernel's cycle cost per matrix element at full density. A slope of 1.0 means each skipped zero saves a full element of dense work. Because cycle counts are exact, these fits are exact: the five points lie on the line to integer precision ($R^2 = 1.00$), so the slope has no fit uncertainty; its only variability is through the mask-dependent non-zero sequence, which we address by reporting both mask families.

Gates. The pre-registered success criteria are: for sparse linearity, a slope at least 0.8 and clearly steeper than the analytic GPU curve; for the wafer claim, a ridge point to the left of the GPU ridge and cycles falling with PE count; for pattern independence, a cycle ratio within about 10% at equal density. Total simulator compute was about 1.8 instance-hours across four jobs.

5 Results

5.1 Sparse linearity: unstructured sparsity is nearly free, with the right kernel

Table 1 reports cycles versus sparsity for the compressed kernel (approach B), for the random-mask family; the fit is cycles $\approx 4{,}083 + 39.0 \cdot \text{nnz}$, where nnz is the number of non-zeros, and every tabulated point lies on this line exactly ($R^2 = 1.00$). The marginal cost is 39.0 cycles per non-zero against a full-density per-element cost of 40.0 cycles, giving a slope of 0.975 for random masks; the Wanda-style masks give 0.982. We treat these as equal within single-seed sampling and do not read the 0.007 difference as a mask-family effect. Both are well above the 0.8 criterion. Relative to the same kernel at 0% sparsity, the speedup at 90% sparsity is 8.1× (random) to 8.6× (Wanda-style); all speedups in this paper are kernel-relative in this way, so the compressed kernel's 8.1× is against its own dense point, not against the separate dense baseline. The analytic GPU unstructured curve reaches only about 3.6× at 90% and is nearly flat at low sparsity (about 1.02× at 25%), while the WSE curve is linear from 0%. The sparse-linearity claim holds.

The mechanism is the kernel, not the hardware alone. Table 2 contrasts the two single-PE kernels. The naive per-element zero-skip loop (approach A) fits cycles $\approx 112{,}876 + 8.0 \cdot \text{nnz}$: its fixed term equals the dense baseline because the loop still iterates every $K \times N$ position and only

Table 2: Single-PE kernel contrast, $d = 64$. The marginal per-non-zero cost is what differs; the 41.0 figure in Table 3 is the same 39.0 marginal cost averaged with the fixed overhead.

Kernel	Fixed term	Marginal cyc/nnz	Slope	Speedup @90%
Dense (no skip)	112,874 (flat)	n/a	0	1.00×
Approach A (zero-skip)	112,876	8.0	0.225	1.25×
Approach B (compressed)	4,083	39.0	0.975	8.1×

skips the multiply-accumulate, not the loop and branch overhead. Its slope is therefore only 0.225, and its speedup at 90% sparsity is 1.25×, worse than the GPU baseline. Linearity requires iterating over non-zeros (approach B), not skipping within a dense loop (approach A). A no-skip control kernel is perfectly flat at 112,874 cycles across all sparsities, confirming that the slope in approach B is real skipping rather than an artifact. Compressed-iteration beating dense-skip is an algorithmic fact that would also hold on a CPU or GPU; what the wafer supplies is that the weights are already resident, so the compressed kernel pays no reload cost to set against the savings. That residency benefit is an architectural argument, not something the A-versus-B contrast alone isolates.

A caveat at low sparsity: approach B costs more per non-zero (39.0 cycles) than the dense inner loop (about 27.5 cycles per multiply-accumulate) because of gather and indirection, so it beats the dense kernel only above about 32% sparsity (and beats approach A above about 14%). The small fixed overhead relative to the non-zero range is specific to $d = 64$; we report the linearity as a kernel-level trend at this size rather than a size-independent law.

5.2 Pattern independence: structured sparsity buys nothing at the single-PE cycle level

Table 3 compares three mask patterns at matched density (to within one mask draw) under the compressed kernel. The pattern-invariant quantity is the per-non-zero cost, which is identical across all three to within 0.02% (41.00 versus 40.99 cycles per non-zero; this 41.0 average folds in the fixed overhead and equals the 39.0 marginal cost of Table 2 plus that overhead). The raw unstructured-to-2:4 cycle ratio is 0.9972, a 0.28% difference, but that difference is almost entirely explained by the 0.29% difference in non-zero count (2042 versus 2048), not by the pattern: read on cycles per non-zero, the pattern effect is negligible. Either way the result is far inside the 10% criterion, so at the single-PE cycle level structured N:M offers no advantage over unstructured, and its accuracy cost is not worth paying. We measured this directly at 50% density; the pre-registered criterion also named 75%, which we did not run with a structured mask and instead infer from the same non-zero-count cost model. We scope the claim accordingly: it is a single-PE, cycle-cost statement. The place structure could still matter on this hardware is the multi-PE regime, through per-PE non-zero balance, which we do not test here (see Limitations).

5.3 Wafer roofline and distributed scaling: a split verdict

The architectural (ridge) half: argued, not measured. The decode GEMV has arithmetic intensity 0.5 to 1.0 FLOP/byte, below every GPU ridge point (153, 295, and 328 for the A100, H100, and RTX-4090 respectively), so it never reaches compute peak on a GPU. Because the WSE holds weights in SRAM and has no HBM wall, its effective ridge sits to the left, and it can service this low-intensity workload from compute. We emphasize that this half is an architectural argument

Table 3: Cycles at matched ≈ 50% density for three sparsity patterns, compressed kernel, $d = 64$. Cost depends on non-zero count only; per-non-zero cost is invariant to within 0.02%.

Pattern	Non-zeros	Cycles	Cycles/nnz
Unstructured	2042	83,721	41.00
2:4 structured	2048	83,955	40.99
Constant fan-in	2048	83,955	40.99

Table 4: Mesh strong-scaling: cycles versus processing-element count at fixed problem size. Scaling is monotonic but strongly sub-linear (exponent $\alpha \approx 0.25$ against ideal 1.0). The largest configuration does not compile: its tile exceeds PE memory.

d	PE rect.	PEs	Cycles
512	8×8	64	14,126
512	16×16	256	9,355
512	32×32	1024	7,379
1024	16×16	256	18,453
1024	32×32	1024	13,121
2048	32×32	1024	n/a (PE memory exceeded)

from the resident-weights premise: we do not report a measured WSE ridge point or a sustained on-chip-bandwidth figure, so we treat ridge-left-of-GPU as assumed by construction rather than as a result of this study.

The scaling half: sub-linear. Table 4 reports the mesh strong-scaling sweep. At fixed $d = 512$, going from 64 to 256 processing elements (4×) reduces cycles 1.51×, and from 64 to 1024 (16×) only 1.91×; at $d = 1024$, 4× more PEs yields 1.41×. Expressed as a strong-scaling exponent α (cycles $\propto \text{PEs}^{-\alpha}$), these give $\alpha \approx 0.23$ to 0.30, an order of magnitude below the ideal $\alpha = 1.0$. Scaling is monotonic but far from ideal. This is consistent with the mesh collective (broadcast down a column, reduce across a row) carrying a cost that grows with the mesh side, so that adding PEs shrinks the per-PE tile compute while inflating the collective cost. We stress that the measurements are end-to-end cycle totals; we do not separate the collective from the compute, and with three points at $d = 512$ we cannot rule out a fixed serial floor as a contributing cause or locate a precise knee. What the data establish is sub-linear strong scaling; the communication-bound reading is the most consistent mechanism, not an isolated measurement.

The memory ceiling. The final configuration, $d = 2048$ on a 32×32 mesh, does not compile: the tile is 64^2 `float32` (16 KB) per PE, which together with the kernel's collective buffers overflows the PE's local SRAM (the linker reports running out of PE data memory). The same problem at $d = 1024$ on the same mesh, with a 32^2 tile (4 KB), fits and runs. We did not separately quantify the collective-buffer footprint, so this specific threshold reflects this kernel's allocation as much as a hard hardware limit; a leaner allocation might push it slightly. The qualitative point is robust, though: a 16 KB tile plus working buffers does not fit a 48 KB PE, so at $d = 2048$ the problem must spread onto more PEs. By the scaling result above, that is exactly the regime where the collective

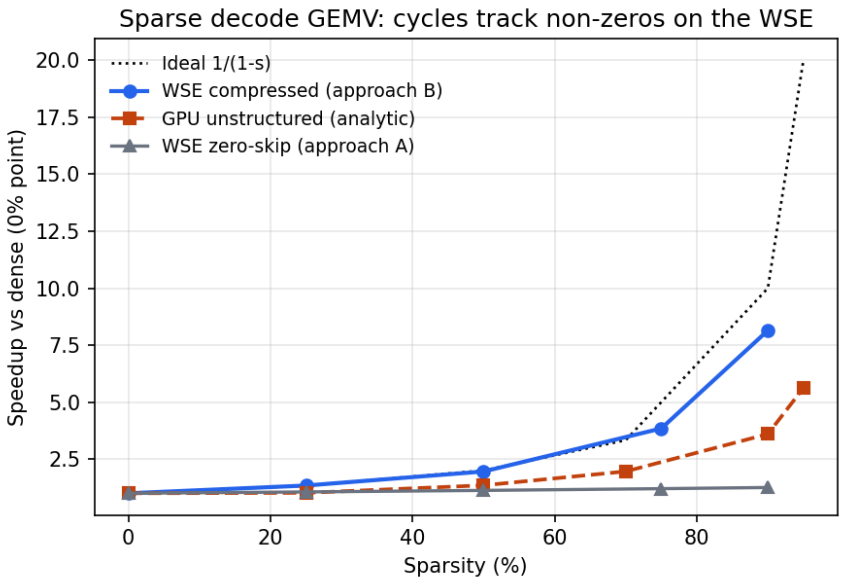

Figure 1: Approach A barely moves; the WSE compressed kernel (approach B) tracks non-zeros almost linearly and stays steeper than the analytic GPU unstructured curve from 0%. Speedup is relative to each kernel at 0% sparsity.

cost dominates. We note that the largest mesh we actually simulate is 32×32 (1024 PEs); the 64×64 configuration a spread-out $d = 2048$ would need is a projection beyond our measured range, not a simulated point.

Figure 1 overlays the sparse-linearity curves against the analytic GPU curve; Figure 2 plots the mesh strong-scaling data against the ideal.

6 Discussion

The three results form a coherent picture of where a wafer-scale engine helps decode inference and where it does not. Sparsity is close to free when the kernel iterates over non-zeros, and this is consistent with the wafer's resident weights: there is no reload cost to swamp the savings. Pattern independence follows from the same mechanism at the single-PE cycle level, since a compressed kernel that visits only non-zeros cannot tell whether those non-zeros are arranged in a structured pattern or not. Together, these argue that on a wafer, at least at the single-PE level, one should prefer unstructured pruning, taking the full cycle benefit without paying the accuracy cost that a structured pattern imposes.

The distributed result is a caution. The single-PE story does not compose trivially into a whole-wafer story, because the primitive that stitches PEs together, the collective reduction, carries a cost that grows with the mesh. For a memory-bound operation made compute-light by resident weights, that collective cost is what plausibly becomes the new bottleneck. This does not contradict the architectural argument that the wafer ridge sits left of the GPU ridge; it says that realizing that advantage at scale looks like a communication problem, and that pinning it down requires the compute-versus-collective decomposition we did not perform.

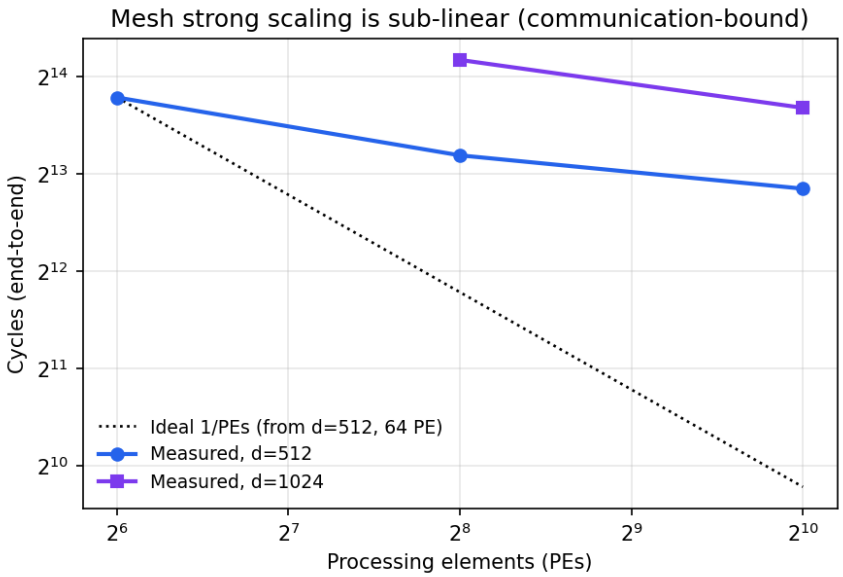

Figure 2: Measured mesh cycles fall far below the ideal 1/PEs line (exponent $\alpha \approx 0.25$ against 1.0), consistent with a collective-reduction cost that grows with the mesh side.

7 Limitations

Construct validity. Every measurement is a simulated cycle count, not wall-clock time on physical silicon, and uses a naive scalar `float32` inner loop (about 27.5 cycles per multiply-accumulate) that is not micro-optimized. All claims are therefore about slopes and ratios, not absolute throughput against a real GPU. The GPU baseline is an analytic roofline, not measured hardware, and the arithmetic-intensity range cites an fp16 case at its upper end even though our runs are `float32`; we did not run an fp16 robustness sweep.

External validity. The single-PE sparsity study is at $d = 64$ and the mesh study at $d \leq 2048$ over rectangles up to 32×32, which are small relative to a full wafer. The sparse-linearity and pattern-independence results are kernel-level trends measured at a single problem size, and the small-fixed-overhead argument behind linearity is d-dependent; extrapolation to full-wafer inference is bounded by the per-PE SRAM ceiling that forces pipeline parallelism, and we label these results as trends rather than whole-system guarantees. The pattern-independence comparison is measured directly at 50% density; the pre-registered 75% point is inferred from the non-zero-count cost model, not measured, and the multi-PE regime, where per-PE non-zero balance could make structure matter, is not tested.

Internal validity. The pre-registered wafer criterion asked for a ridge left of the GPU ridge and for cycles proportional to 1/PEs. We do not measure the WSE ridge point or on-chip bandwidth, so the ridge half is argued, not established. The measured scaling is monotonic but strongly sub-linear ($\alpha \approx 0.25$), so the scaling half does not meet the criterion. We report this split, and the fact that the communication-bound mechanism is inferred from end-to-end totals rather than an isolated collective measurement, rather than selecting the favorable half. Each mask family is a single seeded

draw; we did not run multiple seeds, so the random-versus-Wanda slope difference is not interpreted.

8 Availability

All artifacts for this study, including the WSE kernels (the compressed single-PE kernel and the mesh collectives kernel), the input-generation pipeline, the analytic GPU roofline scripts, and the raw cycle-count data, are available on request from the corresponding author.

9 Conclusion

On a wafer-scale engine with resident weights, unstructured sparsity in decode GEMV is close to free when the kernel iterates over non-zeros: cycles fall almost linearly with sparsity (slope 0.98, about 8× at 90% relative to the same kernel at full density), beating the analytic GPU curve, and at the single-PE cycle level the arrangement of the non-zeros does not matter, so structured pruning's accuracy cost is unjustified there. The distributed picture is less favorable: mesh decode GEMV scales sub-linearly, consistent with a communication-bound collective, and a per-PE memory ceiling forces large problems onto wider meshes. The wafer looks structurally well suited to sparse decode; measuring its roofline advantage and realizing it across many processing elements both remain open.

References

[1] Hannah Atmer, Yuan Yao, Thiemo Voigt, and Stefanos Kaxiras. Prefill vs. Decode Bottlenecks: SRAM-Frequency Tradeoffs and the Memory-Bandwidth Ceiling. 2025. URL https://arxiv.org/abs/2512.22066.

[2] Zhen Bi, Xueshu Chen, Luoyang Sun, Yuhang Yao, Qing Shen, Jungang Lou, and Cheng Deng. RooflineBench: A Benchmarking Framework for On-Device LLMs via Roofline Analysis. 2026. URL https://arxiv.org/abs/2602.11506.

[3] Chao Fang, Wei Sun, Aojun Zhou, and Zhongfeng Wang. Efficient N:M Sparse DNN Training Using Algorithm, Architecture, and Dataflow Co-Design. 2023. URL https://arxiv.org/abs/2309.13015.

[4] Neelesh Gupta, Peter Wang, Rajgopal Kannan, and Viktor K. Prasanna. A Persistent-State Dataflow Accelerator for Memory-Bound Linear Attention Decode on FPGA. 2026. URL https://arxiv.org/abs/2603.05931.

[5] Daniel Haziza, Timothy Chou, Dhruv Choudhary, Luca Wehrstedt, Francisco Massa, Jiecao Yu, Geonhwa Jeong, Supriya Rao, Patrick Labatut, and Jesse Cai. Accelerating Transformer Inference and Training with 2:4 Activation Sparsity. 2025. URL https://arxiv.org/abs/2503.16672.

[6] Congjie He, Yeqi Huang, Pei Mu, Ziming Miao, Jilong Xue, Lingxiao Ma, Fan Yang, and Luo Mai. WaferLLM: Large Language Model Inference at Wafer Scale. 2025. URL https://arxiv.org/abs/2502.04563.

[7] Donghyeon Joo, Helya Hosseini, Ramyad Hadidi, and Bahar Asgari. Mustafar: Promoting Unstructured Sparsity for KV Cache Pruning in LLM Inference. 2025. URL https://arxiv.org/abs/2505.22913.

[8] Rubens Lacouture, Nathan Zhang, Ritvik Sharma, Marco Siracusa, Fredrik Kjolstad, Kunle Olukotun, and Olivia Hsu. FuseFlow: A Fusion-Centric Compilation Framework for Sparse Deep Learning on Streaming Dataflow. 2025. URL https://arxiv.org/abs/2511.04768.

[9] Jaeseong Lee, Seung won Hwang, and Samyam Rajbhandari. SpenseGPT: Practical One-shot Pruning Enabling Sparse and Dense GEMMs for LLM Inference. 2026. URL https://arxiv.org/abs/2606.10445.

[10] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, Yu Wang, and Guohao Dai. Large Language Model Inference Acceleration: A Comprehensive Hardware Perspective. 2024. URL https://arxiv.org/abs/2410.04466.

[11] Vladimír Macko and Vladimír Boža. MACKO: Sparse Matrix-Vector Multiplication for Low Sparsity. 2025. URL https://arxiv.org/abs/2511.13061.

[12] Egor Maximov, Yulia Kuzkina, Azamat Kanametov, Alexander Prutko, Aleksei Goncharov, Maxim Zhelnin, and Egor Shvetsov. From 2:4 to 8:16 sparsity patterns in LLMs for Outliers and Weights with Variance Correction. 2025. URL https://arxiv.org/abs/2507.03052.

[13] Saeed Rashidi, William Won, Sudarshan Srinivasan, Puneet Gupta, and Tushar Krishna. FRED: Flexible REduction-Distribution Interconnect and Communication Implementation for Wafer-Scale Distributed Training of DNN Models. 2024. URL https://arxiv.org/abs/2406.19580.

[14] Luoyang Sun, Jiwen Jiang, Yifeng Ding, Fengfa Li, Yan Song, Haifeng Zhang, Jian Ying, Lei Ren, Kun Zhan, Wei Chen, Yan Xie, and Cheng Deng. Hardware Co-Design Scaling Laws via Roofline Modelling for On-Device LLMs. 2026. URL https://arxiv.org/abs/2602.10377.

[15] Chenpeng Wu, Qiqi Gu, Heng Shi, Jianguo Yao, and Haibing Guan. Samoyeds: Accelerating MoE Models with Structured Sparsity Leveraging Sparse Tensor Cores. 2025. URL https://arxiv.org/abs/2503.10725.

[16] Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. 2023. URL https://arxiv.org/abs/2309.10285.

[17] Zhewen Yu, Sudarshan Sreeram, Krish Agrawal, Junyi Wu, Alexander Montgomerie-Corcoran, Cheng Zhang, Jianyi Cheng, Christos-Savvas Bouganis, and Yiren Zhao. HASS: Hardware-Aware Sparsity Search for Dataflow DNN Accelerator. 2024. URL https://arxiv.org/abs/2406.03088.

[18] Ted Zadouri, Hubert Strauss, and Tri Dao. Hardware-Efficient Attention for Fast Decoding. 2025. URL https://arxiv.org/abs/2505.21487.