NO MAN IS AN ISLAND: OVERLAPPING PRUNING FOR NEAR-UNSTRUCTURED QUALITY AT 2:4 EFFICIENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Semi-structured 2:4 sparsity is the dominant fine-grained pruning pattern on modern ML accelerators, yet for large language models it degrades quality far more than unstructured pruning at the same 50% sparsity. We trace this gap to mask *rigidity*: 2:4 partitions weights into disjoint four-element windows, so both selectors draw from the same four inputs, severely constraining the feasible mask space. We propose **Overlapping Pruning (OP)**, an algorithm–hardware co-designed sparsity structure that overlaps neighboring selector windows, enlarging the mask space while preserving 2:4's selectors, 2-bit index, and datapath. At 4-, 8-, and 16-bit, OP-1D and OP-F2D cost at most $1.09\times$ 2:4's core area, 0.94–$1.04\times$ its delay, and 0.93–$1.16\times$ its power. OP-2D additionally yields transposable masks. Feasible masks are the independent sets of a transversal matroid, so a saliency-ordered greedy pass with a per-candidate feasibility check is exact for every OP pattern; our kernels generate OP-F2D masks within $1.6\times$ the 2:4 mask-generation cost. One-shot OP-F2D with Wanda++ increases perplexity by at most 10.4% over unstructured 50%, versus up to 59.8% for 2:4. OP-1D attains lower perplexity than 4:8 and OP-F2D stays within 1.1% of 8:16; N:M patterns cost 1.2–$4.2\times$ 2:4's core area and 1.7–$3.4\times$ its delay. OP-F2D matches or improves on reported results from learned-mask and factorized 2:4 methods at a fraction of their pruning cost, with no mask training or added parameters. By relaxing mask geometry alone, OP-F2D closes 75.8–88.8% of the unstructured-to-2:4 quality gap at 2:4's hardware cost.

1 INTRODUCTION

Pruning large language models (LLMs) requires balancing model quality against efficient execution. Semi-structured 2:4 sparsity—which retains exactly two weights in each disjoint group of four consecutive weights—is the most widely supported fine-grained pattern with native hardware support, offering up to $2\times$ sparse matrix multiplication throughput on NVIDIA and AMD GPUs (Mishra et al., 2021; Advanced Micro Devices, Inc., 2025; Cai et al., 2024). This regularity comes at a cost: at 50% sparsity, 2:4 pruning incurs substantially greater quality degradation than unstructured pruning. One-shot post-training methods such as SparseGPT, Wanda, and Wanda++ only partially narrow this gap (Frantar & Alistarh, 2023; Sun et al., 2024; Yang et al., 2025), while learned-mask methods such as MaskLLM (Fang et al., 2024) close more of it at a cost of 1,280 A100 GPU hours for Llama 2 7B.

We argue that the root cause of this gap is the *rigidity* of the 2:4 mask family, an issue orthogonal to the saliency criterion: each disjoint four-weight group is forced to retain exactly two weights. Consequently, a group must discard two weights even when all four are more salient than weights retained elsewhere—a constraint no improvement in scoring can overcome. Our central idea is to change *selector connectivity*, i.e. which candidate positions each selector can access, while holding the number of selectors and their input count fixed.

We therefore propose *Overlapping Pruning (OP)*, an algorithm–hardware co-designed sparsity structure that enlarges the feasible mask space while preserving what makes 2:4 attractive: a compact 2-bit index per retained weight and an unchanged datapath. Instead of disjoint windows, neighboring selectors share candidates through overlapping windows. We instantiate three forms: *OP-1D*, overlapping along the row dimension; *OP-2D*, overlapping across rows and columns and admitting transposable masks; and *OP-F2D*, which flattens OP-2D connectivity onto a single row, retaining mask flexibility while simplifying inference hardware.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Overlapping couples selection decisions: a weight claimed by one selector becomes unavailable to its neighbors, so masks can no longer be chosen independently per window. We cast mask construction as a sparse bipartite assignment between selectors and candidates, and show that matchable partial supports are the independent sets of a *transversal matroid*. By the Rado–Edmonds theorem, a saliency-ordered greedy pass is exact for every OP pattern (Edmonds, 1971). Correctness is thus pattern-independent, and only the cost of the feasibility test varies, which we specialize per pattern.

OP's gains come from an axis—selector connectivity—orthogonal to what existing methods optimize: which weights to keep, in what order, and at what precision. Our contributions are:

- **Overlapping sparsity patterns.** We introduce Overlapping Pruning, which overlaps neighboring selector windows to enlarge the feasible mask space while reusing 2:4's selector count, 2-bit index, and datapath. Quality follows directly: rewiring selector inputs alone lets OP-1D beat 4:8 perplexity on all ten models under all three scorers and OP-F2D match 8:16 within 1.1%, closing 75.8–88.8% of the unstructured-to-2:4 gap. OP-2D admits transposable masks for sparse training (future work).
- **Exact mask selection.** Feasible OP masks are the independent sets of a transversal matroid, so a saliency-ordered greedy pass is exact for every pattern and mask selection reduces to one feasibility test. A general primal–dual assignment solver produces all model-quality results reported here and certifies optimality by duality. Specializing only the feasibility test with an incremental union–find check is 13.1–$15.5\times$ faster at identical retained saliency, within $1.6\times$ the cost of 2:4 top-k mask generation. Per-tile optimality is global because candidate windows never cross tile boundaries.
- **Empirical and hardware validation.** We evaluate OP with Wanda, RIA, and Wanda++ on Llama, Qwen, Gemma, and OPT models from 6.7B to 32B. OP-1D and OP-F2D match 4:8 and 8:16 perplexity respectively, and OP-F2D matches or outperforms reported PermLLM, MaskLLM, and ARMOR results without end-to-end retraining. Place-and-route confirms the hardware claim: OP-1D within $1.01\times$ 2:4 on all metrics, and OP-F2D within $1.09\times$ core area, $1.04\times$ delay and $1.16\times$ power, whereas 4:8 and 8:16 cost 1.2–$4.2\times$ core area, 1.7–$3.4\times$ delay, and 1.6–$2.7\times$ power.

2 Background and Related Work

Sparsity granularity and hardware support. Neural network pruning dates back to the foundational works of LeCun et al. (1989) and Hassibi & Stork (1992), and sparse compression for LLMs involves a persistent trade-off between quality, hardware efficiency, and cost. Unstructured pruning tolerates high sparsity with little quality loss (Han et al., 2015; 2016; Frankle & Carbin, 2019), but irregular masks rarely yield proportional wall-clock gains on commodity hardware (Hooker, 2021). Structured pruning is hardware efficient and has seen steady quality improvements for LLMs (Ma et al., 2023; Xia et al., 2024; Li et al., 2025; Ashkboos et al., 2024), yet its coarse granularity limits the sparsity-quality trade-off. Semi-structured N:M sparsity occupies the middle ground, retaining N nonzeros in every group of M consecutive weights (Zhou et al., 2021). Wider ratios such as 4:8 and 8:16 relax the per-block constraint and recover quality, but require more index bits and wider selection logic; consequently, accelerators implement only the narrowest useful case, 2:4, which delivers up to $2\times$ theoretical sparse matrix multiplication throughput (Mishra et al., 2021; Advanced Micro Devices, Inc., 2025; Cai et al., 2024). Wider ratios therefore require alternative execution strategies, motivating more expressive mask families at a comparable selector budget.

Post-training LLM pruning. One line of work improves 2:4 mask quality without end-to-end retraining, scoring weights by reconstruction error, activation awareness, row and column importance, or local gradient refinement (Frantar & Alistarh, 2023; Das et al., 2024; Sun et al., 2024; Zhang et al., 2024; Yang et al., 2025). A second learns the structure directly: MaskLLM (Fang et al., 2024) optimizes masks through differentiable relaxation and PermLLM (Zou et al., 2025) learns channel permutations, at costs from roughly 10 (PermLLM) to over 1,200 (MaskLLM) A100 GPU hours on Llama 2 7B. Closest to our setting, ARMOR (Liu et al., 2026) relaxes the format itself, factorizing each weight matrix into a 2:4 core wrapped by block-diagonal matrices; this recovers substantial accuracy but adds parameters ($\sim 5\%$), a non-standard inference path, and 16 A6000 GPU hours for 7B models. All of these compensate for the structural rigidity of 2:4—scoring masks more carefully, learning them, or padding the format with auxiliary structure—rather than removing it.

Overlapping Pruning. We show that this rigidity is not intrinsic: OP-F2D attains 8:16-like perplexity while retaining 2:4's selector count, fan-in, and index width, without additional inference-time parameters. Empirically, OP matches or improves on both learned-mask methods and ARMOR

(Table 5) while composing with existing scoring criteria, so it stands to benefit from future advances in saliency scoring and reconstruction as well as from post-training quantization (Frantar et al., 2023; Lin et al., 2024), the dominant complementary compression path. Software approaches such as NM-SpMM (Ma et al., 2025) and VENOM (Castro et al., 2023) instead target flexible sparse formats through different execution strategies; adapting them to OP is left to future work.

3 Overlapping Pruning

Semi-structured 2:4 sparsity is an ordered pair of selector decisions within a four-weight window. Writing the retained positions in ascending order as $0 \le a < b \le 3$, the left selector never needs position 3 and the right never needs position 0: one input of each four-input selector is redundant. Yet 2:4 still wires both selectors to all four positions, admitting only $\binom{4}{2} = 6$ masks per window.

The 2:4 mask family is therefore more restrictive than its own hardware requires, and the same holds for any N:M pattern. We propose Overlapping Pruning (OP), which repurposes these redundant inputs to reach neighboring weights. OP retains 2:4's primitives—four-input selectors and a 2-bit index per kept weight—and changes only *connectivity*: instead of routing every selector to the same disjoint block of four, adjacent selectors observe overlapping candidate windows. Selector count, fan-in, and index width are unchanged; we evaluate the physical cost of the changed wiring in §3.6. We instantiate three forms: OP-1D (§3.1), overlapping along rows and backward compatible with 2:4; OP-2D (§3.2), overlapping along both axes to obtain natively transposable masks; and OP-F2D (§3.3), which linearizes OP-2D into a single row in row-major order for inference hardware.

3.1 Overlapping-1D

Construction. Consider a row tile of width L divisible by 4. Semi-structured 2:4 makes $L/2$ selector decisions—two four-input selections per four-column window—with both selectors in a window observing the same four columns. OP-1D keeps the same $L/2$ selectors but rewires their inputs so adjacent selectors observe overlapping windows (Figure 1a). Let C_i denote the candidate window of selector i; for boundary and interior selectors respectively,

$$C_0 = \{0, 1, 2, 3\}, \qquad C_{L/2-1} = \{L{-}4, L{-}3, L{-}2, L{-}1\}, \qquad C_i = \{2i{-}1,\ 2i,\ 2i{+}1,\ 2i{+}2\}.$$

A feasible mask selects exactly one column from each C_i with no column selected twice, keeping exactly $L/2$ weights per tile and hence the same 50% sparsity as 2:4. OP-1D is a strict superset of 2:4 masks (each interior C_i retains the three needed 2:4 positions and adds one).

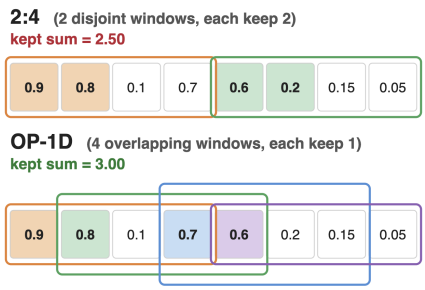
Why overlap helps. Overlap does more than let a selector reach a high-saliency weight across a block boundary; it also *couples* neighboring decisions. Under 2:4, each window is an island optimized in isolation, so the mask concatenates independent per-block top-2 choices. Sharing candidate columns links these decisions into a single coupled problem, letting us maximize *total* retained saliency. This coupling makes mask selection global within a tile rather than per-window, yet still exactly solvable (§3.5).

3.2 Overlapping-2D

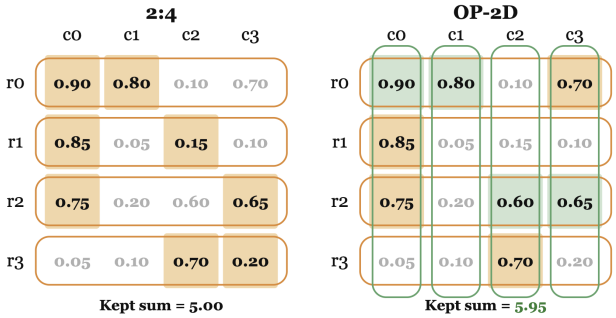
From one axis to two. OP-2D raises the *dimensionality of selector connectivity*: 2:4 has none (each block is an island), OP-1D couples each selector to two neighbors along a single axis, and OP-2D couples each selector to four neighbors across both axes.

Construction. A four-input selector could tap any four positions, but we restrict it to a contiguous $1{\times}4$, $4{\times}1$, or $2{\times}2$ window to keep wiring local. OP-2D uses the two straight layouts such that any two overlapping selectors share *exactly one* element. This minimal overlap is the source of OP-2D's efficiency: OP-1D must share two elements per interior pair to guarantee 50% sparsity. The smallest OP-2D tile supporting both orientations under this non-redundant overlap is $4{\times}4$, in which four row and four column selectors are stacked so every element feeds exactly one selector of each orientation (Figure 1b). A valid mask selects one weight per selector without collision.

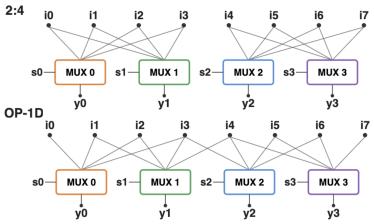
Scaling and transposability. Contiguity and the one-element overlap then determine the scaling of the pattern: a $4k \times 4k$ tile uses 8k² selectors and scales in increments of 4 along both axes. As shown for $8{\times}8$ in Figure 3, four $4{\times}4$ tiles are joined by *boundary bands* that begin and end with $2{\times}2$



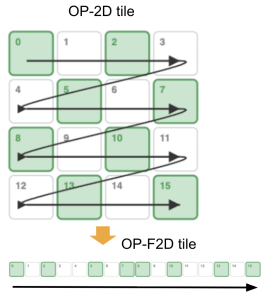
(a) 2:4 vs. OP-1D windows for 1×8.



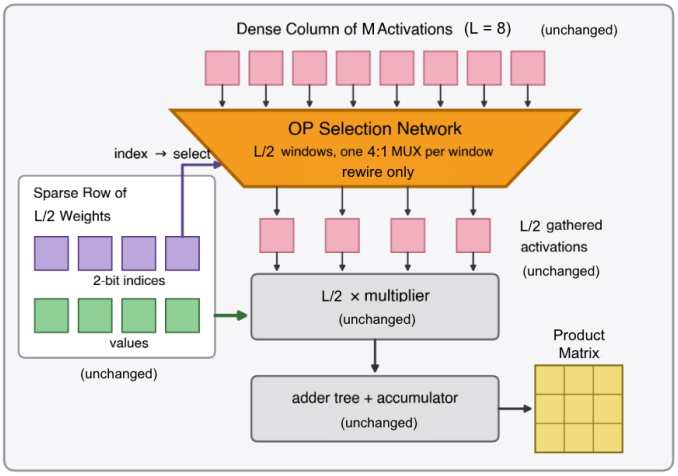
(b) Selection windows for 2:4 vs. OP-2D for 4×4.



(c) 2:4 vs. OP-1D selection wiring.



(d) OP-2D to OP-F2D mapping.



(e) Sparse matrix multiplication hardware. OP-1D and OP-F2D require only selector rewiring compared with 2:4.

Figure 1: OP vs. 2:4 sparsity. OP-1D overlaps selection windows by rewiring the selection logic (1a and 1c). OP-2D extends overlap to 2D and OP-F2D linearizes the 2D space to one tensor row (1b and 1d). OP-1D and OP-F2D only require selector wiring changes as compared to 2:4 (1e).

selectors offsetting the neighboring row and column selectors, preserving the one-element overlap invariant and crosspoint efficiency at every size. The construction is symmetric under transposition—interior row and column selectors exchange roles, and the bands are symmetric under row–column exchange—so if M is feasible then so is $M^\top$. Forward and backward passes thus share the same OP-2D-valid mask family, a structural prerequisite for dynamic sparse training.

3.3 Overlapping-Flattened-2D

OP-2D's 4×1 and 2×2 selectors draw weights from multiple rows, whereas 2:4 accumulates all products into one destination. To recover that behavior, we map each OP-2D tile's 2D positions into a single tensor row via a fixed row-major index (Figure 1d), yielding *OP-F2D*. The mapping is hardware-friendly: wide logic is commonly folded into a 2D layout to improve floorplanning and bound wire length, so treating a linear row as a 2D tile matches how such structures are physically realized. Selected weights retain OP-2D's connectivity within one row, and the inverse mapping recovers each entry's coordinate so all products accumulate into the same destination. OP-F2D achieves this with only selector rewiring, trading mask transposability for greater flexibility.

3.4 Feasible Mask Comparison

We quantify representational flexibility by counting distinct feasible masks exactly (Table 1). For N:M sparsity, the counts factor as $6^{L/4}$, $70^{L/8}$, and $12870^{L/16}$. Overlapping patterns couple decisions

Table 1: Feasible mask counts by sparsity structure; L is the tile size.

L	Unstr. 50%	8:16	4:8	2:4	**OP-1D**	**OP-F2D & 2D**	OP-F2D / 2:4
16	12,870	12,870	4,900	1,296	4,873	$\mathbf{8{,}442}$	6.51
64	1.83×10^{18}	2.74×10^{16}	5.76×10^{14}	2.82×10^{12}	1.38×10^{15}	$\mathbf{2.41\times10^{16}}$	8.5×10^{3}
144	1.48×10^{42}	9.69×10^{36}	1.63×10^{33}	1.03×10^{28}	1.67×10^{34}	$\mathbf{1.31\times10^{37}}$	1.3×10^{9}

across the tile and admit no product form. As proved in Appendix C, OP-1D counts satisfy

$$T_L = 9\,T_{L-4} + 4, \qquad L \geq 16,\ L \equiv 0 \pmod 4. \tag{1}$$

For OP-2D, a mask is feasible exactly when its kept positions can be matched to distinct selectors; these full-density sets are the *bases of the bicircular matroid* of the selector graph, i.e. spanning subgraphs whose every component is unicyclic (Guo & Jerrum, 2021). We count them with a frontier dynamic program in arbitrary-precision arithmetic, so counts up to $\sim10^{66}$ are exact (Appendix B).

Because coupling spans the tile rather than isolated blocks, OP's mask count grows strictly faster than N:M. For the tile sizes presented in Table 1, OP-1D overtakes 4:8 at $L = 64$ and OP-2D overtakes 8:16 at $L = 144$, giving the solver a substantially richer space to optimize over.

3.5 Exact Mask Selection

Overlapping makes mask generation non-decomposable: in 2:4 each four-weight group is optimized in isolation, but in OP a weight claimed by one selector may remove the best option for a neighbor. We therefore solve one coordinated problem per hardware tile. Candidate sets do not cross tile boundaries, so per-tile optimality yields a globally score-optimal mask.

Formulation. Let $I = \{0, \ldots, K-1\}$ index selectors and $J = \{0, \ldots, L-1\}$ index positions within a tile, so a mask keeping one position per selector has density K/L. Let $x_{ij} \in \{0, 1\}$ indicate that selector i keeps candidate $j \in C_i$, and let $s_j \geq 0$ be the saliency of position j. All patterns here use $K = L/2$ and $|C_i| = 4$, though neither is assumed below. We solve

$$\max_{\substack{x_{ij}\in\{0,1\}\\ i\in I,\ j\in C_i}} \sum_{i\in I}\sum_{j\in C_i} s_j x_{ij} \qquad \text{s.t.} \quad \sum_{j\in C_i} x_{ij} = 1\ \ \forall i \in I, \qquad \sum_{i:\, j\in C_i} x_{ij} \leq 1\ \ \forall j \in J. \tag{2}$$

A mask is thus feasible iff its positions can be assigned to distinct selectors. Feasibility couples positions through shared selectors, so the top-K of s is generally infeasible, and greedily repairing it is suboptimal. The formulation covers OP-1D, OP-2D, and OP-F2D; only $\{C_i\}$ differs.

General solver of record: primal–dual assignment. We solve equation 2 exactly for *every* OP pattern as a maximum-weight bipartite matching, using an exact sparse primal–dual method in the Hungarian/Kuhn–Munkres family (Kuhn, 1955; Munkres, 1957) with the shortest-augmenting-path refinement of Jonker and Volgenant (Jonker & Volgenant, 1987; Crouse, 2016). It maintains dual potentials and augments K times via Dijkstra's algorithm (Dijkstra, 1959) over the reduced-cost residual graph. Terminating with a feasible primal and dual of equal objective certifies optimality. The graph is extremely sparse ($|E| = \sum_i |C_i| = 4K$), so each tile costs $O(K(|E| + L)\log L)$ time and $O(L)$ working memory, with all tiles run in parallel as a CUDA kernel (Appendix E).

Nothing in the method depends on $K = L/2$ or $|C_i| = 4$, only on the incidence lists $\{C_i\}$. Varying the selector count changes the density K/L, so other sparsity levels need only rewiring (the rectangular $K \neq L$ assignment problem; Crouse, 2016), as do families with unequal fan-out or positions of arbitrary degree. This solver produces all model-quality results reported here, and its duality certificate validates every specialization below.

Mask selection is a matroid problem. OP patterns carry structure the assignment formulation does not expose, and exploiting it sharply reduces runtime at no loss of exactness. The feasible sets of equation 2 are exactly the independent sets of the transversal matroid $\mathcal{M}(G)$ induced by $\{C_i\}$, which underlies the counts of §3.4. Its rank is K, and full-density masks are its bases.

Algorithm 1 makes a single descending pass, admitting j iff the partial mask stays feasible. By the Rado–Edmonds theorem (Edmonds, 1971) it returns a maximum-weight basis, i.e. an optimum of equation 2, with ties leaving the objective unchanged (proof: Appendix G). Every basis has size K, so the pass reaches 50% density with no budget constraint, at cost $O(L \log L)$ plus at most L calls to EXTENDABLE. Correctness is therefore established once, for all patterns: the reduction

Algorithm 1: GREEDYOPMASK: exact saliency-ordered mask selection for one tile. The oracle EXTENDABLE is the only pattern-dependent component.

Require: saliency $s \in \mathbb{R}^{L}_{\geq 0}$; candidate lists $\{C_i\}_{i \in I}$; oracle state $\mathcal{S}$
Ensure: hardware-valid mask R, $|R| = K$, maximizing $\sum_{j \in R} s_j$

$\pi \leftarrow$ ARGSORTDESCENDING(s) ▷ registers only
$R \leftarrow \emptyset$; INITORACLE$(\mathcal{S})$
for $t \leftarrow 0$ **to** $L - 1$ **do**
  if $|R| = K$ **then break** ▷ optional early exit
  $j \leftarrow \pi[t]$
  if EXTENDABLE$(\mathcal{S}, R \cup \{j\})$ **then**
    $R \leftarrow R \cup \{j\}$; UPDATEORACLE$(\mathcal{S}, j)$
  end if
end for
return R

Table 2: Selector hardware cost of sparsity relative to 2:4 for 1×64 tile (↓); post synthesis, place, and route.

Prec.	Design	A_{cell}	A_{core}	Delay	Power
16-bit	8:16	3.12	4.24	3.41	2.53
	4:8	1.57	1.29	1.83	1.62
	OP-F2D	1.02	1.09	0.96	1.10
	OP-1D	1.00	1.00	0.94	1.01
8-bit	8:16	3.54	4.22	2.27	2.71
	4:8	2.36	1.85	1.67	1.81
	OP-F2D	1.04	1.00	1.04	1.16
	OP-1D	0.99	1.00	0.99	0.97
4-bit	8:16	3.84	2.48	2.59	2.60
	4:8	2.22	1.19	1.71	1.72
	OP-F2D	1.01	1.00	0.99	0.93
	OP-1D	1.00	1.00	0.99	0.99

holds for every OP connectivity, and a generic augmenting-path oracle implements EXTENDABLE in full generality. Only the cost is pattern-specific. The general solver is not redundant, since duality certifies optimality independently of the matroid argument and the assignment formulation extends to objectives outside it, such as selector-dependent scores s_{ij}.

Specialized oracle: union–find on the bicircular matroid. In OP-2D and OP-F2D, at *any* tile width, every position is wired to exactly two selectors, which reduces EXTENDABLE to a near-constant-time test. The pattern is then a graph with selectors as vertices and positions as edges, each joining the two selectors wired to it. Keeping a position assigns its edge to one endpoint, and a selector keeps at most one position, so every vertex receives at most one edge; a component with n vertices thus holds at most n edges, permitting at most one cycle. A position set is feasible precisely when no component of the graph it induces contains more than one cycle: the independent sets of the bicircular matroid of §3.4.

We track this with a union–find forest over selectors and one bit per component recording whether it already contains a cycle. A position is rejected only when its edge would close a cycle in an already-cyclic component or merge two cyclic components. Each test costs $O(\alpha(L))$ amortized, so the sort dominates the pass. At $L = 64$ it returns the same retained saliency as the primal–dual solver while running 13.1–15.5× faster, at 1.3–1.6× the latency of a non-overlapping 2:4 top-k baseline; Table 9 reports absolute cost. Appendix G gives two further oracles—a table lookup at $L = 16$ that runs 62.8–77.5× faster, and matching reuse for positions of degree above two—and verifies that all attain the primal–dual optimum, exhaustively at $L = 16$ and on randomly sampled tiles at $L = 64$.

3.6 OP INDEX AND HARDWARE EFFICIENCY

OP matches 2:4's throughput by construction: it keeps the 2-bit index per weight and $L/2$ weights per tile, leaving the multiply–accumulate count, data fetched per output, and schedule unchanged. Wider $N{:}M$ cannot: 4:8 requires a 3-bit index per kept weight and 8:16 a 4-bit index, growing metadata by 50% and 100%. OP instead preserves 2:4's selector count and four-input fan-in, changing only which positions each selector draws from (Figure 1e).

All selectors are synthesized, placed, and routed at 4-, 8- and 16-bit precisions through an identical flow, corner, and default switching-activity model; Table 2 reports ratios to 2:4 (1.00 throughout, omitted). Each design is floorplanned at its routability limit, the smallest core that routes cleanly: OP matches 2:4 or exceeds it by two search steps, while wider $N{:}M$ needs larger cores for its wider muxes and extra index bits. OP-1D is at parity or better on every metric; OP-F2D trades up to 4% cell area, 9% core area and 16% power for higher mask quality, and is faster than 2:4 at 16-bit. Both stay 2:4-class, whereas 4:8 and 8:16 cost up to 3.8× cell and 4.4× core area at up to 3.4× delay and 2.7× power. At 15 nm, the selector is 1.28% of multiply-accumulate unit (MAC) cell area and 1.78% of MAC power (Gondimalla et al., 2023), so OP's worst case costs 0.05% area and 0.29% power per MAC—a drop-in replacement for 2:4 at the selector level. Absolutes, PDK, and floorplan details are in Appendix H.

Table 3: Experimental setup. Detailed information is available in Appendix I.

Item	Setting
Models	Llama 7/13/30B · Llama 2 7/13B · Llama 3.1 8B · Qwen 2.5 7/32B · Gemma 2 9B · OPT 6.7B · Llama 3 8B (Table 5 only)
Evaluation	*Perplexity:* WikiText-2 (Merity et al., 2017) *Zero-shot:* ARC-C, ARC-E, BoolQ, HellaSwag, OBQA, PIQA, RTE, Winogrande (Gao et al., 2023)
Methods	Wanda, Wanda++, RIA · *Calibration:* C4 (Raffel et al., 2020), 128 samples
Tile size	OP-1D: 1×16, 1×64, 1×256; OP-F2D: 1×16 (4×4), 1×64 (8×8), 1×256 (16×16)
Hardware	NVIDIA DGX Spark nodes (GB10, 128 GB unified memory each) · L40S (only for runtime comparison with ARMOR)

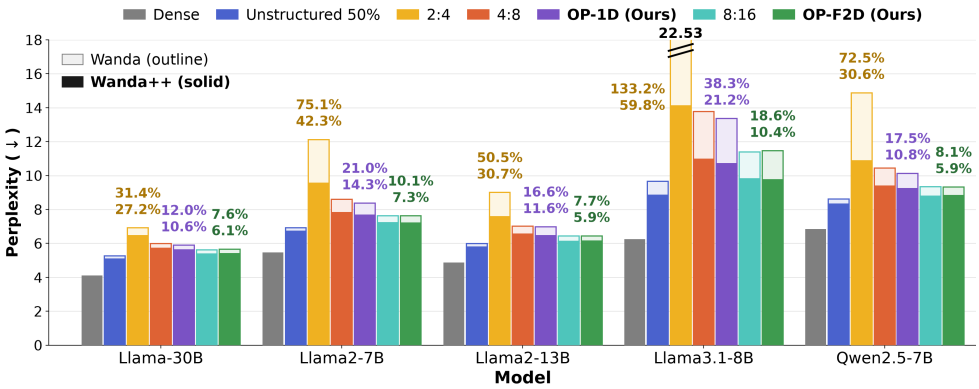


Figure 2: Perplexity at 50% sparsity across sparsity structures. Wanda++ is strictly lower than Wanda for models shown. The percentages shown are the relative percentage difference (Δ_{rel}) between each sparsity and unstructured 50% (top = Wanda, bottom = Wanda++). OP with 1×64 tile size.

4 Experiments

We evaluate two claims: that selector connectivity alone recovers most of the quality lost to 2:4 (§4.1), and that OP composes with orthogonal compression axes (§4.2); Table 3 summarizes the setup. To separate connectivity from saliency scoring, we pair OP with Wanda, RIA (Zhang et al., 2024), and Wanda++, in each case having the solver maximize the saliency scores under the overlapping-window constraints. OP adds no new scoring rule, so any quality change comes from the enlarged mask space. Baselines are dense, unstructured 50%, N:M baselines with $N{:}M \in \{2{:}4,\ 4{:}8,\ 8{:}16\}$, and SOTA 2:4 methods that expand the effective design space: MaskLLM and PermLLM, which learn masks or permutations, and ARMOR, which wraps a 2:4 core in block-diagonal factors at $\sim 5\%$ added parameters (7B/13B). We report OP at tile size 1×64, matching the reduction width of one FP8 structured-sparse tensor-core MMA instruction (`m16n8k64`; NVFP4 doubles this to `m16n8k128`) (NVIDIA Corporation, 2026). Across five calibration seeds on Llama 3.1 8B, standard deviations stay within 0.25 overall and 0.13 under Wanda++ (Table 16, Appendix J.1), well below the differences reported below.

4.1 Model Quality

Perplexity. Figure 2 plots perplexity at 50% sparsity for each structure, annotated with the relative difference to the unstructured oracle, $\Delta_{\text{rel}} = (\text{PPL}_{\text{pattern}} - \text{PPL}_{\text{unstr.}})/\text{PPL}_{\text{unstr.}}$ (↓). The rigidity penalty is large on every model: relative to unstructured 50%, 2:4 inflates perplexity by 28.5–133.2% (mean 58.4%) under Wanda and 24.2–59.8% (mean 34.7%) under Wanda++ (for all ten models in Table 4). Overlapping removes most of it at equivalent hardware and index cost: OP-F2D's mean Δ_{rel} is $+9.1\%$ (Wanda) and $+6.6\%$ (Wanda++), spanning 5.7–18.6% and 5.3–10.4%, with OP-1D consistently in between. Both largest residuals fall on Llama 3.1 8B, which is also where 2:4 degrades most: OP's absolute margin is widest there, recovering an above-average 86.0%/82.6% of the gap. From Wanda to Wanda++, Δ_{rel} falls on $10/10$ models for both OP variants (and under RIA, Appendix K): mask-space freedom and scoring quality are complementary, not substitutes.

Table 4 reports the absolute perplexities behind Figure 2, alongside each model's dense baseline. Two patterns emerge. First, 2:4 → OP-1D → OP-F2D is monotonic on all 20 model–scorer pairs; since OP adds no scoring rule, the mask space is the only difference, and every OP-F2D-vs-2:4 margin is at least 1.07 PPL. Second, the progression tracks an index cost OP does not pay: OP-1D beats 4:8 on $20/20$ pairs (0.04–0.41 PPL), while OP-F2D and 8:16 are indistinguishable ($|\Delta| \le 0.08$, mean

Table 4: Perplexity (↓) at sequence length 2048; sparsity columns are grouped by hardware and index cost. The last two columns report OP-F2D relative to unstructured 50%: Δ_{rel} and fraction of 2:4→unstructured gap recovered (Gap closed). Best 2:4-efficiency in **bold**; RIA data are in Table 19.

		Higher HW & index cost			**At 2:4 HW & index cost**			**OP-F2D vs. Unstr. 50%**	
Model (Dense)	**Method**	**Unstr. 50%**	**8:16**	**4:8**	**2:4**	**OP-1D**	**OP-F2D**	Δ_{rel} (↓)	**Gap closed** (↑)
Llama 7B	Wanda	7.26	7.84	8.56	11.55	8.52	**7.92**	+9.1%	84.6%
(5.68)	Wanda++	6.96	7.42	7.92	9.61	7.83	**7.44**	+6.9%	81.9%
Llama 13B	Wanda	6.15	6.75	7.41	9.62	7.22	**6.69**	+8.8%	84.4%
(5.09)	Wanda++	6.00	6.43	6.84	7.97	6.79	**6.37**	+6.2%	81.2%
Llama 30B	Wanda	5.26	5.61	5.98	6.91	5.89	**5.66**	+7.6%	75.8%
(4.10)	Wanda++	5.08	5.37	5.70	6.46	5.62	**5.39**	+6.1%	77.5%
Llama 2 7B	Wanda	6.92	7.62	8.60	12.12	8.37	**7.62**	+10.1%	86.5%
(5.47)	Wanda++	6.71	7.21	7.82	9.55	7.67	**7.20**	+7.3%	82.7%
Llama 2 13B	Wanda	5.98	6.44	7.01	9.00	6.97	**6.44**	+7.7%	84.8%
(4.88)	Wanda++	5.79	6.12	6.54	7.57	6.46	**6.13**	+5.9%	80.9%
Llama 3.1 8B	Wanda	9.66	11.39	13.77	22.53	13.36	**11.46**	+18.6%	86.0%
(6.24)	Wanda++	8.83	9.81	10.95	14.11	10.70	**9.75**	+10.4%	82.6%
Qwen 2.5 7B	Wanda	8.62	9.34	10.43	14.87	10.13	**9.32**	+8.1%	88.8%
(6.85)	Wanda++	8.32	8.78	9.37	10.87	9.22	**8.81**	+5.9%	80.8%
Qwen 2.5 32B	Wanda	6.29	6.64	7.06	8.08	6.95	**6.65**	+5.7%	79.9%
(5.02)	Wanda++	6.23	6.57	6.93	7.74	6.84	**6.57**	+5.5%	77.5%
Gemma 2 9B	Wanda	8.46	9.10	9.92	12.24	9.77	**9.11**	+7.7%	82.8%
(6.84)	Wanda++	8.47	9.08	9.74	11.32	9.60	**9.05**	+6.8%	79.6%
OPT 6.7B	Wanda	11.99	12.79	13.57	15.95	13.47	**12.84**	+7.1%	78.5%
(10.86)	Wanda++	11.86	12.48	13.11	15.14	13.01	**12.49**	+5.3%	80.8%

Table 5: Comparison of OP with Wanda++ against reported perplexities for SOTA 2:4 pruning methods. PermLLM and MaskLLM are 2:4, while ARMOR‡ adds parameters to 2:4. † Not reported.

	Seq. Len. 2048				**Seq. Len. 4096** (8192 for Llama 3 8B)			
Model	PermLLM	ARMOR‡	OP-1D	OP-F2D	MaskLLM	ARMOR‡	OP-1D	OP-F2D
Llama 2 7B	9.39	7.65	7.67	**7.20**	6.72	7.21	7.14	**6.70**
Llama 2 13B	8.20	6.80	6.46	**6.13**	5.85	6.37	6.06	**5.74**
Llama 3 8B	n/r†	11.27	10.88	**9.94**	n/r†	10.10	9.74	**8.86**

$|\Delta| = 0.03$). Averaged over ten models at a tile size of 1×64, OP-F2D recovers 83.2% (Wanda) and 80.6% (Wanda++) of the $2{:}4 \rightarrow$ unstructured gap, minimum 75.8%. Even at a tile size of $1{\times}16$, OP-F2D still closes 77.3% and 73.6% of that gap, minimum 70.9% (Table 21).

The ordering appears in the objective. Because mask selection for N:M and OP is exact, the saliency a pattern retains is a property of its wiring rather than of the solver. Averaged over all ten evaluated models under Wanda, patterns retain 75.7% (2:4), 77.9% (4:8), 79.2% (8:16), 78.2% (OP-1D) and 79.2% (OP-F2D) of total layer saliency, against 80.7% for unstructured 50%. OP-1D exceeds 4:8 and OP-F2D matches 8:16. Retained saliency reproduces the quality ordering of Table 4.

Comparison to SOTA 2:4 pruning methods. Table 5 positions OP against methods that expand the effective 2:4 design space, either by *training* masks or by *augmenting* the format. Against PermLLM, which learns permutations, OP-F2D lowers perplexity by 23–25%. Against MaskLLM, it matches masks trained for 1,280 and 2,304 A100 GPU hours (6.70 vs. 6.72 on Llama 2 7B; 5.74 vs. 5.85 on Llama 2 13B): enlarging the feasible mask space substitutes for learning within a restrictive one, at zero mask-training cost. Against ARMOR, which wraps a 2:4 core in block-diagonal factors and adds $\sim$5% parameters, OP-1D matches or edges out ARMOR on all comparison models, while OP-F2D improves upon it across all models and context lengths—most clearly on Llama 3 8B at 8k (8.86 vs. 10.10)—with no parameter overhead and $\sim 59\times$ less compute (16 A6000 GPU hours for ARMOR (Liu et al., 2026) vs. 0.27 L40S hour for OP-F2D w/ Wanda++ on Llama 2 7B).

Task accuracy. Table 6 confirms the same ordering on downstream tasks and shows the loss is structural rather than a consequence of 50% sparsity itself. Unstructured pruning stays within 2.5–4.6 percentage points (pp) of dense on most models, widening to 7.73 pp on the pruning-sensitive

Table 6: Average zero-shot performance across eight evaluation tasks (Table 3) for sparsity patterns.

Sparsity Pattern	Llama 7B		Llama 2 7B		Llama 3.1 8B		Qwen 2.5 7B		Gemma 2 9B	
	Wanda	Wanda++	Wanda	Wanda++	Wanda	Wanda++	Wanda	Wanda++	Wanda	Wanda++
Dense	62.38		62.23		67.09		67.42		68.74	
Unstr. 50%	57.95	58.74	58.47	57.95	59.36	59.76	63.55	64.70	64.83	64.15
8:16	57.68	57.48	57.07	56.90	56.64	57.33	63.09	63.84	63.32	62.78
4:8	55.62	55.98	55.42	55.48	54.14	55.48	61.91	62.76	62.52	61.22
2:4	51.75	53.09	51.41	52.58	49.71	52.59	57.79	60.40	57.97	57.37
OP-1D	55.69	56.44	56.36	56.22	54.72	56.01	61.97	62.90	62.01	61.44
OP-F2D	**58.13**	**57.52**	**57.43**	**57.83**	**56.67**	**57.52**	**62.53**	**63.78**	**62.75**	**62.00**

Table 7: Llama 3.1 8B perplexity (↓) under Weight-Only and Weight+Activation quantization from BF16 to FP8 and NVFP4; sparse columns use Wanda++. Shaded columns are at 2:4 HW efficiency.

		Sparsity Pattern							Rel. to Unstr.	
Regime	**Quant.**	**Dense**	**Unstr. 50%**	**8:16**	**4:8**	**2:4**	**OP-1D**	**OP-F2D**	**2:4**	**OP-F2D**
Baseline	BF16	6.24	8.83	9.81	10.95	14.11	10.70	**9.75**	+59.8%	+10.4%
W-Only	FP8	6.27	8.88	9.90	11.03	14.25	10.79	**9.80**	+60.5%	+10.4%
	NVFP4	6.63	9.43	10.50	11.71	15.37	11.59	**10.43**	+63.0%	+10.6%
W+Act	FP8	6.30	8.92	9.96	11.10	14.37	10.87	**9.86**	+61.1%	+10.5%
	NVFP4	6.99	10.02	11.22	12.60	16.77	12.46	**11.18**	+67.4%	+11.6%

Llama 3.1 8B (Wanda); rigid 2:4 then falls a further 9.65 pp below unstructured on that same model. At identical 2:4 hardware cost, moving $2{:}4 \rightarrow$ OP-1D $\rightarrow$ OP-F2D monotonically recovers this loss, with OP-F2D gaining 5.78 pp over 2:4 on average (Wanda) and closing 70–103% of the semi-structured-to-unstructured gap—on Llama 7B it closes the gap entirely. Gains are largest where the model is most fragile: OP-F2D with Wanda++ reaches 57.52% on Llama 3.1 8B, within 2.24 pp of unstructured. OP-F2D exceeds 8:16 for three of the five models and trails by at most 0.8 pp.

4.2 Compatibility with Quantization and Channel Permutation

Because OP changes only which candidates a selector sees, it should compose with compression axes that act on weight values or ordering. Pruning followed by weight-only and weight+activation quantization confirms that OP does not compound quantization error (Table 7; Appendix M). Measured as the perplexity penalty relative to unstructured 50%, 2:4 costs $+59.8\%$ to $+67.4\%$ across BF16, FP8, and NVFP4, whereas OP-F2D stays at 10.4–11.6%; the perplexity ordering $2{:}4 >$ OP-1D $>$ OP-F2D holds at every precision, so OP's quality gains survive aggressive quantization.

Channel permutation reorders which weights compete, while OP changes which selectors see each weight. Structure-unaware orderings can cause contention by concentrating high-importance columns in overlapping selector windows. Our OP-aware ordering reduces contention by spreading high-importance columns across selector-compatible slots, showing that channel permutation complements OP when allocation respects selector connectivity. Details are in Appendix N.

5 Conclusion

We traced the quality gap between 2:4 and unstructured pruning to mask *rigidity*: 2:4's disjoint windows limit which masks are reachable. Overlapping Pruning removes that limit by letting neighboring selectors share candidates. The resulting selection problem is still solved exactly: feasible masks are the independent sets of a transversal matroid, so a saliency-ordered greedy pass maximizes retained saliency with no relaxation or search. One-shot and without retraining, OP-F2D closes 75.8–88.8% of the unstructured-to-2:4 quality gap across ten models and three saliency criteria: OP-1D surpasses 4:8 and OP-F2D comes within 1.1% of 8:16—$N{:}M$ patterns that cost 1.2–$4.2\times$ 2:4's selector core area and 1.7–$3.4\times$ its delay—while OP keeps 2:4's selector count, 2-bit index, and datapath for 0.05% additional area per MAC. Flexible mask geometry need not require costly flexible hardware, which makes OP a design option for the next generation of sparse accelerators.

Limitations and future work. Our hardware claims rest on synthesis and place-and-route, not silicon. A physical implementation, lane-partitioned operand collector, compatibility with software methods for flexible N:M sparsity, and OP-2D dynamic sparse training remain future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist in proving mathematical claims. We also used generative AI tools to retrieve and discover potentially relevant related work and to aid and polish the writing of the manuscript. We did not use generative AI tools for other tasks requiring disclosure, and the remaining required-disclosure tasks were not applicable to this work. The authors reviewed all AI-assisted work: suggested literature was checked against the original sources, mathematical arguments were independently verified, and AI-assisted prose was revised and approved by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies model pruning and accelerator design using existing pretrained models and established text benchmarks. It does not involve human subjects, a user study, the collection of personal information, or the release of a new dataset. Overlapping Pruning may reduce the computational and energy cost of large-model inference; however, as a general-purpose compression technique, it may also lower the cost of deploying models that retain biases, privacy risks, unsafe behaviors, or other limitations of their dense counterparts. Pruning is not a substitute for application-specific safety evaluation. Deployments of pruned models should therefore apply the same safeguards and assess the same domain-specific risks as deployments of the corresponding dense models.

REPRODUCIBILITY STATEMENT

The OP patterns, assignment formulation, and exact mask-selection algorithm are specified in Sections 3.1–3.5. The experimental models, datasets, calibration protocol, evaluation tasks, and tile sizes are summarized in Table 3. The appendix provides solver pseudocode and complexity analysis (Appendix E), proofs and verification of exact greedy selection (Appendix G), the hardware flow, process, constraints, floorplan, and absolute measurements (Appendix H), robustness studies (Appendix J), and detailed perplexity and zero-shot results (Appendices K and L). The anonymous supplementary repository contains the full implementation referenced in the main text and appendix.

REFERENCES

Advanced Micro Devices, Inc. hipSPARSELt features for the AMD Instinct MI300 series, 2025. URL https://rocm.docs.amd.com/projects/hipSPARSELt/en/docs-7.0.1/conceptual/mi300-features.html. ROCm 7.0.1 documentation, hipSPARSELt 0.2.4. Accessed September 11, 2026.

Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367257. doi: 10.1145/3316781.3326334. URL https://doi.org/10.1145/3316781.3326334.

Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, Ronny Krashinsky, and Kyle Aubrey. Introducing NVFP4 for efficient and accurate low-precision inference. NVIDIA Technical Blog, June 2025. URL https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/. Published June 24, 2025. Accessed September 11, 2026.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vXxardq6db.

Jesse Cai, Daniel Haziza, and Supriya Rao. Accelerating neural network training with semi-structured (2:4) sparsity. PyTorch Blog, June 2024. URL https://pytorch.org/blog/accelerating-neural-network-training/. Published June 20, 2024. Accessed September 23, 2026.

Roberto L. Castro, Andrei Ivanov, Diego Andrade, Tal Ben-Nun, Basilio B. Fraguela, and Torsten Hoefler. Venom: A vectorized n:m format for unleashing the power of sparse tensor cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607087. URL https://doi.org/10.1145/3581784.3607087.

James Cherry and The OpenROAD Project. OpenSTA: Parallax static timing analyzer (openroad fork). GitHub Repository, 2020. URL https://github.com/the-openroad-project/opensta.

David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016. doi: 10.1109/TAES.2016.140952.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models, 2024. URL https://arxiv.org/abs/2311.04902.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL https://doi.org/10.1007/BF01386390.

Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971. URL https://doi.org/10.1007/BF01584082.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. MaskLLM: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7736–7758. Curran Associates, Inc., 2024. doi: 10.52202/079017-0248. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0e9a05f5ce62284c91e4a33498899124-Paper-Conference.pdf.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/frantar23a.html.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tcbBPnfwxS.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, December 2023. URL https://zenodo.org/records/10256836.

Ashish Gondimalla, Mithuna Thottethodi, and T. N. Vijaykumar. Eureka: Efficient tensor cores for one-sided unstructured sparsity in dnn inference. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, pp. 324–337, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294. doi: 10.1145/3613424.3614312. URL https://doi.org/10.1145/3613424.3614312.

Heng Guo and Mark Jerrum. Approximately counting bases of bicircular matroids. *Combinatorics, Probability and Computing*, 30(1):124–135, 2021. doi: 10.1017/S0963548320000292.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1510.00149.

Simla Harma, Ayan Chakraborty, Elizaveta Kostenok, Danila Mishin, Dongho Ha, Babak Falsafi, Martin Jaggi, Ming Liu, Yunho Oh, Suvinay Subramanian, and Amir Yazdanbakhsh. Effective interplay between sparsity and quantization: From theory to practice. In *International Conference on Learning Representations*, volume 2025, pp. 95007–95045, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ed032b08a8822c3635cdcd961012ce60-Paper-Conference.pdf.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

Sara Hooker. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021. doi: 10.1145/3467017. URL https://doi.org/10.1145/3467017.

R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, November 1987. ISSN 0010-485X. doi: 10.1007/BF02278710. URL https://doi.org/10.1007/BF02278710.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: 10.1002/nav.3800020109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Guanchen Li, Yixing Xu, Zeping Li, Ji Liu, Xuanwu Yin, Dong Li, and Emad Barsoum. Týr-the-pruner: Structural pruning llms via global sparsity distribution optimization. In *Advances in Neural Information Processing Systems*, volume 38, pp. 165624–165653. Curran Associates, Inc., 2025. doi: 10.52202/085713-4989. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/db2cbf43a349bc866111e791b58c7bf4-Paper-Conference.pdf.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf.

Lawrence Liu, Alexander Liu, Mengdi Wang, Tuo Zhao, and Lin F. Yang. ARMOR: High-performance semi-structured pruning via adaptive matrix factorization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=8NE554wv0m.

Cong Ma, Du Wu, Zhelang Deng, Jiang Chen, Xiaowen Huang, Jintao Meng, Wenxi Zhu, Bingqiang Wang, Amelie Chi Zhou, Peng Chen, Minwen Deng, Yanjie Wei, Shengzhong Feng, and Yi Pan. NM-SpMM: Accelerating Matrix Multiplication Using N:M Sparsity with GPGPU. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 926–937, Los Alamitos, CA, USA, June 2025. IEEE Computer Society. doi: 10.1109/IPDPS64566.2025.00087. URL https://doi.ieeecomputersociety.org/10.1109/IPDPS64566.2025.00087.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21702–21720. Curran Associates, Inc., 2023. doi: 10.52202/075280-0950. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/44956951349095f74492a5471128a7e0-Paper-Conference.pdf.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. Fp8 formats for deep learning, 2022. URL https://arxiv.org/abs/2209.05433.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021. URL https://arxiv.org/abs/2104.08378.

James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. doi: 10.1137/0105003. URL https://doi.org/10.1137/0105003.

NVIDIA Corporation. Parallel Thread Execution ISA, version 9.4. https://docs.nvidia.com/cuda/parallel-thread-execution/, 2026. Accessed: 2026-09-15.

James G. Oxley. *Matroid Theory*. Oxford University Press, Oxford, 2nd edition, 2011.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Red Hat AI and vLLM Project. LLM Compressor, August 2024. URL https://github.com/vllm-project/llm-compressor. Software. Accessed September 11, 2026.

Mohamed Shalan and Tim Edwards. Building openlane: a 130nm openroad-based tapeout-proven flow. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415735. URL https://doi.org/10.1145/3400302.3415735.

SkyWater Technology Foundry and Google LLC. SkyWater SKY130 Process Design Kit (PDK). https://github.com/google/skywater-pdk, 2020. Accessed: 2026-09-21.

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/14c856c7a41297804de4c4890e846b25-Paper-Conference.pdf.

Clifford Wolf and Johann Glaser. Yosys – a free verilog synthesis suite. In *Austrochip Workshop on Microelectronics 2013*, pp. 47–52, October 2013. URL https://yosyshq.net/yosys/files/yosys-austrochip2013.pdf.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In *International Conference on Learning Representations*, volume 2024, pp. 5385–5409, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/160adf2dc118a920e7858484b92a37d8-Paper-Conference.pdf.

Yifan Yang, Kai Zhen, Bhavana Ganesh, Aram Galstyan, Goeric Huybrechts, Markus Müller, Jonas M. Kübler, Rupak Vignesh Swaminathan, Athanasios Mouchtaris, Sravan Babu Bodapati, Nathan Susanj, Zheng Zhang, Jack FitzGerald, and Abhishek Kumar. Wanda++: Pruning large language models via regional gradients. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4321–4333, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.224. URL https://aclanthology.org/2025.findings-acl.224/.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. Plug-and-play: An efficient post-training pruning method for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Tr0lPx9woF.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=K9bw7vqp_s.

Lancheng Zou, Shuo Yin, Zehua Pei, Tsung-Yi Ho, Farzan Farnia, and Bei Yu. Permllm: Learnable channel permutation for n:m sparse large language models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 20183–20205. Curran Associates, Inc., 2025. doi: 10.52202/085713-0682. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1d3fe3adb016edebc4fa615c25d22cb0-Paper-Conference.pdf.

A OP-F2D AND OP-2D SCALING

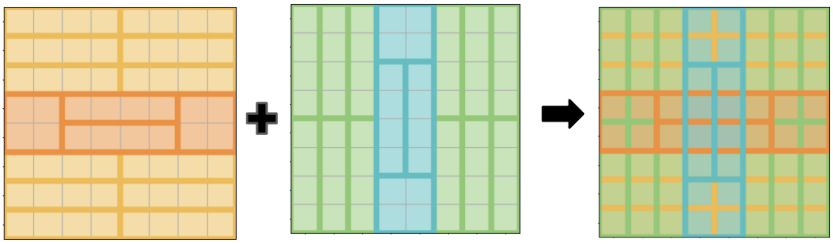


Figure 3: OP-2D scaling to 8×8. OP-F2D maps a single row's positions to OP-2D using row-major order.

B FEASIBLE MASK COUNT COMPARISON

As in the main text, L denotes the total number of weight positions in a tile, or equivalently its width after flattening. In N:M sparsity, N is the number of retained positions per disjoint block of width M; neither symbol denotes the tile size. We count distinct binary masks for fixed wiring, regardless of how many selector assignments realize each mask.

Table 8: Full comparison of distinct feasible mask counts on L positions. OP-1D uses a $1\times L$ tile. OP-2D entries use square tiles, and both OP-2D dimensions must be divisible by 4. OP-F2D uses exactly these shapes and has identical counts under flattening. Counts and ratios are rounded for display; ratios use the unrounded counts. N/A means not applicable, not zero.

L	Unstr. 50%	2:4	4:8	**OP-1D**	8:16	**OP-2D** ***(OP-F2D)***	Ratio to 2:4 OP-1D	Ratio to 2:4 OP-2D
8	70	36	70	60	N/A	N/A	1.667	N/A
16	12,870	1,296	4,900	4,873	12,870	8,442	3.760	6.514
64	1.83×10^{18}	2.82×10^{12}	5.76×10^{14}	$\mathbf{1.38\times10^{15}}$	2.74×10^{16}	$\mathbf{2.41\times10^{16}}$	**487.90**	$\mathbf{8.53\times10^{3}}$
144	1.48×10^{42}	1.03×10^{28}	1.63×10^{33}	1.67×10^{34}	9.69×10^{36}	1.31×10^{37}	1.62×10^{6}	1.27×10^{9}
256	5.77×10^{75}	6.33×10^{49}	1.10×10^{59}	8.76×10^{60}	5.67×10^{65}	1.48×10^{66}	1.38×10^{11}	2.33×10^{16}

B.1 COUNTING CONVENTION AND REFERENCE PATTERNS

For even L, let $J=\{0,\dots,L-1\}$ be the positions in one complete tile and $K=L/2$. Selector i has a four-position candidate set $C_i\subseteq J$. A retained support $T\subseteq J$ is feasible for OP if

$$|T|=K, \qquad \exists \text{ a bijection } \pi:\{0,\dots,K-1\}\to T \text{ with } \pi(i)\in C_i \text{ for every } i. \tag{3}$$

Each selector retains one admissible position, and no position is retained twice. We count supports, not assignments: different bijections onto the same T contribute only once. All counts refer to a complete tile with the stated candidate windows.

For the reference patterns, view the tile as one flattened row. For integers $M\geq1$ and $0\leq N\leq M$, independent N:M blocks give

$$C_{N:M}(L)=\binom{M}{N}^{L/M} \quad (M\mid L), \qquad C_{\text{unstr}}(L)=\binom{L}{L/2} \quad (2\mid L). \tag{4}$$

The unstructured reference imposes only a total retention budget, not separate budgets on the rows of an OP-2D tile. In particular, $C_{2:4}(L)=6^{L/4}$, $C_{4:8}(L)=70^{L/8}$, and $C_{8:16}(L)=12870^{L/16}$. Every ratio in Tables 1 and 8 divides the exact OP count by $6^{L/4}$ before rounding.

C OP-1D: EXACT COUNT AND RECURRENCE

Candidate windows. For $4 \mid L$ and $L \geq 8$, index the selectors in order of their left endpoints. The boundary-anchored candidate windows are

$$C_0 = \{0,1,2,3\}, \qquad C_{K-1} = \{L-4, L-3, L-2, L-1\},$$
$$C_i = \{2i-1, 2i, 2i+1, 2i+2\}, \qquad 1 \leq i \leq K-2. \tag{5}$$

The $L = 8$ example in Table 8 follows this same construction.

Counting each support once. Both endpoints of C_i are nondecreasing in i. If an assignment has an inversion $\pi(i) > \pi(j)$ with $i < j$, then both positions lie in both windows, so swapping them preserves feasibility. Removing inversions gives the unique increasing assignment of that support. Consequently,

$$C_{1\mathrm{D}}(L) = \#\{x_0 < \cdots < x_{K-1} : x_i \in C_i\}. \tag{6}$$

A direct integer evaluation uses $f_0(j) = \mathbf{1}\{j \in C_0\}$ and

$$f_i(j) = \mathbf{1}\{j \in C_i\} \sum_{h<j} f_{i-1}(h), \qquad C_{1\mathrm{D}}(L) = \sum_j f_{K-1}(j).$$

Closed form. Write $x_i = \ell_i + p_i$, where $\ell_i = \min C_i$ and $p_i \in \{0,1,2,3\}$. The first and last window shifts are 1, so they require $p_{i+1} \geq p_i$; all other shifts are 2 and require $p_{i+1} \geq p_i - 1$. Thus

$$C_{1\mathrm{D}}(L) = \boldsymbol{a}^\top Q^{K-3} \boldsymbol{b}, \qquad Q = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}. \tag{7}$$

The vectors count the first and last boundary transitions. Since Q has characteristic polynomial $\lambda^2(\lambda-1)(\lambda-3)$, the sequence $g_m = \boldsymbol{a}^\top Q^m \boldsymbol{b}$ satisfies $g_{m+2} = 4g_{m+1} - 3g_m$ for $m \geq 2$. Direct multiplication gives $g_2 = 180$ and $g_3 = 541$, hence $g_m = (361\,3^{m-2} - 1)/2$ for $m \geq 2$. Therefore, for $L \geq 12$ with $4 \mid L$,

$$\boxed{C_{1\mathrm{D}}(L) = \frac{361\,3^{L/2-5} - 1}{2}}, \qquad C_{1\mathrm{D}}(12) = 541,$$
$$C_{1\mathrm{D}}(L) = 9C_{1\mathrm{D}}(L-4) + 4 \quad (L \geq 16). \tag{8}$$

All lengths here are divisible by 4. At $L = 8$, Equation (7) gives $\boldsymbol{a}^\top Q \boldsymbol{b} = 60$; the closed form does not apply at that length.

D OP-2D AND OP-F2D: FEASIBILITY AND EXACT COUNT

Candidate construction. Let $L_r = 4k_r$, $L_c = 4k_c$, $k_r, k_c \geq 1$, and $L = L_r L_c$. To specify the OP-2D boundary-band construction, define

$$\mathcal{R} = \bigcup_{j=1}^{k_r - 1} \{4j-1, 4j\}, \qquad H_{r,c} = \{(r, c+t) : 0 \leq t < 4\},$$
$$D_{r,c} = \{(r+a, c+b) : a, b \in \{0,1\}\}.$$

With integer indices, its horizontal selector family is

$$\begin{aligned} \mathcal{H}_{L_r,L_c} = &\{H_{r,4q} : 0 \leq r < L_r,\ r \notin \mathcal{R},\ 0 \leq q < k_c\} \\ &\cup \{H_{r,2+4q} : r \in \mathcal{R},\ 0 \leq q < k_c - 1\} \\ &\cup \{D_{4j-1,c} : 1 \leq j < k_r,\ c \in \{0, L_c - 2\}\}. \end{aligned}$$

The vertical family is $\mathcal{H}^\top_{L_c,L_r}$. Each family partitions the tile: ordinary rows use disjoint 1×4 windows, while each boundary-row pair uses two end 2×2 windows and shifted 1×4 windows between them. Thus there are $L/4$ selectors per orientation, and every position belongs to exactly one horizontal and one vertical selector.

Feasibility as a graph property. Form the selector graph $G = (V, E)$: its $K = L/2$ vertices are the selectors, and each weight position is an edge joining its two selectors. For a retained support, let $A \subseteq E$ be its corresponding edges. We claim that the support is feasible if and only if every connected component of the spanning graph (V, A) is *unicyclic*, meaning it contains exactly one cycle.

For necessity, a feasible assignment gives each vertex one distinct incident edge. Within each component, this is a bijection between vertices and retained edges, so the component has equally many of each and is unicyclic. Conversely, in a unicyclic component, orient its cycle consistently and all attached tree edges toward the cycle. Every vertex then has exactly one outgoing edge; assigning that edge to its vertex gives a feasible assignment. This proves the claim and yields

$$C_{\mathrm{2D}}(L_r, L_c) = \#\{A \subseteq E : \text{every component of } (V, A) \text{ is unicyclic}\}. \tag{9}$$

The condition already implies $|A| = |V| = L/2$ and excludes isolated vertices. These are the bases of the selector graph's bicircular matroid (Guo & Jerrum, 2021). We count each edge subset once, not its possible cycle orientations or assignments.

Exact evaluation. Process the L edges in a fixed order, deciding whether to discard or retain each one. The frontier contains vertices incident to both processed and unprocessed edges. A state σ records the partition of frontier vertices into connected components and whether each component already contains a cycle. Introduce an endpoint as an isolated tree when it first appears. Reject a retained edge if it closes a cycle within an already-cyclic component or joins two already-cyclic components. Also reject a transition if a tree component loses its last frontier vertex. Otherwise forget vertices with no unprocessed incident edges and canonically relabel the remaining components.

Let $D_t(\sigma)$ count the partial retained edge sets reaching state σ after t edges. Let $\Phi_t(\sigma, b)$ be the transition for the next edge and decision $b \in \{0, 1\}$, with invalid transitions omitted. Initialize $D_0(\varnothing) = 1$ and all other counts to zero, then use

$$D_{t+1}(\sigma') = \sum_{\sigma} \sum_{\substack{b \in \{0,1\} \\ \Phi_t(\sigma, b) = \sigma'}} D_t(\sigma), \qquad C_{\mathrm{2D}}(L_r, L_c) = D_L(\varnothing). \tag{10}$$

Every edge subset follows one decision sequence. The state preserves exactly the connectivity and cycle information needed by future decisions, and a component is accepted when it closes if and only if it is unicyclic. Induction on the processed edges therefore proves that the recurrence evaluates Equation (9) without omission or duplication. For efficiency, all already-cyclic components may share one label: joining any two cyclic vertices is always invalid, whereas attaching a tree to either is valid.

Flattening and verification. For each OP-2D shape $L_r \times L_c$ in Table 8, OP-F2D applies the bijection $(r, c) \mapsto rL_c + c$. This relabeling preserves every candidate set and feasible support, so $C_{\mathrm{F2D}}(L) = C_{\mathrm{2D}}(L_r, L_c)$ for the corresponding shape, where $L = L_r L_c$. The count depends on the specified shape and wiring, not just on L. Transposition also bijectively exchanges the two selector families, proving $C_{\mathrm{2D}}(L_r, L_c) = C_{\mathrm{2D}}(L_c, L_r)$.

OP-1D's closed form agrees with the sorted-support dynamic program for every multiple of 4 from 12 through 256. Exhaustive matching over all $\binom{16}{8} = 12{,}870$ supports independently gives 4,873 OP-1D masks and 8,442 OP-2D masks; for OP-2D, matching and the unicyclic criterion agree on every support. Larger counts are evaluated by the frontier recurrence with exact integer arithmetic, not extrapolation. Reversing the traversal order and retaining distinct labels for cyclic components give identical counts at every applicable table size.

E ASSIGNMENT SOLVER PSEUDOCODE

Algorithm 2 Sparse Primal–Dual Assignment Solver for Overlapping Pruning

Require: Scores $s \in \mathbb{R}^{L}_{\geq 0}$, candidate lists $C_i \subseteq \{0, \ldots, L-1\}$ for selectors $i = 0, \ldots, K-1$, with $|C_i| = 4$ and $L = 2K$
Ensure: Assignment π, where π_i is the selected position for selector i

Initialize selector-to-position map $\pi_i \leftarrow \perp$ for all i
Initialize position-to-selector map $\rho_j \leftarrow \perp$ for all j
Initialize dual potentials $u_i \leftarrow 0$ and $v_j \leftarrow 0$
for $a = 0, \ldots, K-1$ **do**
    $(t, \mathcal{R}, \mathcal{T}, d, p) \leftarrow$ SPARSEAUGMENT$(a, s, C, \pi, \rho, u, v)$
    if $t = \perp$ **then**
        continue
    end if
    $\Delta \leftarrow d_t$
    $u_a \leftarrow u_a + \Delta$
    for all $r \in \mathcal{R} \setminus \{a\}$ **do**
        $u_r \leftarrow u_r + \Delta - d_{\pi_r}$
    end for
    for all $j \in \mathcal{T}$ **do**
        $v_j \leftarrow v_j - (\Delta - d_j)$
    end for
    $j \leftarrow t$
    repeat
        $r \leftarrow p_j$
        $j_{\text{old}} \leftarrow \pi_r$
        $\pi_r \leftarrow j$
        $\rho_j \leftarrow r$
        $j \leftarrow j_{\text{old}}$
    until $r = a$
end for
return π

Algorithm 3 Sparse Shortest Augmenting Path Subroutine

function SPARSEAUGMENT(a, s, C, π, ρ, u, v)
  $\mathcal{R} \leftarrow \{a\}, \mathcal{T} \leftarrow \emptyset$
  $d_j \leftarrow \infty$ and $p_j \leftarrow \perp$ for all positions j
  Initialize an empty min-priority queue Q
  $r \leftarrow a, \delta \leftarrow 0$
  while true do
    for all $j \in C_r$ **do**
      if $j \in \mathcal{T}$ **then**
        continue
      end if
      $\alpha \leftarrow \delta - s_j - u_r - v_j$
      if $\alpha < d_j$ **then**
        $d_j \leftarrow \alpha$
        $p_j \leftarrow r$
        Insert or decrease key of j in Q with priority d_j
      end if
    end for
    if Q is empty **then**
      return $(\perp, \mathcal{R}, \mathcal{T}, d, p)$
    end if
    $j \leftarrow$ EXTRACTMIN(Q)
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$
    $\delta \leftarrow d_j$
    if $\rho_j = \perp$ **then**
      return $(j, \mathcal{R}, \mathcal{T}, d, p)$
    end if
    $r \leftarrow \rho_j$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
  end while
end function

The full solver implementation is available in the GitHub repository: [anonymous].

F SOLVER COMPLEXITY

Time complexity. Let E be the edge set, with $|E| = \sum_i |C_i| = 4K$. The constant selector degree is the key structural property exploited by the solver: each selector contributes only four candidate edges, so the sparse formulation avoids the cost of materializing or traversing a dense $K \times L$ matrix. With a binary heap, one shortest-path augmentation costs $O((|E| + L) \log L)$ in the worst case. Since at most K augmentations are required, the worst-case runtime per tile is $O(K(|E| + L) \log L)$. Under the fixed 50% structure $L = 2K$ and $|E| = 4K$, this simplifies to $O(K^2 \log K)$.

Space complexity. The implementation is specialized to the sparse overlapping-selector structure. Candidate connectivity is stored as a tensor $\mathrm{col_idx} \in \mathbb{Z}^{K \times 4}$, so the solver never materializes a dense $K \times L$ cost matrix. For each instance, it stores only linear-size work arrays for the dual variables, predecessor map, tentative distances, heap state, and visitation stamps. Iteration-stamped buffers eliminate the need to clear all temporary arrays between augmentations. This yields $O(|E|+K+L) = O(K + L)$ working memory per instance, which reduces to $O(L)$ under $L = 2K$. For a batch of B independent tiles, the total working memory is therefore $O(B(K + L)) = O(BL)$, rather than the $O(BKL)$ storage required merely to materialize dense costs.

G Fast Exact Mask Selection: Why Greedy Is Optimal, and How to Make It Cheap

The primal–dual assignment solver of Appendix E is exact for every OP pattern, but pays for that generality with data-dependent iteration counts, dual bookkeeping, and global memory traffic. That is acceptable for one-shot pruning, where a mask is computed once per weight matrix. It is too slow for regimes that regenerate masks online—dynamic sparse training (masks refresh every few steps) and activation sparsity (a mask per token at inference)—where mask generation sits on the critical path.

This appendix substantiates the reduction used in §3.5: sort positions by saliency, scan once, and keep each position if the partial mask is still realizable by the selectors. Three claims, in plain terms:

1. **Greedy is optimal, not heuristic** (§G.2). Greedy under constraints is usually suboptimal, so this needs proof. The feasible masks turn out to have the same structural property that makes Kruskal's algorithm exact for minimum spanning trees, and the same theorem applies.
2. **For the deployed pattern, the feasibility test is union–find** (§G.3). For OP-2D and OP-F2D the test becomes "does adding this position create a forbidden cycle?"—Kruskal's cycle check with one extra bit per component.
3. **Other patterns need a different test, not a different algorithm** (§G.4, §G.5). Only the feasibility test changes; the sort-and-scan skeleton of Algorithm 1 is identical, and a generic augmenting-path test covers every connectivity.

§G.6 gives the verification protocol and §G.7 the measured cost.

G.1 Setup

A tile has L positions $J = \{0, \dots, L-1\}$ and K selectors $I = \{0, \dots, K-1\}$; all patterns here use $K = L/2$ (50% density), though nothing below requires it. Each position j has a saliency $s_j \geq 0$ (Wanda score, $|W_j|$, or any per-weight score). The OP pattern fixes a bipartite *connectivity graph* $G = (I \cup J, E)$, where $(i, j) \in E$ means selector i is *wired to* position j and may keep it. Let $d = \max_j \deg_G(j)$ be the largest number of selectors wired to any single position.

At inference each selector keeps exactly one of its wired positions, and no position may be kept twice. A set $T \subseteq J$ is therefore *feasible* iff each position in T can be assigned its own distinct selector, i.e. iff G has a matching saturating T. Write $\mathcal{I} = \{T \subseteq J : T \text{ feasible}\}$, so mask selection is

$$\max_{T \in \mathcal{I},\, |T| = K} \sum_{j \in T} s_j,$$

the set form of equation 2. The difficulty is that feasibility is not per-position: two individually keepable positions may be jointly unkeepable if they compete for the same selectors. This is why the top-K of s is generally not realizable.

G.2 Why a Single Greedy Pass Is Optimal

Greedy algorithms are normally only heuristics for constrained selection, so the natural objection is that sort-and-scan should lose accuracy relative to the assignment solver. It does not, for a specific structural reason.

A family of "allowed" sets is a *matroid* if (i) subsets of allowed sets are allowed, and (ii) whenever A and B are allowed and $|A| < |B|$, some element of B can be added to A keeping it allowed. Property (ii) is what prevents greedy from painting itself into a corner: an early choice can never make a later repair impossible. The canonical example is forests in a graph, which is why Kruskal's algorithm—sort edges, add each unless it closes a cycle—finds the optimal spanning tree. Our feasible masks are a second example.

Proposition 1. *$\mathcal{M}(G) = (J, \mathcal{I})$ is a matroid, the* transversal matroid *of G. If moreover G has a matching saturating all K selectors, then $\mathcal{M}(G)$ has rank K, and its bases—the maximal feasible sets—are exactly the feasible masks of density K/L.*

Proof. That the matchable subsets of one side of a bipartite graph are the independent sets of a matroid is the transversal matroid theorem of Edmonds and Fulkerson (Oxley, 2011). For the rank: a matching gives each element of T a distinct selector, so $|T| \le |I| = K$ for every $T \in \mathcal{I}$; and a matching saturating I exhibits a feasible set of size K. Hence the rank is K, and the bases are the feasible T with $|T| = K$. □

The saturating-matching hypothesis is a property of the *wiring*, independent of the data, and it holds for every OP pattern here: OP refines a non-overlapping $N{:}M$ baseline in which each selector owns a private "home" position, and those home positions are distinct, so assigning each selector its home position saturates I. We check this once per pattern; it is also implied whenever the primal–dual solver returns a full-density mask.

Corollary 1. *For any $s \ge 0$ and any tie-breaking rule, Algorithm 1 returns a feasible mask of density exactly K/L maximizing $\sum_{j \in T} s_j$, i.e. an optimum of equation 2.*

Proof. For any matroid, processing elements in nonincreasing weight order and keeping each one that preserves independence yields a maximum-weight basis (Rado–Edmonds; Edmonds, 1971). Apply this to $\mathcal{M}(G)$, whose bases are the density-K/L feasible masks by Proposition 1. Two consequences are worth stating explicitly. First, the pass ends with exactly K kept positions: greedy returns a basis, and all bases have size K, so the $|R| = K$ exit in Algorithm 1 is an optimization that skips the oracle once the basis is complete, not a constraint that produces the density. Second, the theorem does not depend on how ties in s are resolved, so tie-breaking can change *which* optimal mask is returned but never the objective. We break ties by smaller index for determinism. □

The pass costs $O(L \log L)$ for the sort plus at most L calls to a *feasibility oracle* (EXTENDABLE): "is $T \cup \{c\}$ still feasible?" Correctness of the entire solver therefore reduces to correctness of that one predicate, and speed is determined entirely by its cost. Nothing above assumed anything about G, so the reduction holds for every OP pattern; §G.5 gives an oracle that is likewise fully general. The specializations in §G.3 and §G.4 buy speed, not coverage.

Kernel realization. We instantiate Algorithm 1 with one thread per tile. The sort is a fixed bitonic network over registers—data-oblivious, so every lane in a warp executes the same comparison sequence regardless of s—and the scan is fully unrolled at the fixed $L \in \{16, 64\}$. Ties are broken by index inside the sort. All three oracles below plug into this skeleton unchanged.

G.3 DEPLOYED ORACLE: A UNION–FIND CYCLE TEST ($d = 2$)

In OP-2D and OP-F2D every position is wired to exactly one row selector and one column selector, so $d = 2$, at any tile width. This makes the feasibility test almost free.

Intuition. Turn the picture inside out: let each *selector* be a vertex and each *position* be an edge joining its two selectors. Keeping a position means assigning it to one of its two endpoints, and each selector can take at most one position, so a set of positions is feasible iff its edges can be oriented with no vertex receiving two arrows. A component with n vertices can absorb at most n edges, and a component with n vertices and n edges has exactly one cycle. The rule is therefore: *each connected component may contain at most one cycle*. This is Kruskal's "no cycles" condition relaxed by exactly one cycle per component.

Proposition 2. *If $d = 2$, then T is feasible iff every connected component of the subgraph formed by the edges of T contains at most one cycle. Equivalently, $\mathcal{M}(G)$ is the* bicircular matroid *of that graph (Oxley, 2011).*

Proof. Feasibility means orienting each edge of T toward one endpoint with in-degree ≤ 1 everywhere. Necessity: summing in-degrees, a component with n vertices admits at most n edges. Sufficiency: a connected component with $m \le n$ edges is a tree or a tree plus one edge; orient a tree away from a root, so every vertex receives at most its parent edge, and in the second case orient the unique cycle consistently around itself and the remaining edges away from it. A connected graph satisfies $m \le n$ iff it has at most one cycle. □

Implementation (Algorithm 5). Maintain a union–find forest over the K selector-vertices plus one bit *cyclic*$[\cdot]$ per component recording whether it already holds its single permitted cycle. For a candidate position with endpoints (u, v) and roots r_u, r_v:

(i) $r_u = r_v$ (endpoints already connected): the edge closes a cycle, so accept iff the component is not yet cyclic, then set the bit.

(ii) $r_u \neq r_v$: merging gives $n_u + n_v$ vertices and $m_u + m_v + 1$ edges, which stays within budget iff at most one side was cyclic. Accept unless both are, then union and OR the bits.

With path halving each call is $O(\alpha(L))$ amortized (α the inverse Ackermann function, ≤ 4 for any realistic L), so the solve is $O(L \log L)$, dominated by the sort. State is K integers and K bits—no adjacency scan, no queue, no shared memory. The kernel (`hv_bicircular_mask`) is identical to the $L = 16$ kernel except for this oracle. This is the configuration used for the deployed OP-F2D 1×64 results.

G.4 Precomputed Oracle: Bit-Packed Table ($L = 16$)

At $L = 16$ a set of kept positions fits in a 16-bit word, so feasibility of *every* possible set can be precomputed once per pattern: run a reference matching test offline on all $2^{16} = 65{,}536$ subsets and store one bit each. Packed into 1024 64-bit words this is 8 KiB instead of the 64 KiB of a byte-per-entry array—small enough to stay in L1/constant cache across all tiles, rows, and layers, which is what matters online where the table is reused for every token and step.

The oracle state is the current bitmask `selected`. Extendable forms $b = \texttt{selected} \,|\, (1 \ll c)$ and reads one bit, $(\texttt{table}[\, b \gg 6\,] \gg (b \,\&\, 63)) \,\&\, 1$, and Commit sets $\texttt{selected} \,|= (1 \ll c)$. Each test is $O(1)$. This is a memoization of the same predicate implemented by §G.3 and §G.5, not a different method; we report it to bound how cheap the oracle can get, and our quality results do not use it.

G.5 General Oracle: Incremental Augmenting Path (any d)

The cycle characterization needs $d = 2$, and at $L = 64$ a table cannot be materialized. Neither restriction applies to the oracle below, which implements Extendable for *any* connectivity: we keep an explicit matching alongside the greedy pass and test candidates with one augmenting-path search, the standard bipartite-matching primitive. We use it for OP-1D on a 1×64 tile, where boundary windows overlap more than pairwise and give a few positions three incident selectors.

Proposition 3. *Let T be feasible and let μ be a matching saturating T. Then $T \cup \{c\}$ is feasible iff there is a μ-augmenting path from c to some selector left free by μ.*

Proof. Restrict to the graph induced by $T \cup \{c\}$. There μ has size $|T|$ and leaves only c unsaturated on the position side, so a matching saturating $T \cup \{c\}$ exists iff μ is not maximum, iff a μ-augmenting path exists (Berge's theorem), and any such path must start at c. □

Each position stores its wired selectors in an `owners` list of length $\leq d$. The kernel (`overlapping_rank_mask`) runs a bounded breadth-first search from the candidate, alternating position $\rightarrow$ wired selector $\rightarrow$ currently matched position; on success it flips the matching along the path and keeps the position, otherwise it rejects and leaves the state untouched (Algorithm 4). A search visits $O(L)$ positions and $O(L)$ selectors, so a tile costs $O(L^2 d)$, with all state (≤ 64 positions, ≤ 32 selectors) in registers and local memory. Across the pass there are at most K successful augmentations and matched structure is reused between calls, so this is far cheaper than the K full-graph augmentations of the primal–dual solver—but it retains a search per call, so the speedup is smaller than for the other two oracles (Table 9).

G.6 Verification

Corollary 1 reduces correctness of the solver to correctness of the oracle, so we test the oracle directly.

Exhaustive at $L = 16$. The oracle is a predicate over 2^{16} subsets, so we enumerate all of them and compare every oracle's answer against a reference bipartite matching test. Agreement on all $65{,}536$

Algorithm 4 Augmenting-path oracle: EXTENDABLE for any degree-d connectivity

state σ: partial matching colOf$[i]{=}-1$, selOf$[j]{=}-1$; owners$[j]$ ($\leq d$ selectors)
function EXTENDABLE+COMMIT(σ, c) ▷ fused test-and-augment
  BFS from position c over alternating (position → wired selector → matched position) edges
  if a free selector is reached **then**
    flip the matching along the augmenting path; **return true**
  else return false ▷ σ unchanged
  end if
end function

Algorithm 5 Union–find (bicircular) oracle: EXTENDABLE for degree-two connectivity (OP-2D, OP-F2D)

state σ: parent$[i]{=}i$, *cyclic*$[i]{=}$FALSE for $i \in \{0, \ldots, K{-}1\}$
function FIND(x)
  while parent$[x] \neq x$ **do**
    parent$[x] \leftarrow$ parent[parent$[x]]$; $x \leftarrow$ parent$[x]$ ▷ path halving
  end while
  return x
end function
function EXTENDABLE+COMMIT(σ, c)
  $(u, v) \leftarrow$ owners$[c]$; $r_u \leftarrow$ FIND(u); $r_v \leftarrow$ FIND(v)
  if $r_u = r_v$ **then** ▷ edge would close a cycle
    if $\neg$ *cyclic*$[r_u]$ **then**
      cyclic$[r_u] \leftarrow$ TRUE; **return true** ▷ the one cycle allowed
    else return false ▷ a second cycle is infeasible
    end if
  else
    if *cyclic*$[r_u] \wedge$ *cyclic*$[r_v]$ **then**
      return false ▷ merge would hold two cycles
    else
      parent$[r_u] \leftarrow r_v$; *cyclic*$[r_v] \leftarrow$ *cyclic*$[r_u] \vee$ *cyclic*$[r_v]$; **return true**
    end if
  end if
end function

subsets establishes that the oracle *is* the feasibility predicate, so by Corollary 1 the greedy solver is optimal for *every* saliency vector at $L = 16$—a guarantee over all inputs, not a sample-based one.

Sampled at $L = 64$. Enumeration is impossible, so we compare end to end. Over randomly sampled saliency vectors per pattern, we check that the fast solver's retained saliency equals the primal–dual optimum, and that the returned mask is exactly 50% dense and admits a perfect matching. Masks may differ from the primal–dual solution where saliencies tie; the totals never do.

Both checks passed with no discrepancies, so the speedups below come at identical mask quality.

G.7 KERNEL STRUCTURE AND MEASURED COST

All variants process one tile per thread and eliminate the generic solver's data-dependent iteration counts, dual bookkeeping, and global memory traffic. Working state is register-resident; the fixed $L \in \{16, 64\}$ trip counts let the compiler fully unroll the sort and the scan; and the only shared read-only structure is the 8 KiB table ($L = 16$) or the small owners list ($L = 64$), both cache-resident. The result is coalesced loads, uniform per-thread work, and minimal divergence.

Because the skeleton is fixed, the speedups track oracle cost alone (Table 9). At $L = 16$ the $O(1)$ bit test gives 47.3–55.0× for OP-1D and 62.8–77.5× for OP-F2D over the primal–dual solver. At $L = 64$ the oracle runs online: OP-1D's $O(L^2 d)$ augmenting-path test gives 3.07–3.13×, while OP-F2D's degree-two structure reduces the oracle to the $O(\alpha(L))$ union–find test and recovers 13.1–15.5×, closing most of the gap back toward the $L = 16$ table within the same skeleton. In absolute terms the deployed OP-F2D $L = 64$ solver costs 1.3–1.6× a non-overlapping 2:4 top-k

baseline. Memory behaves the same way across oracles: the fast solvers allocate only the output mask, while `topk` baselines additionally materialize values, int64 indices, and sort buffers (Table 10).

Table 9: Mask-generation latency on Llama 3.1 8B linear projections (NVIDIA GB10, FP16, idle GPU), in ms per call, averaged over 200 timed iterations after 30 warmup iterations; speedups are baseline ÷ fast. Every fast solver shares the skeleton of Algorithm 1 and differs only in the feasibility oracle (§G.3–§G.5). All retain saliency identical to the assignment baseline. Non-overlapping patterns need no assignment solver and use `topk`.

	Q/O 4096×4096	K/V 1024×4096	Gate/up 14336×4096	Down 4096×14336
OP-1D (overlapping 2:4 line)				
$L = 16$				
Assignment (baseline)	137.3	34.3	482.4	482.9
Rank-greedy (fast)	2.61	0.73	8.78	8.78
Speedup	52.6×	47.3×	54.9×	55.0×
$L = 64$				
Assignment (baseline)	257.1	64.9	891.4	891.5
Rank-greedy (fast)	82.0	21.1	289.5	289.5
Speedup	3.13×	3.07×	3.08×	3.08×
OP-F2D (flattened 2D)				
$L = 16$				
Assignment (baseline)	185.0	45.4	642.0	640.0
Rank-greedy (fast)	2.49	0.72	8.28	8.29
Speedup	74.2×	62.8×	77.5×	77.2×
$L = 64$				
Assignment (baseline)	363.6	92.5	1256.0	1255.2
Rank-greedy (fast)	25.8	7.06	81.0	81.1
Speedup	14.1×	13.1×	15.5×	15.5×
Non-overlapping baselines (no assignment solver)				
2:4 `topk`	18.22	4.50	63.60	63.58
4:8 `topk`	9.73	2.36	33.98	34.04
8:16 `topk`	5.32	1.28	18.55	18.55
Unstructured 50% `topk`	2.14	0.47	7.45	20.66

Table 10: Peak solver working-set memory on Llama 3 8B linear projections (NVIDIA GB10, FP16), in MiB allocated by the generator *beyond the input tensor*, from per-process PyTorch allocator stats (whole-process high-water mark over the full call). "Output mask" is the returned boolean mask, one byte per weight. The OP fast solvers allocate only the output mask; `topk` paths also materialize top-k values, int64 indices, and scatter/sort buffers. The $L = 16$ legality table (8 KiB) and $L = 64$ `owners` list (≈1 KiB) are negligible and omitted.

	Q/O 4096×4096	K/V 1024×4096	Gate/up 14336×4096	Down 4096×14336
Input (FP16)	32.0	8.0	112.0	112.0
Output mask (bool)	16.0	4.0	56.0	56.0
Peak memory allocated by the generator (beyond input)				
OP-1D / OP-F2D, $L = 16$	16.0	4.0	56.0	56.0
OP-1D / OP-F2D, $L = 64$	16.0	4.0	56.0	56.0
2:4 / 4:8 / 8:16 (`topk`)	80.0	20.0	280.0	280.0
Unstructured 50% (`topk`)	86.1	21.5	301.4	1064.0

H HARDWARE COST: FLOW, FLOORPLAN, AND ABSOLUTE RESULTS

This appendix reports the absolute measurements behind the ratios of Table 2, together with the flow, PDK, constraints, floorplan procedure, and metric definitions used to obtain them.

H.1 DESIGN UNDER TEST

We synthesize the *selection network only*, modelled as a monolithic block in which all L activations are available to every selector; whether a given accelerator exposes the selection network this way or partitions it across lanes is an architectural choice outside our scope. For a tile of $L = 64$ positions at b-bit precision, every design takes 64 b-bit inputs and produces 32 b-bit outputs through $K = 32$ selectors. The designs differ in selector fan-in and in the index width that fan-in requires:

Design	Selector	Index bits	Wiring of each selector's input ports
2:4 (control)	4:1	$2 \times 32 = 64$	disjoint windows $\{4q, \ldots, 4q{+}3\}$
4:8	8:1	$3 \times 32 = 96$	disjoint windows of 8
8:16	16:1	$4 \times 32 = 128$	disjoint windows of 16
OP-1D	4:1	$2 \times 32 = 64$	overlapping windows
OP-F2D	4:1	$2 \times 32 = 64$	row-major flattening of the $8{\times}8$ tile

The index-bit column is the hardware form of the index cost discussed in §3.6: both OP variants carry the 2-bit per kept weight metadata of 2:4 and instantiate the *same* $4{:}1$ selector primitive at the same count and fan-in, differing only in which top-level net reaches which selector port. 4:8 and 8:16 instead require $1.5\times$ and $2\times$ as many index pins and correspondingly deeper selectors. This distinction matters when reading Tables 12 and 13: among 2:4, OP-1D, and OP-F2D the logic is identical by construction and any measured difference is a placement and routing effect, whereas 4:8 and 8:16 differ in logic as well as wiring.

Multipliers, adder tree, and accumulator lie outside the synthesized block. They are bit-identical across designs by construction (Figure 1e), so including them would add the same constant to all five and compress the very differences these tables are meant to expose. All reported figures are therefore *selector* quantities; §H.6 converts them to a per-MAC bound. In particular, the delay column is the selector's input-to-output path delay, not a MAC cycle time.

H.2 FLOW, PDK, AND CONSTRAINTS

All designs pass through an identical open-source RTL-to-GDSII flow: LibreLane `3.0.14` (Shalan & Edwards, 2020) driving Yosys `0.62` (Wolf & Glaser, 2013) for synthesis, OpenROAD (`dcf36133a369abc8f3c5e5738cd4d82e4903c0e0`) (Ajayi et al., 2019) for floorplanning, I/O placement, PDN generation, placement, clock-tree synthesis, routing, fill insertion, and parasitic extraction, OpenSTA `2.7.0` (Cherry & The OpenROAD Project, 2020) for post-route timing and power, KLayout for GDSII streamout, Magic for DRC and SPICE extraction, and Netgen for LVS. The target PDK is the open-source SkyWater sky130A process (130 nm) (SkyWater Technology Foundry & Google LLC, 2020), revision. We report the `nom_tt_025C_1v80` corner (typical–typical, 25°C, 1.80 V) throughout.

Hierarchy is preserved at the selector boundary (`SYNTH_HIERARCHY_MODE` = `"keep"` with the selector module listed in `SYNTH_KEEP_HIERARCHY_MODULES`) so that all designs are compared at the same structural granularity, preventing synthesis from dissolving one design's selectors into surrounding logic but not another's. Absolute areas are therefore conservative: they are not the minimum a fully flattened synthesis would reach.

Every design completes detailed routing with zero routing DRC violations and passes Magic DRC and Netgen LVS at all three precisions. Two designs do not meet the 5.0 ns timing constraint: 4:8 at 16-bit (-0.131 ns setup slack) and 8:16 at 16-bit (-4.570 ns). We report their measured path delay rather than relaxing the constraint or enlarging their floorplans, because the violation is one of logic depth — a $16{:}1$ selector is deeper than a $4{:}1$ one — and is not remediable by additional area.

Because the selection network is purely combinational, we constrain it against a virtual clock with a 5.0 ns period, zero input and output delay, and `set_max_delay 5.0` from all inputs to all outputs.

The reported combinational delay is 5.0 ns minus the post-route worst setup slack. The same SDC is applied to all five designs at all three precisions. There is no clock network, which is why total power and combinational power agree to within a few nanowatts in every run (Table 13).

Switching power is computed from the static timing engine's default activity model rather than from annotated inference traces. Absolute power is consequently not a workload-specific figure; we report it only as a ratio between designs synthesized, placed, routed, and analyzed under identical assumptions, which is the quantity the claims in §3.6 depend on. One consequence is that small power differences between the three designs with identical logic are not evidence of superiority in either direction; see §H.5.

Metric provenance. All figures are read programmatically from step metrics rather than parsed from logs:

Reported quantity	Metric key	Step
Cell area, stdcell	`design__instance__area__stdcell`	STA post-PnR
Core area	`design__core__area`	STA post-PnR
Die area	`design__die__area`	STA post-PnR
Core utilization	`design__instance__utilization`	STA post-PnR
Setup / hold slack	`timing__setup__ws` / `timing__hold__ws`	STA post-PnR
GRT wirelength	`route__wirelength__estimated`	Global routing
DRT wirelength	`route__wirelength`	Detailed routing
Vias	`route__vias`	Detailed routing
Routing DRC	`route__drc_errors`	Detailed routing
Power	`power.rpt`, corner `nom_tt_025C_1v80`	STA post-PnR

Cell area is the post-place-and-route *standard-cell* area, which excludes fill and decap cells; we do not report the fill-inflated total instance area, nor the pre-route synthesis area estimate. Utilization is likewise the pre-fill figure.

H.3 Floorplan

Because the designs are not area-substitutable — 4:8 and 8:16 require wider selectors and more index pins — no single floorplan can host all five. We therefore fix a *procedure* rather than a geometry, and report the geometry it produces for every design.

Sizing procedure. We first harden the 2:4 control. Its die is set to the smallest square that accommodates the top-level pin ring at the PDK minimum pin pitch, and its core is then shrunk to the smallest square that still completes detailed routing with zero violations. That geometry becomes the common starting point for every other design at the same precision. A design that does not route at the starting geometry is grown by a single rule, whose clauses are applied in order until routing closes: (i) enlarge the core by $5\,\mu$m per side, preserving a square aspect ratio; (ii) if the pin ring no longer fits, adopt the minimum die size the tool reports as sufficient for I/O placement, then resume (i); (iii) once the core is within $20\,\mu$m of the die boundary on all sides, grow die and core together, holding that $20\,\mu$m margin. Only 4:8 and 8:16 ever invoked (ii). The search is deterministic given the tool versions of §H.2 and halts at the first geometry that routes, so no design receives a floorplan tuned beyond what its own routing demand requires. The procedure is, if anything, favorable to the $N{:}M$ baselines: a design that needs more area is given it, and extra area reduces routing pressure.

What the procedure controls, and what it does not. Core area is an *output* of this search, not a setting, and so is utilization: at 16-bit the five designs land between 28.8% and 47.7%. We therefore do not treat utilization as a cost metric, and it is comparable neither across designs nor across precisions. The consequence for reading Table 12 is that a design's cost is divided between two columns in a design-dependent way: one that fits the starting geometry pays entirely in wire and vias, while one that must grow pays partly in core area and correspondingly less in wire. OP-F2D exhibits both regimes. It fits the baseline core at 4-bit and 8-bit, paying $+9.3\%$ and $+25.0\%$ detailed-route wirelength at no area cost; at 16-bit it takes two growth steps ($+9.2\%$ core) and its wirelength penalty falls to $+17.4\%$. Area and wirelength must be read jointly; neither column alone is the cost of a design.

Two limits of the search should be stated plainly. First, the $5\,\mu$m step is coarse relative to the effects being measured: on the 16-bit control's core it corresponds to a $\sim 4.4\%$ change in core area, larger than the 0.6–4.2% cell-area differences between 2:4 and OP-F2D. The core-area column therefore has a resolution of one search step, and small differences within it are not meaningful; the routing columns are the finer-grained evidence. Second, "smallest routable" is a property of the router rather than of the netlist. We report the first size that routes under the stated tool versions, and for every design we verify that the accepted size routes and the preceding size fails under independent global-placement seeds.

Table 11: Per-design floorplan geometry produced by the sizing procedure of §H.3. *Total pins* counts data inputs, index pins, and outputs; *ring* is the perimeter those pins require at the $0.92\,\mu$m minimum pitch. *Die* and *core* are the requested `DIE_AREA` and `CORE_AREA`; *core area* and *util.* are as reported post-route by OpenROAD (`design__core__area`, `design__instance__utilization`) and are outputs of the search, not settings.

Prec.	Design	Index pins	Total pins	Ring (μm)	Die (μm)	Core (μm)	Core area (μm^2)	Util. (%)
16-bit	2:4	64	1600	1472	530×530	230×230	51,821	39.22
	4:8	96	1632	1501	530×530	260×260	67,039	47.67
	8:16	128	1664	1531	630×630	470×470	219,726	28.84
	OP-1D	64	1600	1472	530×530	230×230	51,821	39.29
	OP-F2D	64	1600	1472	530×530	240×240	56,604	36.77
8-bit	2:4	64	832	765	235×235	125×125	15,258	62.56
	4:8	96	864	795	295×295	170×170	28,163	80.05
	8:16	128	896	824	335×335	255×255	64,464	52.43
	OP-1D	64	832	765	235×235	125×125	15,258	62.10
	OP-F2D	64	832	765	235×235	125×125	15,258	65.15
4-bit	2:4	64	448	412	135×135	115×115	12,774	44.02
	4:8	96	480	442	155×155	125×125	15,258	81.63
	8:16	128	512	471	220×220	180×180	31,718	68.09
	OP-1D	64	448	412	135×135	115×115	12,774	44.10
	OP-F2D	64	448	412	135×135	115×115	12,774	44.29

Total pins counts data inputs, index pins, and outputs; *ring* is the perimeter those pins require at the minimum pitch; *die* and *core* are the requested `DIE_AREA` and `CORE_AREA`; *core area* and *utilization* are as reported post-route by OpenROAD (`design__core__area`, `design__instance__utilization`).

I/O pins are placed at the PDK minimum pitch ($0.92\,\mu$m) with unit thickness and extension multipliers (`IO_PIN_V_THICKNESS_MULT` $=$ `IO_PIN_H_EXTENSION_MULT` $= 1$), the smallest legal geometry the PDK permits. The power distribution network uses $2\,\mu$m straps on a $30\,\mu$m pitch in both directions, identical for every design. Each design is hardened as a macro, so top-level ports are metal pin shapes on the routing layers rather than I/O pads; no pad cells or padframe are instantiated. The pin ring sets the *initial* die size but is not the binding constraint at the final geometry: the 16-bit control's $1{,}600$ pins require $1{,}472\,\mu$m of perimeter, well inside what its routability-driven die provides. Routability sets the final size in every case.

H.4 ABSOLUTE RESULTS

Table 12: Absolute post-place-and-route selector area, timing, and routing at $L = 64$ (1×64 windows for 2:4, 4:8, 8:16, and OP-1D; 8×8 flattened for OP-F2D). Delay is 5.0 ns minus post-route worst setup slack; negative setup slack indicates the design does not meet the 5.0 ns constraint. Core area is an output of the sizing procedure of §H.3, so utilization is not comparable across columns or across precision blocks.

	N:M			Ours	
Metric	2:4 (control)	4:8	8:16	OP-1D	OP-F2D
16-bit					
Cell area, stdcell (μm^2)	20,315	31,961	63,371	20,362	20,814
Core area (μm^2)	51,821	67,039	219,726	51,821	56,604
Core utilization (%)	39.22	47.67	28.84	39.29	36.77
Setup slack (ns)	2.193	-0.131	-4.570	2.375	2.315
Combinational delay (ns)	2.81	5.13	9.57	2.63	2.69
Hold slack (ns)	0.117	0.114	0.0906	0.106	0.106
Global-route wirelength (μm)	545,891	608,071	888,724	562,586	625,173
Detailed-route wirelength (μm)	590,379	710,166	1,201,749	610,591	693,310
Vias	28,389	50,112	104,535	30,764	34,964
8-bit					
Cell area, stdcell (μm^2)	9,545	22,545	33,800	9,475	9,941
Core area (μm^2)	15,258	28,163	64,464	15,258	15,258
Core utilization (%)	62.56	80.05	52.43	62.10	65.15
Setup slack (ns)	3.138	1.902	0.777	3.148	3.066
Combinational delay (ns)	1.86	3.10	4.22	1.85	1.93
Hold slack (ns)	0.062	0.067	0.077	0.070	0.068
Global-route wirelength (μm)	117,418	179,573	260,733	117,736	143,129
Detailed-route wirelength (μm)	126,010	215,822	369,566	123,976	157,515
Vias	12,473	28,992	50,446	12,295	14,885
4-bit					
Cell area, stdcell (μm^2)	5,623	12,456	21,597	5,633	5,658
Core area (μm^2)	12,774	15,258	31,718	12,774	12,774
Core utilization (%)	44.02	81.63	68.09	44.10	44.29
Setup slack (ns)	3.570	2.558	1.303	3.593	3.586
Combinational delay (ns)	1.43	2.44	3.70	1.41	1.41
Hold slack (ns)	0.054	0.051	0.123	0.057	0.044
Global-route wirelength (μm)	41,636	60,186	115,026	41,626	45,504
Detailed-route wirelength (μm)	45,203	74,431	156,775	44,399	49,418
Vias	6,399	14,940	28,040	6,206	6,750

H.5 INTERPRETING THE MEASUREMENTS

The differences among equal-logic designs are routing, not logic. Among the three designs that instantiate identical $4{:}1$ selectors — 2:4, OP-1D, and OP-F2D — internal power, which depends on cell count and cell type, varies by at most 0.4% at 16-bit, 1.4% at 8-bit, and 0.7% at 4-bit, and leakage is bit-identical (6.2985×10^{-9} W at 16-bit, 4.0221×10^{-9} W at 8-bit, 2.4765×10^{-9} W at 4-bit). Synthesis therefore drew on the same threshold-voltage flavours and drive strengths in every case. Switching power varies far more and tracks wirelength rather than cell area: at 16-bit OP-F2D adds 2.5% cell area but 17.4% detailed-route wirelength and 23.2% vias, and its switching power rises 13.0%. This is the expected signature of the flattened 8×8 connectivity, in which reaching across the row-major mapping demands longer nets and more layer changes. By contrast, 4:8 and 8:16 differ in logic as well as wiring: at 16-bit their internal power rises 82% and 178% and their leakage $1.9\times$ and $3.3\times$ over the control, because a deeper selector is a larger circuit.

Delay. Relative to 2:4, OP-1D is faster at all three precisions and OP-F2D is faster at 16-bit and 4-bit but 3.8% slower at 8-bit, all while holding selector count and fan-in fixed. We do not claim a delay advantage for either OP variant: the spread across these three designs is smaller than the range OP-F2D alone spans across precisions, which is what one expects when the underlying logic

Table 13: Absolute post-place-and-route power at the `nom_tt_025C_1v80` corner. The designs are purely combinational, so total power including the clock network agrees with combinational total to within a few nanowatts (5.7 nW at 4-bit). Leakage contributes under 0.001% of total power in this flow and is reported only as evidence that synthesis drew on the same cell mix: it is bit-identical across 2:4, OP-1D, and OP-F2D at each precision, and rises with selector depth for 4:8 and 8:16.

	N:M			Ours	
Power (W)	2:4 (control)	4:8	8:16	OP-1D	OP-F2D
16-bit					
Internal	9.44×10^{-4}	1.72×10^{-3}	2.62×10^{-3}	9.46×10^{-4}	9.48×10^{-4}
Switching	2.61×10^{-3}	4.03×10^{-3}	6.36×10^{-3}	2.64×10^{-3}	2.95×10^{-3}
Leakage	6.30×10^{-9}	1.21×10^{-8}	2.10×10^{-8}	6.30×10^{-9}	6.30×10^{-9}
Combinational total	3.55×10^{-3}	5.75×10^{-3}	8.98×10^{-3}	3.58×10^{-3}	3.90×10^{-3}
Including clock	3.55×10^{-3}	5.75×10^{-3}	8.98×10^{-3}	3.58×10^{-3}	3.90×10^{-3}
8-bit					
Internal	4.88×10^{-4}	9.09×10^{-4}	1.35×10^{-3}	4.88×10^{-4}	4.95×10^{-4}
Switching	6.08×10^{-4}	1.08×10^{-3}	1.63×10^{-3}	5.82×10^{-4}	7.83×10^{-4}
Leakage	4.02×10^{-9}	9.67×10^{-9}	1.30×10^{-8}	4.02×10^{-9}	4.02×10^{-9}
Combinational total	1.10×10^{-3}	1.99×10^{-3}	2.98×10^{-3}	1.07×10^{-3}	1.28×10^{-3}
Including clock	1.10×10^{-3}	1.99×10^{-3}	2.98×10^{-3}	1.07×10^{-3}	1.28×10^{-3}
4-bit					
Internal	2.53×10^{-4}	4.86×10^{-4}	7.00×10^{-4}	2.53×10^{-4}	2.51×10^{-4}
Switching	2.82×10^{-4}	4.36×10^{-4}	6.93×10^{-4}	2.76×10^{-4}	2.45×10^{-4}
Leakage	2.48×10^{-9}	5.57×10^{-9}	7.81×10^{-9}	2.48×10^{-9}	2.48×10^{-9}
Combinational total	5.35×10^{-4}	9.21×10^{-4}	1.39×10^{-3}	5.29×10^{-4}	4.96×10^{-4}
Including clock	5.35×10^{-4}	9.21×10^{-4}	1.39×10^{-3}	5.29×10^{-4}	4.96×10^{-4}

is identical and only net length differs. 4:8 and 8:16 are slower by 1.7–3.4× and miss the 5.0 ns constraint at 16-bit, which *is* a logic-depth effect and is the point of the comparison. Hold slack remains positive for every design at every precision.

Power under a vectorless activity model. OP-F2D's combinational total is $+10\%$ at 16-bit, $+16\%$ at 8-bit, and -7% at 4-bit relative to the control. The 4-bit decrease comes almost entirely from switching power (2.823×10^{-4} to 2.447×10^{-4} W, -13.3%), with internal power flat and leakage bit-identical. We do not read this as evidence that OP-F2D is intrinsically more efficient at low precision. Because switching activity is propagated from the tool's default statistical model rather than annotated from inference traces (§H.2), differences of this size between designs whose logic is identical by construction reflect buffer insertion and drive-strength sizing under a vectorless assumption, not a workload-level power difference. The conservative reading, which we adopt in §H.6, is to charge the *worst* observed delta across all precisions to the selector and ignore the favourable one.

The measured deltas are an upper bound. Because 2:4, OP-1D, and OP-F2D instantiate identical selector primitives at identical count and fan-in, the intrinsic logic ratio is 1.000; what Table 12 records is the buffering and sizing that timing closure adds in response to net length. Two aspects of our setup make that response pessimistic.

First, placement is unconstrained. OP-F2D's row-major flattening is a *logical* relabeling, chosen so that all products accumulate into a single destination; it says nothing about layout. Half of its selectors read positions eight apart in the flattened index, and with I/O pins ordered by index these become the longest nets in the design. Placed instead on the 8×8 grid the mapping came from, a stride of eight is simply the cell one row below, and every selector becomes nearest-neighbour. We hand the global placer a flat netlist with no such hint, so it reproduces the worst case rather than the folded layout assumed in §3.3. Second, sky130A provides five routing layers against fifteen or more in contemporary processes, so congestion-driven buffering is more severe here than it would be on a modern stack.

We therefore read the OP-F2D column as a conservative bound: at worst $+4.2\%$ cell area and $+25.0\%$ detailed-route wirelength, both at 8-bit, are what the structure costs when nothing is done to place it well.

H.6 From selector cost to per-MAC cost

The ratios in Table 2 describe the selection network in isolation. To place them at the MAC level we use the reported selector share of a sparse tensor-core MAC, 1.28% of area and 1.78% of power at 15 nm (Gondimalla et al., 2023). Charging the worst-case measured delta over all three precisions entirely to the selector gives $1.28\% \times 4.2\% = 0.053\%$ of MAC area and $1.78\% \times 16.4\% = 0.29\%$ of MAC power.

This transfers a ratio measured at 15 nm on a different tensor-core design to a 130 nm implementation. We use it because the transferred quantity is a *share* of a like-for-like structure rather than an absolute, and because both terms would have to be wrong by more than an order of magnitude to change the conclusion. It nonetheless remains an estimate, and the resulting bound should be read as an order of magnitude rather than a measurement. The two nodes differ in the direction that matters here: relative to a contemporary process, sky130A's coarser pitch and shallower metal stack inflate the routing-driven component of our measurements, so applying that worst case against a 15 nm selector share is conservative in both terms.

I Detailed Experimental Setup

Table 14: Model-quality experiment configuration.

Setting	Value
Model precision	BF16
Target sparsity	50%
Calibration data	C4, 128 sequences
Sequence length	2048
Primary seed	0
Unstructured baseline	Row-wise 50%
$N{:}M$ baselines	2:4, 4:8, 8:16
Main OP tile width	64
OP tile ablations	16, 64, 256
Wanda++ optimizer	RMSprop
Wanda++ learning rate	1×10^{-5}
Wanda++ RGS α	100
Wanda++ RO rounds	5
RO samples per round	32
RIA γ	0.5
Perplexity	WikiText-2 test
Zero-shot tasks	8 tasks, 0-shot
Compute	NVIDIA DGX Spark (GB10)
Memory	128 GB LPDDR5X unified
Memory bandwidth	273 GB/s

Models and compute. We evaluate Llama 7B, 13B, and 30B; Llama 2 7B and 13B; Llama 3 8B; Llama 3.1 8B; Qwen 2.5 7B and 32B; Gemma 2 9B; and OPT 6.7B. We use the Hugging Face checkpoints `huggyllama/llama-7b`, `huggyllama/llama-13b`, `huggyllama/llama-30b`, `meta-llama/Llama-2-7b-hf`, `meta-llama/Llama-2-13b-hf`, `meta-llama/Meta-Llama-3-8B` (for Table 5 only), `meta-llama/Llama-3.1-8B`, `Qwen/Qwen2.5-7B`, `Qwen/Qwen2.5-32B`, `google/gemma-2-9b`, and `facebook/opt-6.7b`. Models are loaded in BF16.

All model-side pruning and evaluation experiments are run on NVIDIA DGX Spark systems based on the GB10 platform. Each system contains an NVIDIA Blackwell GPU with fifth-generation Tensor Cores, a 20-core Arm CPU (10 Cortex-X925 and 10 Cortex-A725 cores), and 128 GB of LPDDR5X unified memory with a 256-bit interface and 273 GB/s memory bandwidth.

Calibration and pruning procedure. Wanda, RIA, and Wanda++ use 128 calibration sequences from the English C4 training set. Each sequence contains 2048 contiguous tokens sampled from a document longer than the sequence length. For a fixed model and random seed, the same calibration samples are reused across all sparsity structures.

We prune the weight matrices of linear layers in the transformer decoder blocks, including attention and MLP projections. Embeddings, normalization layers, biases, and the final language-model head are not pruned. Decoder blocks are processed sequentially: statistics are collected for one block, its pruning mask is applied, and the resulting activations are propagated to the next block.

Pruning methods. For Wanda, we use the standard activation-aware score

$$s_{ij} = |W_{ij}| \cdot ||X_j||_2,$$

where X is the input activation of input channel j. For RIA, we use the relative weight-importance score combined with activation importance, with activation exponent $\gamma = 0.5$.

For Wanda++, we use Regional Gradient Scoring (RGS) and Regional Optimization (RO). RGS uses coefficient $\alpha = 100$. RO minimizes the mean-squared error between the outputs of the dense decoder block and its pruned counterpart. We use RMSprop with learning rate 1×10^{-5}, five RO rounds per decoder block, and 32 randomly selected calibration samples per round. The same Wanda++ hyperparameters are used for all models and sparsity structures.

Sparsity structures. All experiments use 50% weight sparsity. We compare row-wise unstructured 50% pruning, conventional 2:4, 4:8, and 8:16 sparsity, and our OP-1D and OP-F2D patterns. The unstructured baseline prunes the lowest-scoring 50% of weights independently within each row.

The main OP experiments use tile width $L = 64$. OP-1D operates on 1×64 row tiles, while OP-F2D uses the flattened connectivity corresponding to an 8×8 logical tile. Tile-size ablations additionally use $L = 16$ and $L = 256$, corresponding to 4×4 and 16×16 logical OP-F2D tiles. Unless explicitly stated otherwise, channel permutation is disabled.

Mask generation. For OP-1D and OP-F2D, each weight matrix is divided into independent tiles. Within each tile, we maximize the total retained saliency subject to the OP selector constraints using the exact assignment solver described in § 3.5. All reported model-quality results therefore use exact, rather than approximate, OP masks.

Perplexity evaluation. We evaluate perplexity on the `wikitext-2-raw-v1` test split. The tokenized text is divided into non-overlapping windows, with incomplete final windows discarded. The main results use sequence length 2048 and batch size 1. Longer-context comparisons use the sequence lengths stated in the corresponding tables.

Zero-shot evaluation. We evaluate ARC-Challenge, ARC-Easy, BoolQ, HellaSwag, OpenBookQA, PIQA, RTE, and Winogrande using the `lm-eval` harness with `num_fewshot=0` and batch size 1. We report the standard accuracy for each task and use the unweighted mean of the eight task accuracies as the aggregate zero-shot score.

Evaluation context lengths. For perplexity evaluation, we use context lengths of 2K, 4K, and 8K tokens, depending on the comparison setting. Our main WikiText-2 results use a 2K context length, while the 4K and 8K evaluations are used for direct comparison with the reported results of PermLLM, MaskLLM, and ARMOR at their corresponding context lengths. For zero-shot evaluation, we use a 2K context length for all eight tasks.

Randomness and timing. The primary experiments use random seed 0. Python, NumPy, and PyTorch are seeded before each run. For the calibration-seed robustness experiment on Llama 3.1 8B, we use seeds $\{48288, 70362, 12041, 7998, 33580\}$ while keeping all other settings fixed.

Each pruning configuration starts from the same dense pretrained checkpoint and uses the same calibration samples. Pruning time is measured separately from perplexity and zero-shot evaluation. Unless otherwise stated, reported pruning runtimes exclude downstream evaluation.

Pruning runtime. Table 15 reports the end-to-end pruning time for the evaluated sparsity patterns and pruning methods. All runtimes are reported in hours and were measured on a single NVIDIA DGX Spark system (GB10, 128 GB unified memory). For the OP patterns, we use the general exact assignment solver described in § 3.5, consistent with the solver used to construct the masks for the model-quality experiments. The reported runtimes may exhibit minor inconsistencies due to system throttling or interference from other concurrently running jobs.

Table 15: Pruning wall-clock time in hours on a single NVIDIA DGX Spark system (GB10, 128 GB unified memory), using the general solver for OP patterns. Downstream perplexity and zero-shot evaluation are excluded.

		Unstr.	N:M			OP-1D		OP-F2D	
Model (Dense)	**Method**	**50%**	**8:16**	**4:8**	**2:4**	**16**	**64**	**16**	**64**
Llama 7B	Magnitude	0.00	0.00	0.00	0.00	0.01	0.03	0.02	0.04
	Wanda	0.06	0.06	0.06	0.06	0.05	0.08	0.06	0.09
	RIA	0.06	0.06	0.06	0.06	0.05	0.08	0.06	0.09
	Wanda++	0.58	0.59	0.58	0.58	0.52	0.70	0.55	0.76
Llama 13B	Magnitude	0.00	0.01	0.00	0.00	0.03	0.04	0.04	0.07
	Wanda	0.11	0.11	0.11	0.10	0.10	0.15	0.11	0.17
	RIA	0.11	0.11	0.11	0.10	0.10	0.15	0.11	0.17
	Wanda++	1.10	1.11	1.10	1.09	0.96	1.38	1.00	1.51
Llama 30B	Magnitude	0.01	0.01	0.01	0.01	0.08	0.12	0.10	0.17
	Wanda	0.17	0.17	0.17	0.17	0.25	0.29	0.27	0.33
	RIA	0.17	0.18	0.18	0.17	0.25	0.29	0.27	0.33
	Wanda++	1.95	1.96	1.92	1.91	2.34	2.55	2.48	2.83
Llama 2 7B	Magnitude	0.00	0.00	0.00	0.00	0.01	0.02	0.02	0.03
	Wanda	0.05	0.06	0.06	0.05	0.06	0.08	0.06	0.08
	RIA	0.06	0.06	0.06	0.06	0.06	0.08	0.06	0.09
	Wanda++	0.58	0.59	0.59	0.58	0.52	0.73	0.55	0.79
Llama 2 13B	Magnitude	0.00	0.01	0.00	0.00	0.03	0.04	0.04	0.06
	Wanda	0.11	0.11	0.10	0.10	0.10	0.15	0.11	0.17
	RIA	0.11	0.11	0.10	0.10	0.10	0.15	0.11	0.17
	Wanda++	1.10	1.11	1.10	1.09	0.97	1.39	1.01	1.52
Llama 3.1 8B	Magnitude	0.00	0.00	0.00	0.00	0.02	0.02	0.02	0.03
	Wanda	0.06	0.06	0.06	0.06	0.06	0.08	0.06	0.09
	RIA	0.06	0.06	0.06	0.06	0.06	0.08	0.06	0.09
	Wanda++	0.62	0.62	0.62	0.61	0.54	0.75	0.57	0.81
Qwen 2.5 7B	Magnitude	0.00	0.00	0.00	0.00	0.02	0.02	0.02	0.03
	Wanda	0.05	0.05	0.05	0.05	0.05	0.08	0.06	0.09
	RIA	0.05	0.05	0.05	0.05	0.05	0.08	0.06	0.09
	Wanda++	0.58	0.56	0.56	0.56	0.49	0.69	0.52	0.74
Qwen 2.5 32B	Magnitude	0.01	0.01	0.01	0.01	0.07	0.10	0.09	0.15
	Wanda	0.18	0.18	0.17	0.17	0.24	0.27	0.26	0.31
	RIA	0.17	0.18	0.17	0.17	0.24	0.28	0.26	0.32
	Wanda++	1.87	1.90	1.87	1.85	2.27	2.47	2.39	2.73
Gemma 2 9B	Magnitude	0.00	0.00	0.00	0.00	0.02	0.03	0.02	0.04
	Wanda	0.08	0.08	0.08	0.08	0.08	0.11	0.08	0.12
	RIA	0.08	0.08	0.08	0.08	0.08	0.11	0.08	0.12
	Wanda++	1.10	0.87	0.86	0.86	0.73	1.02	0.77	1.09
OPT 6.7B	Magnitude	0.00	0.00	0.00	0.00	0.01	0.02	0.02	0.03
	Wanda	0.05	0.05	0.05	0.05	0.05	0.07	0.05	0.08
	RIA	0.05	0.05	0.05	0.05	0.05	0.07	0.05	0.08
	Wanda++	0.47	0.48	0.48	0.47	0.47	0.60	0.49	0.66

J Robustness Studies

Across all three robustness studies, the relative ordering of methods and sparsity patterns established in our main tables is preserved.

J.1 Random Seed Sensitivity

We repeat pruning on Llama 3.1 8B with five calibration seeds: 48288, 70362, 12041, 7998, and 33580. All other pruning and evaluation settings are held fixed. The seeds determine the sampled calibration data used by Wanda, Wanda++, and RIA; magnitude pruning is calibration-free and therefore has no seed-dependent variation.

Table 16: Random-seed sensitivity of Llama 3.1 8B pruning at 50% sparsity. Values are WikiText-2 perplexity (↓) at sequence length 2048, reported as mean ± standard deviation across five calibration seeds. Magnitude pruning is deterministic and is shown once.

Sparsity Type	**Random Seed Test (Llama3.1-8B)**			
	Magnitude	**Wanda**	**Wanda++**	**RIA**
Unstructured (50%)	33.97	9.70 ± 0.02	8.85 ± 0.02	9.41 ± 0.03
Structured 2:4	789.96	22.92 ± 0.25	14.09 ± 0.13	22.58 ± 0.24
Structured 4:8	212.00	13.85 ± 0.02	10.93 ± 0.08	13.49 ± 0.03
Structured 8:16	46.96	11.42 ± 0.04	9.81 ± 0.07	11.09 ± 0.03
Overlapping-1D (1×64)	100.48	13.40 ± 0.09	10.69 ± 0.08	12.94 ± 0.03
Overlapping-F2D (1×64)	58.09	11.55 ± 0.04	9.77 ± 0.06	11.11 ± 0.04

The results are stable across calibration samples. Standard deviations are at most 0.03 for unstructured pruning, 0.09 for OP-1D, and 0.06 for OP-F2D. The largest variation occurs for the more restrictive 2:4 pattern, reaching 0.25 under Wanda and 0.24 under RIA; Wanda++ reduces it to 0.13. These variations are small relative to the gaps between sparsity patterns: under every scorer, the ordering unstructured $<$ OP-F2D $<$ OP-1D $<$ 2:4 is preserved, as is Wanda++'s advantage over Wanda and RIA. The seed-averaged perplexities also closely match the single-seed results in the main tables.

J.2 Score Normalization

Wanda-style saliency scores are reliable as relative rankings within an output row; under standard row-wise pruning, the absolute score scale of one row does not affect another (Sun et al., 2024). However, OP-2D differs because its selector constraints couple neighboring rows within a tile, causing their weights to compete in a shared assignment objective. Directly maximizing raw saliency can therefore allow a row with a large aggregate score to dominate the tile and leave lower-scale rows severely under-selected.

To remove this row-scale bias, we normalize each nonnegative score row before OP-2D mask generation. For a saliency matrix S, we use

$$\widetilde{S}_{ij} = \frac{S_{ij}}{\sum_k S_{ik}}. \tag{11}$$

This transformation preserves every within-row weight ordering while rescaling each row to unit total saliency. It would not change a strict per-row top-k mask, but it changes how OP-2D compares weights across rows. Importantly, normalization does not impose 50% sparsity on each row: OP-2D may still retain different numbers of weights per row while satisfying 50% sparsity over the complete tile. It therefore preserves OP-2D's allocation flexibility while preventing score scale alone from determining which rows receive the retained weights.

Table 17 shows a consistent benefit across tile sizes. With Wanda, normalization reduces perplexity by 6.26–6.76%, averaging 6.51%. The improvement under Wanda++ is smaller but remains nonnegative, averaging 0.32%. This difference is consistent with the Regional Optimizer partially correcting the block-output mismatch produced by an imbalanced mask, whereas frozen-weight Wanda depends

Table 17: Effect of per-row score normalization on Llama 3.1 8B OP-2D pruning at 50% sparsity. Values are WikiText-2 perplexity (↓) at sequence length 2048. $\Delta_{\text{rel}} = (\text{PPL}_{\text{raw}} - \text{PPL}_{\text{norm.}})/\text{PPL}_{\text{raw}}$; positive values indicate improvement.

Sparsity Type	Normalization Test (Llama3.1-8B)					
	Wanda			Wanda++		
	w/o Norm	with Norm	Δ_{rel}	w/o Norm	with Norm	Δ_{rel}
Overlapping 2D (4×4)	12.74	11.91	6.51%	10.23	10.20	0.29%
Overlapping 2D (8×8)	11.96	11.21	6.26%	9.90	9.85	0.56%
Overlapping 2D (16×16)	11.85	11.05	6.76%	9.80	9.79	0.12%

more directly on the initial score calibration. The effect is stable from 4×4 through 16×16, indicating that normalization addresses cross-row score scale rather than a particular window granularity. Unless otherwise stated, all OP-2D results in the main paper use per-row score normalization.

J.3 Learning Rate Sensitivity

Wanda++ introduces a Regional Optimizer (RO) that uses RMSprop to update the surviving weights within each decoder block, minimizing the output discrepancy between the dense and pruned blocks (Yang et al., 2025). The original Wanda++ configuration uses a learning rate of 3×10^{-7}. To assess whether this setting transfers to our implementation and sparsity patterns, we sweep the RO learning rate over $\{3\times10^{-7}, 1\times10^{-6}, 1\times10^{-5}\}$ while holding the remaining pruning and evaluation settings fixed.

Table 18: Learning-rate sensitivity of the Wanda++ Regional Optimizer on Llama 3.1 8B at 50% sparsity. Values are WikiText-2 perplexity (↓) at sequence length 2048. The 3×10^{-7} column uses the learning rate reported by Wanda++; the other columns test larger update steps under otherwise identical settings. OP-F2D and OP-2D results can differ due to the two patterns drawing different weights into the tile (a 1×64 row segment vs an 8×8 block).

Sparsity Type	Perplexity for Different Learning Rates (Llama3.1-8B)		
	3.00×10^{-7}	1.00×10^{-6}	1.00×10^{-5}
Dense	6.24		
Unstructured (50%)	9.58	9.49	8.83
Structured 2:4	23.82	22.09	14.11
Structured 4:8	13.95	13.52	10.95
Structured 8:16	11.41	11.23	9.81
Structured 16:32	10.49	10.36	9.27
Structured 32:64	10.09	9.97	9.05
Overlapping 1D (64)	13.51	13.15	10.70
Overlapping F2D (1×64)	11.56	11.37	9.75
Overlapping 2D (8×8)	11.29	11.17	9.85

Perplexity decreases monotonically for every sparse pattern as the learning rate increases over the tested range, with no observed optimization instability. The largest gain occurs for 2:4, whose perplexity falls from 23.82 at the Wanda++ learning rate to 14.11 at 1×10^{-5}, a 40.8% relative reduction. OP also benefits consistently: OP-1D improves from 13.51 to 10.70 and OP-F2D from 11.56 to 9.75. These results indicate that the published learning rate is overly conservative in our setting and does not fully correct the pruning-induced block-output mismatch. We use an RO learning rate of 1×10^{-5} for all Wanda++ results reported in this paper.

K Detailed Perplexity Data (Sequence Length 2048)

Table 19: WikiText-2 perplexity (↓) at sequence length 2048 and tile size 1×64. Sparsity columns are grouped by hardware and index cost; the shaded group (2:4, OP-1D, OP-F2D) shares 2:4's selectors, 2-bit index, and datapath. *OP-F2D vs. Unstr. 50%* is OP-F2D's relative perplexity increase over the unstructured oracle, $(\mathrm{PPL}_{\text{OP-F2D}}-\mathrm{PPL}_{\text{Unstr.}})/\mathrm{PPL}_{\text{Unstr.}}$, and *Gap closed* is the fraction of the 2:4→unstructured gap OP-F2D recovers, $(\mathrm{PPL}_{2:4}-\mathrm{PPL}_{\text{OP-F2D}})/(\mathrm{PPL}_{2:4}-\mathrm{PPL}_{\text{Unstr.}})$. Best results in the 2:4-efficiency group in **bold**.

		Higher HW & index cost			At 2:4 HW & index cost			OP-F2D vs.	Gap closed
Model (Dense)	**Method**	**Unstr. 50%**	**8:16**	**4:8**	**2:4**	**OP-1D**	**OP-F2D**	**Unstr. 50%** (↓)	**2:4→Unstr.** (↑)
Llama 7B (5.68)	Wanda	7.26	7.84	8.56	11.55	8.52	**7.92**	+9.1%	84.6%
	RIA	7.12	7.63	8.32	11.15	8.26	**7.68**	+7.9%	86.1%
	Wanda++	6.96	7.42	7.92	9.61	7.83	**7.44**	+6.9%	81.9%
Llama 13B (5.09)	Wanda	6.15	6.75	7.41	9.62	7.22	**6.69**	+8.8%	84.4%
	RIA	6.06	6.57	7.15	8.92	7.00	**6.54**	+7.9%	83.2%
	Wanda++	6.00	6.43	6.84	7.97	6.79	**6.37**	+6.2%	81.2%
Llama 30B (4.10)	Wanda	5.26	5.61	5.98	6.91	5.89	**5.66**	+7.6%	75.8%
	RIA	5.09	5.44	5.78	6.74	5.74	**5.48**	+7.7%	76.4%
	Wanda++	5.08	5.37	5.70	6.46	5.62	**5.39**	+6.1%	77.5%
Llama 2 7B (5.47)	Wanda	6.92	7.62	8.60	12.12	8.37	**7.62**	+10.1%	86.5%
	RIA	6.80	7.46	8.40	11.30	8.15	**7.47**	+9.9%	85.1%
	Wanda++	6.71	7.21	7.82	9.55	7.67	**7.20**	+7.3%	82.7%
Llama 2 13B (4.88)	Wanda	5.98	6.44	7.01	9.00	6.97	**6.44**	+7.7%	84.8%
	RIA	5.83	6.24	6.75	8.41	6.67	**6.22**	+6.7%	84.9%
	Wanda++	5.79	6.12	6.54	7.57	6.46	**6.13**	+5.9%	80.9%
Llama 3.1 8B (6.24)	Wanda	9.66	11.39	13.77	22.53	13.36	**11.46**	+18.6%	86.0%
	RIA	9.45	11.07	13.55	22.61	12.95	**11.13**	+17.8%	87.2%
	Wanda++	8.83	9.81	10.95	14.11	10.70	**9.75**	+10.4%	82.6%
Qwen 2.5 7B (6.85)	Wanda	8.62	9.34	10.43	14.87	10.13	**9.32**	+8.1%	88.8%
	RIA	8.45	9.10	10.12	13.83	9.90	**9.11**	+7.8%	87.7%
	Wanda++	8.32	8.78	9.37	10.87	9.22	**8.81**	+5.9%	80.8%
Qwen 2.5 32B (5.02)	Wanda	6.29	6.64	7.06	8.08	6.95	**6.65**	+5.7%	79.9%
	RIA	6.26	6.55	6.93	7.86	6.84	**6.57**	+5.0%	80.6%
	Wanda++	6.23	6.57	6.93	7.74	6.84	**6.57**	+5.5%	77.5%
Gemma 2 9B (6.84)	Wanda	8.46	9.10	9.92	12.24	9.77	**9.11**	+7.7%	82.8%
	RIA	8.47	9.19	9.98	12.42	9.83	**9.18**	+8.4%	82.0%
	Wanda++	8.47	9.08	9.74	11.32	9.60	**9.05**	+6.8%	79.6%
OPT 6.7B (10.86)	Wanda	11.99	12.79	13.57	15.95	13.47	**12.84**	+7.1%	78.5%
	RIA	11.81	12.63	13.45	15.80	13.29	**12.56**	+6.4%	81.2%
	Wanda++	11.86	12.48	13.11	15.14	13.01	**12.49**	+5.3%	80.8%

Table 20: Aggregate behavior of OP-F2D across the ten models with tile size 1×64 of Table 19. Left: OP-F2D's relative perplexity increase over the unstructured 50% oracle. Right: the fraction of the 2:4→unstructured gap that OP-F2D recovers. The two columns must be read jointly: methods with stronger 2:4 baselines have a smaller gap available to close, so a lower *gap closed* does not imply worse absolute quality. Under every scoring method, OP-F2D's mean relative increase over the unstructured oracle stays in single digits while recovering roughly four-fifths of the 2:4 gap at 2:4 hardware cost.

	OP-F2D vs. Unstr. 50% (↓)			**Gap closed 2:4→Unstr.** (↑)		
Method	Mean	Median	Range	Mean	Median	Range
Wanda	+9.1%	+7.9%	5.7–18.6%	83.2%	84.5%	75.8–88.8%
RIA	+8.6%	+7.9%	5.0–17.8%	83.4%	84.1%	76.4–87.7%
Wanda++	**+6.6%**	**+6.2%**	5.3–10.4%	80.6%	80.9%	77.5–82.7%

Table 21: WikiText-2 perplexity (↓) at sequence length 2048 and tile size 1×16.

		Higher HW & index cost			**At 2:4 HW & index cost**			**OP-F2D vs. Unstr. 50%** (↓)	**Gap closed 2:4→Unstr.** (↑)
Model (Dense)	**Method**	**Unstr. 50%**	**8:16**	**4:8**	**2:4**	**OP-1D**	**OP-F2D**		
Llama 7B (5.68)	Wanda	7.26	7.84	8.56	11.55	8.80	**8.08**	+11.3%	80.9%
	RIA	7.12	7.63	8.32	11.15	8.49	**7.93**	+11.4%	79.9%
	Wanda++	6.96	7.42	7.92	9.61	8.01	**7.55**	+8.5%	77.7%
Llama 13B (5.09)	Wanda	6.15	6.75	7.41	9.62	7.49	**6.95**	+13.0%	76.9%
	RIA	6.06	6.57	7.15	8.92	7.19	**6.73**	+11.1%	76.6%
	Wanda++	6.00	6.43	6.84	7.97	6.93	**6.55**	+9.2%	72.1%
Llama 30B (4.10)	Wanda	5.26	5.61	5.98	6.91	6.02	**5.72**	+8.7%	72.1%
	RIA	5.09	5.44	5.78	6.74	5.85	**5.54**	+8.8%	72.7%
	Wanda++	5.08	5.37	5.70	6.46	5.72	**5.47**	+7.7%	71.7%
Llama 2 7B (5.47)	Wanda	6.92	7.62	8.60	12.12	8.75	**7.94**	+14.7%	80.4%
	RIA	6.80	7.46	8.40	11.30	8.49	**7.76**	+14.1%	78.7%
	Wanda++	6.71	7.21	7.82	9.55	7.87	**7.42**	+10.6%	75.0%
Llama 2 13B (4.88)	Wanda	5.98	6.44	7.01	9.00	7.13	**6.62**	+10.7%	78.8%
	RIA	5.83	6.24	6.75	8.41	6.82	**6.38**	+9.4%	78.7%
	Wanda++	5.79	6.12	6.54	7.57	6.58	**6.24**	+7.8%	74.7%
Llama 3.1 8B (6.24)	Wanda	9.66	11.39	13.77	22.53	14.13	**12.25**	+26.8%	79.9%
	RIA	9.45	11.07	13.55	22.61	13.89	**11.80**	+24.9%	82.1%
	Wanda++	8.83	9.81	10.95	14.11	11.07	**10.13**	+14.7%	75.4%
Qwen 2.5 7B (6.85)	Wanda	8.62	9.34	10.43	14.87	10.46	**9.66**	+12.1%	83.4%
	RIA	8.45	9.10	10.12	13.83	10.22	**9.39**	+11.1%	82.5%
	Wanda++	8.32	8.78	9.37	10.87	9.38	**8.96**	+7.7%	74.9%
Qwen 2.5 32B (5.02)	Wanda	6.29	6.64	7.06	8.08	7.07	**6.77**	+7.6%	73.2%
	RIA	6.26	6.55	6.93	7.86	6.97	**6.69**	+6.9%	73.1%
	Wanda++	6.23	6.57	6.93	7.74	6.95	**6.67**	+7.1%	70.9%
Gemma 2 9B (6.84)	Wanda	8.46	9.10	9.92	12.24	10.01	**9.37**	+10.8%	75.9%
	RIA	8.47	9.19	9.98	12.42	10.11	**9.45**	+11.6%	75.2%
	Wanda++	8.47	9.08	9.74	11.32	9.81	**9.30**	+9.8%	70.9%
OPT 6.7B (10.86)	Wanda	11.99	12.79	13.57	15.95	13.63	**13.12**	+9.4%	71.5%
	RIA	11.81	12.63	13.45	15.80	13.52	**12.98**	+9.9%	70.7%
	Wanda++	11.86	12.48	13.11	15.14	13.16	**12.75**	+7.5%	72.9%

Table 22: WikiText-2 perplexity (↓) at sequence length 2048 and tile size 1×256.

Model (Dense)	Method	Higher HW & index cost			At 2:4 HW & index cost			OP-F2D vs. Unstr. 50% (↓)	Gap closed 2:4→Unstr. (↑)
		Unstr. 50%	8:16	4:8	2:4	OP-1D	OP-F2D		
Llama 7B (5.68)	Wanda	7.26	7.84	8.56	11.55	8.42	**7.86**	+8.3%	86.0%
	RIA	7.12	7.63	8.32	11.15	8.20	**7.67**	+7.7%	86.4%
	Wanda++	6.96	7.42	7.92	9.61	7.78	**7.38**	+6.0%	84.2%
Llama 13B (5.09)	Wanda	6.15	6.75	7.41	9.62	7.16	**6.66**	+8.3%	85.3%
	RIA	6.06	6.57	7.15	8.92	6.96	**6.50**	+7.3%	84.6%
	Wanda++	6.00	6.43	6.84	7.97	6.73	**6.37**	+6.2%	81.2%
Llama 30B (4.10)	Wanda	5.26	5.61	5.98	6.91	5.87	**5.58**	+6.1%	80.6%
	RIA	5.09	5.44	5.78	6.74	5.70	**5.39**	+5.9%	81.8%
	Wanda++	5.08	5.37	5.70	6.46	5.60	**5.34**	+5.1%	81.2%
Llama 2 7B (5.47)	Wanda	6.92	7.62	8.60	12.12	8.32	**7.57**	+9.4%	87.5%
	RIA	6.80	7.46	8.40	11.30	8.09	**7.37**	+8.4%	87.3%
	Wanda++	6.71	7.21	7.82	9.55	7.63	**7.12**	+6.1%	85.6%
Llama 2 13B (4.88)	Wanda	5.98	6.44	7.01	9.00	6.90	**6.38**	+6.7%	86.8%
	RIA	5.83	6.24	6.75	8.41	6.62	**6.18**	+6.0%	86.4%
	Wanda++	5.79	6.12	6.54	7.57	6.44	**6.07**	+4.8%	84.3%
Llama 3.1 8B (6.24)	Wanda	9.66	11.39	13.77	22.53	13.24	**11.39**	+17.9%	86.6%
	RIA	9.45	11.07	13.55	22.61	12.85	**11.07**	+17.1%	87.7%
	Wanda++	8.83	9.81	10.95	14.11	10.64	**9.73**	+10.2%	83.0%
Qwen 2.5 7B (6.85)	Wanda	8.62	9.34	10.43	14.87	10.05	**9.24**	+7.2%	90.1%
	RIA	8.45	9.10	10.12	13.83	9.82	**9.01**	+6.6%	89.6%
	Wanda++	8.32	8.78	9.37	10.87	9.18	**8.74**	+5.0%	83.5%
Qwen 2.5 32B (5.02)	Wanda	6.29	6.64	7.06	8.08	6.92	**6.65**	+5.7%	79.9%
	RIA	6.26	6.55	6.93	7.86	6.82	**6.55**	+4.6%	81.9%
	Wanda++	6.23	6.57	6.93	7.74	6.81	**6.55**	+5.1%	78.8%
Gemma 2 9B (6.84)	Wanda	8.46	9.10	9.92	12.24	9.72	**9.07**	+7.2%	83.9%
	RIA	8.47	9.19	9.98	12.42	9.78	**9.15**	+8.0%	82.8%
	Wanda++	8.47	9.08	9.74	11.32	9.54	**9.05**	+6.8%	79.6%
OPT 6.7B (10.86)	Wanda	11.99	12.79	13.57	15.95	13.40	**12.70**	+5.9%	82.1%
	RIA	11.81	12.63	13.45	15.80	13.23	**12.56**	+6.4%	81.2%
	Wanda++	11.86	12.48	13.11	15.14	12.97	**12.38**	+4.4%	84.1%

L Detailed Zero-Shot Accuracy (Sequence Length 2048)

Table 23: Average zero-shot performance across eight evaluation tasks (Table 3) under **Wanda** scoring. Shaded rows are at 2:4 hardware efficiency; best of that group in **bold**.

Sparsity Pattern	Llama 7B	Llama 13B	Llama 2 7B	Llama 3.1 8B	Qwen 2.5 7B	Gemma 2 9B	OPT 6.7B
Dense	62.38	64.92	62.23	67.09	67.42	68.74	54.62
Unstr. 50%	57.95	62.26	58.47	59.36	63.55	64.83	52.14
8:16	57.68	60.13	57.07	56.64	63.09	63.32	51.37
4:8	55.62	58.19	55.42	54.14	61.91	62.52	50.63
2:4	51.75	55.14	51.41	49.71	57.79	57.97	48.98
OP-1D	55.69	59.21	56.36	54.72	61.97	62.01	50.80
OP-F2D	**58.13**	**60.27**	**57.43**	**56.67**	**62.53**	**62.75**	**51.35**

Table 24: Average zero-shot performance across eight evaluation tasks (Table 3) under **Wanda++** scoring. Shaded rows are at 2:4 hardware efficiency; best of that group in **bold**.

Sparsity Pattern	Llama 7B	Llama 13B	Llama 2 7B	Llama 3.1 8B	Qwen 2.5 7B	Gemma 2 9B	OPT 6.7B
Dense	62.38	64.92	62.23	67.09	67.42	68.74	54.62
Unstr. 50%	58.74	62.18	57.95	59.76	64.70	64.15	52.09
8:16	57.48	60.34	56.90	57.33	63.84	62.78	51.28
4:8	55.98	58.64	55.48	55.48	62.76	61.22	50.92
2:4	53.09	56.03	52.58	52.59	60.40	57.37	49.53
OP-1D	56.44	59.72	56.22	56.01	62.90	61.44	51.08
OP-F2D	**57.52**	**60.06**	**57.83**	**57.52**	**63.78**	**62.00**	**51.40**

Table 25: Llama 7B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Llama 7B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	42.15	75.34	77.06	56.97	33.60	78.67	66.06	69.22	62.38
Wanda	Unstr. 50%	37.80	70.54	71.90	51.99	30.40	75.84	57.40	67.72	57.95
	2:4	29.69	61.95	68.38	41.57	24.60	70.89	55.60	61.33	51.75
	4:8	33.19	66.75	70.61	46.83	28.40	74.65	58.84	65.67	55.62
	8:16	36.01	69.57	71.41	49.33	29.60	75.68	62.82	67.01	57.68
	OP-1D (64)	34.73	66.84	71.28	47.38	26.60	73.99	59.21	65.51	55.69
	OP-F2D (1×64)	37.63	69.40	72.26	49.64	28.40	75.63	65.70	66.38	58.13
Wanda++	Unstr. 50%	36.69	72.10	72.54	52.41	30.20	76.44	62.09	67.48	58.74
	2:4	29.44	64.35	67.28	44.68	23.00	72.36	59.57	64.01	53.09
	4:8	33.28	66.54	71.50	48.57	27.00	73.99	61.01	65.98	55.98
	8:16	35.92	69.61	71.71	50.18	28.60	75.30	61.73	66.77	57.48
	OP-1D (64)	34.98	67.59	73.21	49.09	26.40	73.50	61.01	65.75	56.44
	OP-F2D (1×64)	35.24	68.81	72.54	50.60	27.60	75.30	64.62	65.43	57.52

Table 26: Llama 13B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Llama 13B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	45.65	77.27	79.63	59.94	34.80	79.60	69.68	72.77	64.92
Wanda	Unstr. 50%	43.77	73.57	77.16	55.75	33.60	77.26	64.26	72.69	62.26
	2:4	33.79	66.75	71.28	46.44	28.20	74.10	53.43	67.09	55.14
	4:8	38.74	70.75	73.76	51.24	30.60	76.22	54.87	69.30	58.19
	8:16	41.13	72.90	77.52	53.43	31.00	76.82	58.12	70.09	60.13
	OP-1D (64)	40.53	71.68	74.98	51.87	29.80	76.01	58.48	70.32	59.21
	OP-F2D (1×64)	41.13	72.85	75.32	53.34	33.40	77.48	58.48	70.17	60.27
Wanda++	Unstr. 50%	43.09	73.74	78.41	56.06	33.60	78.07	62.82	71.67	62.18
	2:4	34.04	67.00	72.72	49.03	27.40	75.19	54.15	68.67	56.03
	4:8	39.08	71.25	74.04	52.71	30.20	76.77	54.87	70.17	58.64
	8:16	39.68	73.48	77.61	54.33	31.60	77.58	57.76	70.64	60.34
	OP-1D (64)	39.51	71.97	74.95	53.08	31.00	76.39	61.01	69.85	59.72
	OP-F2D (1×64)	39.93	72.77	75.38	54.32	32.60	77.15	57.40	70.96	60.06

Table 27: Llama 2 7B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Llama 2 7B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	42.41	75.46	79.51	57.12	33.20	77.80	63.18	69.14	62.23
Wanda	Unstr. 50%	39.42	72.69	75.87	52.51	30.40	76.61	53.07	67.17	58.47
	2:4	29.10	61.66	69.11	41.40	24.00	70.62	53.43	61.96	51.41
	4:8	34.56	67.21	73.06	46.78	27.00	74.21	55.23	65.27	55.42
	8:16	36.01	70.45	75.60	49.65	28.20	75.90	55.60	65.19	57.07
	OP-1D (64)	35.49	68.86	73.61	47.35	26.60	74.05	58.84	66.06	56.36
	OP-F2D (1×64)	37.29	69.02	74.62	49.87	30.20	74.86	57.76	65.82	57.43
Wanda++	Unstr. 50%	37.46	72.56	75.60	52.32	30.00	76.22	54.15	65.27	57.95
	2:4	30.55	62.58	69.05	44.08	24.20	71.44	54.87	63.85	52.58
	4:8	33.53	67.89	71.19	48.25	27.40	74.65	54.51	66.46	55.48
	8:16	34.64	69.40	73.98	50.22	29.00	75.73	56.32	65.90	56.90
	OP-1D (64)	34.81	68.14	73.03	48.76	27.20	74.21	58.84	64.80	56.22
	OP-F2D (1×64)	36.01	69.49	73.21	50.07	31.00	75.57	60.65	66.61	57.83

Table 28: Llama 3.1 8B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Llama 3.1 8B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	52.22	82.20	83.06	60.67	34.00	78.94	71.48	74.19	67.09
Wanda	Unstr. 50%	42.58	73.78	79.76	51.75	26.60	75.46	56.32	68.67	59.36
	2:4	27.65	60.69	66.06	38.15	19.40	70.40	54.51	60.85	49.71
	4:8	34.56	67.68	68.99	45.16	24.40	72.80	53.07	66.46	54.14
	8:16	37.88	70.12	72.66	48.75	25.80	74.54	54.15	69.22	56.64
	OP-1D (64)	34.13	68.14	70.15	45.92	24.40	72.42	55.60	67.01	54.72
	OP-F2D (1×64)	36.95	71.59	74.07	48.05	26.60	74.65	54.51	66.93	56.67
Wanda++	Unstr. 50%	41.38	74.07	79.27	52.47	28.00	76.55	57.76	68.59	59.76
	2:4	30.46	64.60	66.33	43.29	22.80	72.36	58.12	62.75	52.59
	4:8	34.81	69.53	71.22	47.71	26.40	73.56	53.79	66.85	55.48
	8:16	38.05	70.75	75.38	50.09	26.60	75.14	54.87	67.72	57.33
	OP-1D (64)	35.67	70.88	73.18	48.19	24.40	73.34	56.68	65.75	56.01
	OP-F2D (1×64)	37.20	72.26	75.99	50.18	25.40	74.97	56.32	67.80	57.52

Table 29: Qwen 2.5 7B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Qwen 2.5 7B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	48.04	80.47	84.71	60.00	33.80	78.73	80.87	72.77	67.42
Wanda	Unstr. 50%	41.21	75.93	82.97	53.04	31.00	77.15	76.90	70.24	63.55
	2:4	36.43	71.59	74.37	44.43	26.00	71.87	73.29	64.33	57.79
	4:8	42.06	75.46	81.10	49.50	27.00	74.81	76.53	68.82	61.91
	8:16	41.89	75.38	83.52	51.65	28.00	74.97	79.78	69.53	63.09
	OP-1D (64)	41.04	73.65	80.46	49.71	28.60	75.03	77.98	69.30	61.97
	OP-F2D (1×64)	41.21	75.51	79.82	51.69	27.20	76.55	77.98	70.32	62.53
Wanda++	Unstr. 50%	43.34	77.23	84.28	54.08	31.60	78.18	79.06	69.85	64.70
	2:4	39.16	73.40	80.83	47.45	27.00	74.05	75.45	65.82	60.40
	4:8	42.83	76.56	82.42	50.77	27.20	76.55	75.45	70.32	62.76
	8:16	43.26	76.30	84.01	52.27	30.40	76.28	78.34	69.85	63.84
	OP-1D (64)	42.06	74.45	83.03	51.11	29.20	76.66	76.90	69.77	62.90
	OP-F2D (1×64)	44.45	77.10	82.87	52.97	26.80	77.53	77.62	70.88	63.78

Table 30: Gemma 2 9B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (Gemma 2 9B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
	Dense	61.77	87.12	84.25	61.22	33.60	81.28	66.79	73.88	68.74
Wanda	Unstr. 50%	50.77	81.10	82.35	54.31	31.80	78.29	67.15	72.85	64.83
	2:4	40.53	74.16	75.63	46.33	26.40	74.92	58.84	66.93	57.97
	4:8	45.39	79.76	80.28	50.62	27.60	77.20	68.95	70.32	62.52
	8:16	47.70	80.39	81.65	52.64	30.80	77.48	64.26	71.67	63.32
	OP-1D (64)	46.67	78.41	80.70	51.09	28.20	76.50	63.18	71.35	62.01
	OP-F2D (1×64)	47.18	78.49	81.04	52.53	31.20	77.04	62.45	72.06	62.75
Wanda++	Unstr. 50%	50.94	81.82	78.53	54.41	33.20	79.05	63.18	72.06	64.15
	2:4	39.59	73.57	75.20	47.15	27.00	75.19	55.96	65.27	57.37
	4:8	45.48	79.00	75.38	50.32	29.20	77.20	64.62	68.59	61.22
	8:16	46.76	80.39	79.11	52.44	30.20	77.91	64.62	70.80	62.78
	OP-1D (64)	45.82	78.96	77.03	51.13	29.20	77.09	62.82	69.46	61.44
	OP-F2D (1×64)	46.59	79.00	77.68	52.46	31.60	77.53	61.01	70.17	62.00

Table 31: OPT 6.7B: per-task zero-shot accuracy (%), sequence length 2048.

Methods	Sparsity Type	Zero-shot Evaluation (OPT 6.7B)								
		ARC-C	ARC-E	BoolQ	HS	OBQA	PIQA	RTE	WG	Average
Dense		30.03	65.99	66.97	50.28	26.40	76.39	55.96	64.96	54.62
Wanda	Unstr. 50%	28.33	63.43	64.92	46.83	24.60	74.43	52.71	61.88	52.14
	2:4	24.74	57.24	62.23	41.42	23.00	70.95	53.07	59.19	48.98
	4:8	27.13	59.64	63.21	44.50	24.60	72.69	52.71	60.54	50.63
	8:16	26.62	61.87	64.59	45.86	24.60	73.39	52.35	61.72	51.37
	OP-1D (64)	26.19	60.56	63.39	45.04	24.60	73.01	52.35	61.25	50.80
	OP-F2D (1×64)	27.30	61.66	63.33	45.86	24.00	73.56	52.35	62.75	51.35
Wanda++	Unstr. 50%	27.65	63.85	63.36	46.69	25.40	74.54	52.71	62.51	52.09
	2:4	25.60	58.33	62.32	41.91	23.60	71.82	52.35	60.30	49.53
	4:8	26.71	61.78	63.18	44.60	24.80	72.91	52.71	60.69	50.92
	8:16	26.79	62.08	62.94	45.84	24.60	73.61	52.71	61.64	51.28
	OP-1D (64)	26.62	60.69	62.78	44.85	26.20	73.72	52.71	61.09	51.08
	OP-F2D (1×64)	27.05	62.33	63.06	45.72	25.00	74.27	52.35	61.40	51.40

M Compatibility with Quantization

We investigate our method's compatibility with quantization of Llama 3.1 8B with FP8 (E4M3) (Micikevicius et al., 2022) and NVFP4 (Alvarez et al., 2025), under both weight-only and dynamic weight-and-activation settings. Following Harma et al. (2025), models are pruned before quantizing. Quantization code is adapted from the LLM Compressor library (Red Hat AI & vLLM Project, 2024). All sparse rows use Wanda++ scoring. Table 32 reports perplexity (↓) as precision is reduced from BF16 to FP8 and NVFP4. The "16 → 4" columns report the relative degradation from BF16 to NVFP4 within each quantization setting.

Several trends are evident. First, FP8 is near-lossless for every sparsity pattern. Weight-only NVFP4 results in a slight perplexity increase (6–9%), while quantizing activations roughly doubles the BF16-to-NVFP4 perplexity increase. This is precisely where model quality further breaks down for semi-structured 2:4 (+18.9%) relative to OP-F2D (+14.7%). Second, each OP sparsity remains effective under quantization: the perplexity ordering 2:4 > OP-1D > OP-F2D is preserved across all precision levels. Most importantly, our overlapping patterns do not compound quantization error. Measured as per-column overhead relative to the Unstr. 50% baseline, naive 2:4 costs $+59.8\%$ to $+67.4\%$, whereas OP-F2D stays flat at 10.4–11.6% across BF16, FP8, and NVFP4.

Table 32: Llama 3.1 8B perplexity (↓) under weight-only and weight-and-activation quantization from BF16 to FP8 and NVFP4; sparse rows use Wanda++ scoring. Shaded rows are at 2:4 HW efficiency.

		Weight Only			Weight and Activation		
Sparsity	**BF16**	**FP8**	**NVFP4**	**16 → 4**	**FP8**	**NVFP4**	**16 → 4**
Dense	6.24	6.27	6.63	+6.3%	6.30	6.99	+12.0%
Unstr. 50%	8.83	8.88	9.43	+6.8%	8.92	10.02	+13.5%
8:16	9.81	9.90	10.50	+7.0%	9.96	11.22	+14.4%
4:8	10.95	11.03	11.71	+6.9%	11.10	12.60	+15.1%
2:4	14.11	14.25	15.37	+8.9%	14.37	16.77	+18.9%
OP-1D	10.70	10.79	11.59	+8.3%	10.87	12.46	+16.4%
OP-F2D	**9.75**	**9.80**	**10.43**	**+7.0%**	**9.86**	**11.18**	**+14.7%**
2:4 vs. Unstr.	+59.8%	+60.5%	+63.0%	-	+61.1%	+67.4%	-
OP-F2D vs. Unstr.	+10.4%	+10.4%	+10.6%	-	+10.5%	+11.6%	-

N OP-Aware Channel Permutation Constructions

As noted in Section 4.2, channel permutation (CP) changes which columns compete within a pruning tile, whereas OP changes which positions each selector can observe. Their interaction therefore depends not only on which channels share a tile but also on the slots to which those channels are assigned. Both allocations in Table 33 begin with the same global channel ranking. Let $C = BT$ be the number of columns, where T is the allocation-tile width and B is the number of tiles, and let

Table 33: Llama 3.1 8B WikiText-2 perplexity (↓) at 50% sparsity for Wanda and RIA baselines and for RIA combined with HCA or dyadic Midpoint channel allocation. $\Delta_{\text{rel}} = (\text{PPL}_{\text{RIA}} - \text{PPL}_{\text{Midpoint}})/\text{PPL}_{\text{RIA}}$; positive values indicate improvement. Sequence length is 2048.

Sparsity Type	Baseline		RIA + Channel Permutation		
	Wanda	RIA	HCA	Midpoint	Δ_{rel}
Unstr. 50%	9.66	9.45	-	-	-
2:4	22.53	22.61	20.84	20.13	10.97%
4:8	13.77	13.55	12.99	13.28	1.99%
8:16	11.39	11.07	10.84	11.04	0.27%
OP-1D (1×64)	13.36	12.95	18.45	13.05	-0.77%
OP-F2D (1×64)	11.46	11.13	11.62	10.89	2.16%
OP-2D (8×8)	11.21	11.31	11.15	10.85	4.07%

$\rho_0, \ldots, \rho_{C-1}$ list the channel indices from least to most important. The saliency matrix is reordered by the resulting index, pruned in the permuted coordinates, and the mask is then mapped back to the original tensor indices.

HCA. The heuristic channel allocation used by RIA (Zhang et al., 2024) divides the ranked list into T contiguous importance bands of B channels each. Band j is assigned to slot j of every tile, with alternating bands reversed:

$$\pi_{kT+j} = \begin{cases} \rho_{jB+k}, & j \text{ even}, \\ \rho_{(j+1)B-1-k}, & j \text{ odd}, \end{cases} \qquad 0 \le j < T, \quad 0 \le k < B. \tag{12}$$

Thus every tile receives one channel from each importance band, while the alternating direction forms a serpentine ordering that balances ranks across tiles. This is effective for disjoint $N{:}M$ blocks, where every position in a block participates in the same top-N decision. For OP, however, the strongest bands occupy adjacent slots at one end of the tile. Several high-importance columns can consequently enter the same overlapping selector windows and contend for retention; the OP-1D HCA result in Table 33 exhibits this failure mode.

Dyadic Midpoint allocation. Midpoint uses the same ranking but changes the order in which tile slots receive importance bands. For a power-of-two tile width T, it visits slot 0 first and then recursively fills the midpoints of the remaining intervals:

$$(p_0, p_1, \ldots, p_{T-1}) = (0,\ T/2,\ T/4,\ 3T/4,\ T/8,\ 3T/8,\ 5T/8,\ 7T/8, \ldots). \tag{13}$$

Equivalently, the implementation orders nonzero slot indices by decreasing lowest set bit, treating slot 0 as having scale T. It then assigns channels from most to least important by

$$\pi_{kT+p_j} = \rho_{C-1-(jB+k)}, \qquad 0 \le j < T, \quad 0 \le k < B. \tag{14}$$

For example, $T = 8$ gives the slot order $(0, 4, 2, 6, 1, 3, 5, 7)$. Hence the top half of channels occupy the even slots before any odd slot is filled: the highest-ranked bands are placed at the coarsest spacing, and lower-ranked bands progressively fill the gaps. Unlike HCA's contiguous placement of its strongest bands, this dyadic ordering spreads high-importance columns across each tile before filling the gaps, reducing selector contention.

Application to OP patterns. For OP-1D, Midpoint applies directly to the linear tile in Figure 1a; at 50% sparsity, its even-slot placement admits a collision-free assignment of the top half of channels to distinct selectors. For OP-F2D, the same allocation is applied to the row-major flattened tile of Figure 1d. For OP-2D, row and column indices are ranked and allocated independently using the corresponding tile dimensions before mask construction. The residue-class construction in Equations 12 and 14 applies across all tiles, so no pattern-specific lookup table is required as the tile size grows.

The results in Table 33 show why slot placement matters. HCA improves the conventional $N{:}M$ baselines but sharply degrades OP-1D and slightly degrades OP-F2D because its strongest bands

remain clustered. Midpoint removes the large OP-1D failure and improves OP-F2D by 2.16% relative to the RIA baseline. Channel permutation is therefore complementary to OP-F2D and OP-2D, while leaving OP-1D unchanged.