PERMUTATION-PACKABLE N:M SPARSITY FOR SPARSE-TO-SPARSE MUON

Anonymous authors
Paper under double-blind review

ABSTRACT

Applying Muon to sparse vision Transformers creates a mismatch: its Newton–Schulz (NS) iterations can generate updates at pruned weights, which projection then discards, while dense training tensors can limit the memory benefits of sparsity. We introduce *Permutation-Packable N:M* (PP-N:M), a locally balanced support that is N:M in both directions and packs exactly into dense active matrices. Its closure under NS cubic products enables matrix updates entirely on active coordinates. On this support, *S2S Muon* constructs update directions through pre-NS row–column equilibration and component-wise NS, with a second-moment-based Adam reference controlling their magnitude. To retain compact storage beyond the optimizer, we develop *PP-linear* kernels that fuse channel indexing with matrix products and compute packed weight gradients directly. In a seven-method ImageNet-1K comparison at 300 epochs over two seeds, ViT with S2S Muon reaches 78.21% mean top-1 accuracy versus 77.31% for projected Muon, a gain of 0.90 percentage points, while retaining half of the eligible weights. Complementary CIFAR-100 experiments reach $77.06 \pm 0.14\%$ (mean $\pm$ standard error) across three seeds, 3.76 percentage points above projected Muon. As a cross-domain test, 350M and 1B language models improve perplexity over projected Muon by 2.59% and 2.31%, respectively, at a fixed 1.008B-token budget. On RTX 5090, PP-linear reduces peak allocated training memory for single-momentum Muon+PP-N:M by 18.29% on ViT and 40.22% on a 1B language model relative to projected Muon. With the complete S2S Muon update rule held fixed, it saves 15.17% and 29.09%, respectively, relative to a dense-masked implementation. These memory benefits come with workload- and implementation-dependent throughput trade-offs.

1 INTRODUCTION

Vision Transformers (ViTs) process image-patch embeddings with self-attention and feed-forward layers, achieving strong recognition performance (Dosovitskiy, 2020; Touvron et al., 2021). Their repeated matrix projections make training expensive. Unstructured sparsification, as explored by SViTE, reduces the parameter budget (Chen et al., 2021), but irregular connectivity does not automatically translate into faster GPU execution (Mishra et al., 2021).

Semi-structured N:M sparsity retains N weights per group of M entries; 2:4 has dedicated NVIDIA Sparse Tensor Core support (Mishra et al., 2021). Transposable masks extend legality to forward and backward orientations (Hubara et al., 2021). In ViTs, Cai et al. (2024) report $1.3\times$ faster MLP linear operations and 6% shorter DINOv2 training using a sparse-then-dense schedule. Accuracy under restricted support remains a challenge (Hu et al., 2024a;b). Training-memory savings are possible: SLoPe combines N:M storage with a double-pruned backward pass (Mozaffari et al., 2025). Yet hardware legality alone does not make a support compatible with a matrix optimizer.

Muon is a compelling optimizer for this setting. Recent ViT studies report benefits across classification, segmentation, and masked autoencoding (Southworth et al., 2026); language-model studies also show strong training efficiency (Jordan et al., 2024; Liu et al., 2025). Muon approximately orthogonalizes matrix momentum with Newton–Schulz (NS) iterations instead of independently rescaling each coordinate. Its dense-training success, however, does not ensure compatibility with sparse weights.

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.

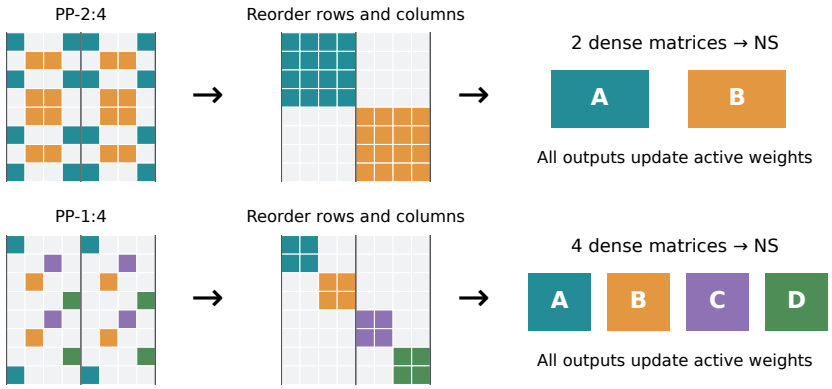

Figure 1: **Permutation changes storage order, not which weights are retained.** Color identifies a component; gray cells are absent weights. Vertical boundaries mark consecutive groups of four in the original layer. After grouping rows and columns by color, PP-2:4 produces two dense matrices and PP-1:4 produces four. NS operates on the colored matrices only, and all of its outputs return to active positions.

Two obstacles arise. NS matrix products can fill inactive positions, requiring projection that discards part of the computed update. Moreover, dense-latent sparse training can retain physical-size weights, gradients, and states despite a sparse forward mask (Cai et al., 2024; Hu et al., 2024b). The central question is therefore which supports jointly satisfy local N:M constraints, remain closed under matrix updates, and admit compact storage throughout training. We introduce *Permutation-Packable N:M* (PP-N:M): a locally balanced, bidirectionally N:M support that packs into $K = M/N$ dense active matrices (Figure 1). Closure under the NS cubic products keeps the transform on active coordinates, allowing eligible weights, gradients, and parameter-shaped states to remain compact.

Building on PP-N:M, *Sparse-to-sparse Muon* (S2S Muon) combines two moment statistics, row–column equilibration (PRE), component-wise NS, and Adam-reference magnitude control into an update computed entirely on active coordinates. Realizing this compact update as a memory-efficient training system also requires the linear operator to preserve the layout: reconstructing a dense weight in the forward pass or a dense weight gradient in backward would reintroduce physical-size tensors. We therefore develop *PP-linear*.

PP-linear's kernels fuse channel indexing with matrix products and write weight gradients directly in packed form. Weights, gradients, and optimizer states thus share one compact representation throughout training. We evaluate this execution path for both complete S2S and its single-momentum control, Muon+PP. On RTX 5090, Muon+PP saves 18.29% and 40.22% peak training memory versus projected Muon on a ViT and a 1B language model, at $0.85\times$ and $0.96\times$ throughput (Table 4). Holding the complete S2S update rule fixed, PP-linear saves 15.17% and 29.09% versus its dense-masked implementation, at $1.27\times$ and $0.82\times$ throughput, respectively.

- **PP-N:M: a sparse format compatible with matrix updates.** We introduce a support family that combines bidirectional local N:M balance with exact packing into dense active components. Its closure under NS cubic products resolves the mismatch between sparse weights and matrix-valued updates, allowing Muon to operate without off-support expansion and projection. To our knowledge, PP-N:M is the first N:M support format jointly enabling these structural properties and compact active-only Muon training.
- **S2S Muon: an optimizer built on PP-N:M.** We design an active-only matrix optimizer that separates direction construction from magnitude control: PRE and component-wise NS transform the first moment, while a second-moment-based Adam reference sets the update amplitude. The complete design improves ViT accuracy over projected Muon by 0.90 points on ImageNet-1K (300 epochs, two seeds) and 3.76 points on CIFAR-100 (three seeds), with complementary language-model gains. Muon+PP and paired ablations examine the contributions of the additional optimizer components.

- **PP-linear kernels with measured training-memory benefits.** We implement forward and backward kernels that fuse channel indexing with matrix products and generate packed weight gradients directly, preserving compact storage through the entire training step. On RTX 5090, Muon+PP reduces peak allocated training memory by 18.29% for a ViT and 40.22% for a 1B language model versus projected Muon. With the complete S2S update rule held fixed, PP-linear saves 15.17% and 29.09% on the same two model scales versus its dense-masked implementation. Throughput is reported alongside memory to characterize the execution trade-off.

2 SPARSE-TO-SPARSE MATRIX OPTIMIZATION

Sparse computation versus sparse-to-sparse updates. Sparse-from-scratch methods such as SET (Mocanu et al., 2018) and sparse-backward methods such as Top-KAST (Jayakumar et al., 2020) establish the broader context of sparse training. A sparse forward weight, however, does not imply a sparse-to-sparse update: some methods retain dense latent parameters or require gradients at inactive coordinates, as in SR-STE (Zhou et al., 2021). The relevant distinction is whether the update can be computed from sparse parameters, active gradients, and compact state, rather than only whether the forward mask is sparse.

Sparse-to-sparse matrix optimization. We formalize this requirement for matrix optimizers in an *active-only, fixed-support* setting. For an eligible matrix with support Ω, weights, gradients, and parameter-shaped optimizer states are stored only at active coordinates. An optimizer step uses these quantities to produce an update on the same support, without requiring inactive-coordinate gradients or materializing a physical-size parameter or update. For a projection-free matrix transform $\mathcal{T}$, the additional structural requirement is

$$\operatorname{supp}(U) \subseteq \Omega \quad \Longrightarrow \quad \operatorname{supp}(\mathcal{T}(U)) \subseteq \Omega, \tag{1}$$

for every supported input U, together with an implementation that evaluates the transform directly in compact coordinates.

This requirement exposes the distinction from elementwise optimization. Active-state sparse Adam can update gathered coordinates (Xiao et al., 2026), whereas Muon's NS iterations couple entries through matrix products such as $UU^\top U$. Sparse inputs and compact moments alone do not keep these products on Ω. Computing a dense direction and masking it afterward restores sparse output, but not an active-only matrix-update path. Our extension of sparse-to-sparse optimization is therefore to make the internal matrix transform, not just its final output, compatible with the sparse support. PP-N:M realizes this requirement through NS-closed, packable components. The requirement concerns eligible weights and their updates; activations, attention, dense auxiliary parameters, and Gram workspaces may remain dense.

3 PP-N:M SPARSITY AND S2S MUON

3.1 A PATTERN THAT CAN BE PACKED BY PERMUTATION

Our construction separates the *support family*, its exact *packed representation*, and the *update rule* applied to its components. The first restricts connectivity, the second preserves the chosen sparse linear map, and the third determines the optimizer.

We consider $N \mid M$, set $K = M/N$, and assume m, n are multiples of M. Partition row indices into $R_1, \ldots, R_K$ and column indices into $C_1, \ldots, C_K$. In each consecutive group of M row indices, assign exactly N to every R_k; impose the analogous rule on column indices. Define

$$\Omega_{ij} = \mathbf{1}\{i \in R_k,\ j \in C_k \text{ for some } k\}, \qquad W_k = W[R_k, C_k] \in \mathbb{R}^{m/K \times n/K}. \tag{2}$$

For any row in R_k, each consecutive $1 \times M$ group retains its N columns in C_k. The same property holds for $W^\top$. The mask is therefore N:M in both directions. The row and column balance conditions also give $|R_k| = m/K$ and $|C_k| = n/K$.

Let P_r, P_c be the permutations that group these indices. Then

$$W' = P_r W P_c^\top = \operatorname{diag}(W_1, \ldots, W_K). \tag{3}$$

Here "block diagonal" simply means that the dense active matrices sit along the diagonal, with zeros between them. The original mask need not contain contiguous dense blocks: they appear *after permutation*. This is a structured subset of N:M masks, not a packing guarantee for every legal N:M mask. It includes the 2:4, 1:4, 2:8, and 4:8 patterns studied here. At the level of a single layer, Equation (3) is a permuted grouped linear map. PP-N:M adds local balance in the original coordinates and an explicit connection to matrix-update compatibility; it does not require the layer to be represented as a product of structured factors.

Initialization. Starting from a dense initialization W^{d}, a balanced assignment heuristic favors retained energy $\sum_k \sum_{i\in R_k, j\in C_k} (W^{\mathrm{d}}_{ij})^2$. This selects a member of the support family rather than defining the family itself. Appendix B specifies the finite candidate assignments and tie-breaking; no global optimum is assumed. We then rescale the retained values to preserve the original matrix RMS:

$$W^{(0)} = \alpha(\Omega \odot W^{\mathrm{d}}), \qquad \alpha = \frac{\|W^{\mathrm{d}}\|_F}{\|\Omega \odot W^{\mathrm{d}}\|_F}, \qquad \mathrm{RMS}(W^{(0)}) = \mathrm{RMS}(W^{\mathrm{d}}). \tag{4}$$

We store only the blocks of $W^{(0)}$ during training. The assignments are fixed; permutations are indexing metadata rather than trainable parameters.

3.2 Closure of matrix updates on the active support

For a block-diagonal input $U' = \mathrm{diag}(U_1, \ldots, U_K)$,

$$U'U'^{\top}U' = \mathrm{diag}(U_1U_1^{\top}U_1, \ldots, U_KU_K^{\top}U_K). \tag{5}$$

Consequently, the matrix products in NS never create values between these blocks. Computing each block separately yields outputs with exactly the shapes needed to update the active weights. No N-of-M projection follows the NS transform.

The block structure is more than a storage convenience. If we require $UU^{\top}U$ to stay inside a mask for *every* choice of its active values, each connected row–column component must be fully populated. Combined with bidirectional N:M counts, this gives precisely the balanced blocks above (Proposition 2). PP-N:M makes that structure explicit instead of relying on cancellation at particular weights. The underlying support-graph fact is classical (Block et al., 2020); the characterization here connects it to matrix updates and local N:M counts. Avoiding off-support output removes discarded computation, but is not by itself a guarantee of a better optimization direction.

This identity also explains the relationship to a dense implementation. If all blocks use the same normalization and polynomial coefficients, evaluating NS on the zero-filled block-diagonal matrix is algebraically equivalent to evaluating it block by block. S2S Muon additionally normalizes each block separately and preconditions its entries before NS. These are choices of update rule, not consequences of merely changing storage order. Appendix A gives the precise equivalence and its relationship to an exact polar transform.

3.3 The S2S Muon update

For component k, let $g_{t,k} = \partial \mathcal{L}_t / \partial W_{t,k}$ be the active gradient at update t. We maintain first and second moments only at active coordinates:

$$\begin{aligned} U_{t,k} &= \beta_1 U_{t-1,k} + (1-\beta_1) g_{t,k}, & V_{t,k} &= \beta_2 V_{t-1,k} + (1-\beta_2) g_{t,k}^{\odot 2}, \\ \widehat{U}_{t,k} &= U_{t,k}/(1-\beta_1^t), & A_{t,k} &= \frac{\widehat{U}_{t,k}}{\sqrt{V_{t,k}/(1-\beta_2^t)} + \epsilon}. \end{aligned} \tag{6}$$

$A_{t,k}$ is an Adam-style reference used *only to set the update magnitude*. The NS input is the first moment $\widehat{U}_{t,k}$, not $A_{t,k}$.

Preconditioning before NS. For an input matrix Z, PRE first divides each entry by the RMS of its row, then by the RMS of its column in the resulting matrix:

$$\widetilde{Z}_{ij} = \frac{Z_{ij}}{\max(\sqrt{\frac{1}{q}\sum_{j'} Z_{ij'}^2}, \epsilon_p)}, \qquad \mathrm{PRE}(Z)_{ij} = \frac{\widetilde{Z}_{ij}}{\max(\sqrt{\frac{1}{p}\sum_{i'} \widetilde{Z}_{i'j}^2}, \epsilon_p)}, \quad Z \in \mathbb{R}^{p\times q}. \tag{7}$$

The divisions are elementwise, but their scales are shared across a row or column. This balances large and small row/column magnitudes before the finite NS iteration. It does not introduce another per-weight running statistic. This sequential equilibration is related to pre-orthogonalization balancing in MuonEq (Chang et al., 2026). S2S combines it with active-only component updates and reference-magnitude control. PRE changes the direction supplied to NS; it is distinct from packing and from a scalar normalization.

Direction and magnitude. We apply five NS iterations independently to each prepared component (Jordan et al., 2024; Liu et al., 2025). Writing RMS_Ω for RMS over *all* active entries of a layer, the complete update is

$$Q_{t,k} = \mathrm{NS}_5\Big(\mathrm{PRE}(\widehat{U}_{t,k})\Big),$$
$$D_{t,k} = \frac{K\ \mathrm{RMS}_\Omega(A_t)}{\max(\mathrm{RMS}_\Omega(Q_t), \epsilon_s)} Q_{t,k}, \qquad W_{t+1,k} = (1-\eta_t\lambda)W_{t,k} - \eta_t D_{t,k}. \tag{8}$$

Thus $Q_{t,k}$ is the NS output, $D_{t,k}$ is its rescaled version, and λ is decoupled weight decay. The factor K specifies the density-dependent scale convention used in our experiments. NS coefficients and numerical floors are given in Appendix B.

One active-only S2S Muon step (eligible matrix).
1. Receive packed gradients $g_{t,k}$ from PP-linear; apply the clipping convention.
2. Update $U_{t,k}, V_{t,k}$ and form $\widehat{U}_{t,k}, A_{t,k}$ by Equation (6).
3. Apply PRE, component Frobenius normalization, and NS5 to $\widehat{U}_{t,k}$.
4. Scale the directions using the layer-wide active RMS of A_t and Q_t.
5. Apply decoupled weight decay and update only packed $W_{t,k}$.

The second moment sets the reference magnitude; it is not the NS input.

Component normalization controls each NS input's scale locally; closure does not require this choice. Support-matched PRE/normalization controls (Appendix D.4) evaluate these update choices separately from the representation, while Muon+PP tests a single-momentum alternative.

3.4 PP-LINEAR: RETAINING THE LAYOUT THROUGH TRAINING

PP-linear preserves the packed parameter layout through forward, backward, and optimizer updates, without reconstructing physical-size weights or weight gradients. Activations retain their original channel order, allowing adjacent Transformer layers to use different PP partitions.

Storage layout and operator contract. We store the active weights contiguously as $\mathcal{W} \in \mathbb{R}^{K\times(m/K)\times(n/K)}$, together with the row and column lists R_k, C_k. These lists partition the two axes and contain $m+n$ indices in total; the execution path needs neither a physical-size Boolean mask nor stored zeros between components. For a batch $X \in \mathbb{R}^{b\times n}$, where b also folds in token or spatial positions, let $X' = XP_c^\top = [X_1, \dots, X_K]$ and $Y' = YP_r^\top$. Then

$$Y' = [X_1W_1^\top, \dots, X_KW_K^\top], \quad \nabla_{X_k}\mathcal{L} = \nabla_{Y'_k}\mathcal{L}\, W_k, \quad \nabla_{W_k}\mathcal{L} = (\nabla_{Y'_k}\mathcal{L})^\top X_k. \tag{9}$$

The permutations specify addresses, not additional dense matrices to multiply. The measured RTX 5090 implementation specializes this contract to PP-2:4 ($K=2$) with three tiled Triton kernels.

Forward: gather at load, scatter at store. A kernel program owns one component and one tile of output tokens and channels. It reads input columns through C_k and a contiguous tile of W_k, performs a dense tile multiplication, and writes the result directly to output columns R_k. Thus channel gathering is fused into operand loads and the inverse permutation into output stores: no full permuted activation buffer is materialized by this path. Because the R_k sets are disjoint, components write disjoint output channels without a cross-component reduction. BF16 operands use FP32 accumulation; FP32 master weights are converted to the compute dtype inside the kernel rather than expanded into a physical-size low-precision weight.

Backward: compute only the trainable weight gradient. The input-gradient kernel gathers output-gradient columns through R_k, multiplies by W_k, and stores directly to input columns C_k. Separately, each weight-gradient program owns a tile of W_k, reduces $(\nabla_{Y'_k}\mathcal{L})^\top X_k$ over the batch/token axis, and writes an FP32 tile directly into the packed gradient. It never computes an $m \times n$ gradient and then masks or gathers it. This is the exact gradient of the fixed-support parameterization, rather than an estimate of missing inactive entries. Each output tile has a single owner, so the kernel does not require atomic accumulation across components or a global split-reduction buffer.

Autograd saves the compute-dtype input and references to packed weights and indices, without retaining a separate permuted activation. The returned gradient directly matches the optimizer layout, eliminating per-step unpacking and repacking.

Memory accounting. There are mn/K active values in each parameter-shaped tensor. With FP32 weights, gradients, and s moment tensors, their combined storage when gradients are live is $4(2+s)mn/K$ bytes. The current module also allocates a compact BF16 weight cache and uses 32-bit indices, giving

$$B_{\text{eligible}} = \underbrace{4(2+s)mn/K}_{\text{weights, gradients, moments}} + \underbrace{2mn/K}_{\text{compact cache}} + \underbrace{4(m+n)}_{\text{index lists}}. \tag{10}$$

Here $s = 1$ for Muon+PP and $s = 2$ for S2S; the allocated BF16 cache is included in the measurements. For PP-2:4, every parameter-shaped term is half its physical-size counterpart. Index storage scales with the sum of the layer dimensions, rather than the number of weight entries. This ledger excludes transient NS/PRE workspaces, activations, and dense auxiliary parameters.

Arithmetic and whole-model behavior. Each eligible forward, input-gradient, and weight-gradient product uses $1/K$ of the corresponding dense arithmetic. For a square $d \times d$ weight, component NS cubic products cost $K\,O((d/K)^3) = O(d^3/K^2)$. These arithmetic reductions do not directly determine latency or peak memory, which also depend on indexed access, kernel utilization, activations, and temporary workspaces. Saved activations, attention, and noneligible layers do not shrink with the packed weight count. Section 4.5 reports whole-step measurements; Appendix B details the implementation.

4 Experiments

We evaluate sparse vision training at 50% and 75% sparsity, separate support construction from optimizer design, and test cross-domain transfer to language modeling. Multi-seed results report mean $\pm$ standard error (SE), computed as sample standard deviation divided by $\sqrt{n}$; seed counts accompany each table.

4.1 Primary comparison on CIFAR-100

We train for 600 epochs (52,800 updates) and evaluate the final checkpoint on the official test set. Table 1 compares both sparsities; details are in Appendix C.

Table 1 exhibits a *sparse advantage*: S2S exceeds dense Muon by 0.91 points at 50% sparsity and 0.42 at 75%. Sparse networks matching or surpassing dense counterparts have also been reported in Cannistraci–Hebb training (Zhang et al., 2024b); here the effect holds for PP-N:M ViT training. At 75% sparsity, S2S retains 76.58% accuracy, outperforming projected Muon by 3.44 points and Muon+PP by 3.17. At 50%, Muon+PP and S2S instead perform similarly, highlighting the greater benefit of the complete optimizer design at higher sparsity. Short-run validation tests across 2:4, 1:4, 4:8, and 2:8 also favor S2S over pattern-matched projected Muon by 3.86–5.68 points (Table 16).

4.2 Sparse ViT training on ImageNet-1K

All seven methods train a ViT from scratch on ImageNet-1K (Russakovsky et al., 2015) for 300 epochs, with global batch 1,024, 375,300 updates, five-epoch warm-up, and cosine decay. We evaluate the final checkpoint on the official validation set using two common seeds. Table 2 compares their final accuracy and CE; architecture and training details are in Appendix C.3.

Table 1: **CIFAR-100, ViT, 600 epochs.** Accuracy (%) and CE, mean ± SE over three seeds, except S2S at 75% sparsity (two). Dense references use 0% sparsity and appear once. Both projected baselines exclude PRE; SMET at 1:4 is a structured adaptation. Top-N AdamW uses $N = 2$ and $N = 1$, respectively. CE means Cross-Entropy Loss

	Dense Muon (0% sparsity)		**Dense AdamW** (0% sparsity)	
	Accuracy ↑	CE ↓	Accuracy ↑	CE ↓
	76.15 ± 0.27	1.11 ± 0.01	71.66 ± 0.08	1.23 ± 0.00
	50% sparsity (2:4)		**75% sparsity** (1:4)	
Method	Accuracy ↑	CE ↓	Accuracy ↑	CE ↓
S2S Muon (Ours)	77.06 ± 0.14	$\mathbf{1.04 \pm 0.01}$	$\mathbf{76.58 \pm 0.35}$	$\mathbf{1.05 \pm 0.01}$
Muon+PP (Ours)	$\mathbf{77.19 \pm 0.36}$	$\mathbf{1.04 \pm 0.01}$	73.41 ± 0.19	1.17 ± 0.01
Projected Muon	73.30 ± 0.13	1.17 ± 0.00	73.14 ± 0.12	1.17 ± 0.00
SMET	70.09 ± 0.21	1.28 ± 0.01	69.84 ± 0.15	1.31 ± 0.01
Top-N AdamW	72.29 ± 0.33	1.21 ± 0.01	69.77 ± 0.26	1.33 ± 0.00

Table 2: **ImageNet-1K, ViT, 300 epochs.** Final validation, mean ± SE over two seeds. Sparsity is the zero fraction of eligible linear weights. Bold marks the best mean within dense and sparse methods separately.

Method	Sparsity (%)	Top-1 (%) ↑	Top-5 (%) ↑	CE ↓
Dense AdamW	0	78.75 ± 0.25	$\mathbf{94.25 \pm 0.10}$	0.98 ± 0.02
Dense Muon	0	$\mathbf{79.28 \pm 0.09}$	94.21 ± 0.12	$\mathbf{0.97 \pm 0.01}$
Projected Muon	50	77.31 ± 0.08	93.63 ± 0.13	1.02 ± 0.01
Muon+PP-N:M (Ours)	50	77.49 ± 0.01	93.61 ± 0.01	1.02 ± 0.00
S2S Muon (Ours)	50	$\mathbf{78.21 \pm 0.00}$	$\mathbf{94.04 \pm 0.02}$	$\mathbf{0.98 \pm 0.00}$
Top2 AdamW	50	76.43 ± 0.18	93.05 ± 0.03	1.07 ± 0.00
SMET	50	77.91 ± 0.12	93.98 ± 0.02	0.99 ± 0.01

S2S achieves 78.21% top-1, exceeding projected Muon by 0.90 percentage points, Muon+PP by 0.72, and SMET by 0.30 (Table 2). Muon+PP improves on projected Muon by 0.18 points; dense Muon remains 1.08 points above S2S. S2S and Muon+PP also reduce the augmented training CE earlier than projected Muon at equal update counts (Figure 2). The Muon+PP comparison tests the support-associated matrix update; S2S adds the complete optimizer design.

4.3 Support and optimizer controls in vision

An independent two-seed, 30-epoch ImageNet study examines support assignment (Table 11). Replacing magnitude-guided PP support with locally balanced random PP changes accuracy by only -0.12 points; contiguous grouped support with matched component shapes reduces it by 2.62 points. A separate update diagnostic finds zero off-support energy for active-only PP input, versus 14.12% for row-wise 2:4 support (Table 15).

Separating direction and amplitude. A further two-seed, 30-epoch study fixes PP support and the first-moment recurrence while varying PRE and amplitude (Table 12). Without PRE, NS with Adam-reference scaling reaches 67.44% top-1, versus 60.73% for an Adam direction with the same target-RMS rule: a 6.71-point advantage in a same-device comparison. With and without PRE, replacing shape-based scaling by Adam-reference scaling improves the mean by 0.75 and 0.85 points, respectively. PRE adds only 0.14 points with Adam-reference scaling and has opposite effects across the two seeds; with shape-based scaling it adds 0.24 points. Thus the stronger evidence in this short-horizon study concerns direction construction and reference amplitude, while PRE's incremental benefit is configuration-dependent. Execution differences across the full panel are detailed in Appendix D.3; the separate 300-epoch PRE pair differs by only 0.02 points (Appendix D.2).

4.4 CROSS-DOMAIN VALIDATION: LANGUAGE MODELING

LLaMA-style decoders (Touvron et al., 2023) at 350M and 1B scale receive 61,500 OpenWebText updates of 16,384 tokens (1.008B training tokens), followed by full 5,315,072-token validation. Table 3 reports PPL; configurations are in Appendix C.

Table 3: **Language modeling.** Mean ± SE: three seeds at 350M and two at 1B.

Method	Sparsity (%)	350M PPL	1B PPL
Dense Muon	0	15.90 ± 0.01	13.94 ± 0.00
Dense AdamW	0	16.12 ± 0.04	14.68 ± 0.01
S2S Muon (Ours)	50	$\mathbf{16.88 \pm 0.02}$	$\mathbf{14.90 \pm 0.00}$
Projected Muon	50	17.33 ± 0.01	15.25 ± 0.01
SMET	50	17.97 ± 0.01	15.90 ± 0.00
Top2 AdamW	50	18.09 ± 0.01	15.47 ± 0.01

S2S reduces projected Muon's PPL by 2.59% at 350M and 2.31% at 1B. Dense Muon remains best at both scales. At 75% sparsity, S2S reaches 18.70 PPL versus 19.29 for projected Muon with PRE (Table 17). Fixed-support WikiText controls isolate PRE gains of 0.26/0.15 PPL under global/component normalization, with the same direction of improvement on OpenWebText (Table 14). A common-NS support control also favors PP over row-wise 2:4 (28.37 versus 29.60 PPL; Table 14).

4.5 EXECUTION EVIDENCE

On RTX 5090, we measure both Muon+PP versus projected Muon and full S2S with PP-linear versus the same update implemented with dense tensors and masks. The former includes support/update-domain differences; the latter isolates implementation. Each arm uses three fresh processes, ten warm-up and thirty timed complete eager training steps with device-resident synthetic inputs. Table 4 summarizes the memory–throughput trade-off; absolute throughput is in Appendix E.1.

Table 4: **PP-linear execution on RTX 5090.** Throughput is packed/reference, mean ± SE over three process repetitions; values below one indicate a slowdown. Memory is mean peak allocated GiB; Save is its percentage reduction. ViT micro/global batch is 16/64; LM is 4/32 with context 512.

Model	Throughput ratio	Ref. GiB	PP GiB	Save (%)
Muon+PP / Projected Muon				
ViT	0.85 ± 0.01	1.01	0.82	18.29
LM 1B	0.96 ± 0.00	20.51	12.26	40.22
S2S Muon: PP-linear / dense+mask				
ViT	1.27 ± 0.01	1.02	0.87	15.17
LM 1B	0.82 ± 0.00	25.45	18.05	29.09

Muon+PP saves 18.29% peak memory on ViT and 40.22% on the 1B language model, at $0.85\times$ and $0.96\times$ reference throughput. Full S2S saves 15.17% and 29.09%, respectively; its PP-linear implementation achieves $1.27\times$ throughput on ViT and $0.82\times$ on the language model. Packing reduces stored tensors, while activation storage, NS workspaces, channel movement, and kernel efficiency determine the workload-dependent throughput trade-off.

5 RELATED WORK

Sparse training: execution and storage. Unstructured pruning permits flexible connectivity, whereas N:M patterns enable regular sparse execution and compressed inference weights (Mishra et al., 2021). Transposable N:M patterns extend legality to forward and input-gradient products (Hubara et al., 2021; Hu et al., 2024a). Training methods differ in the information and state they retain: SR-STE and S-STE use dense latent weights behind sparse computation (Zhou et al., 2021; Hu et al., 2024b), while SLoPe demonstrates training-time and memory reductions (Mozaffari et al., 2025). Building on sparse training, we study the additional requirement imposed by matrix optimizers:

keeping internal matrix transformations on the active support while retaining a compact training representation. Section 2 formalizes the active-only, fixed-support, projection-free setting addressed here.

Matrix optimizers and sparsity-aware optimization. AdamW rescales coordinates independently (Loshchilov & Hutter, 2017); Shampoo uses matrix or tensor preconditioners (Gupta et al., 2018). Muon orthogonalizes matrix momentum approximately through NS (Jordan et al., 2024; Liu et al., 2025), with benefits reported in ViTs (Southworth et al., 2026). Related designs address pre-NS row/column balancing in MuonEq (Chang et al., 2026) and reduced optimization and communication costs through selected-row or selected-column orthogonalization in Dion2 (Ahn et al., 2025). For sparse optimization, STEP uses second-moment estimates for N:M mask learning (Lu et al., 2023), and SMET maintains active gradients and sparse-Adam states (Xiao et al., 2026). Our structural contribution concerns closure of the weight support under matrix products, beyond sparse coordinate-wise state storage. On this support, S2S combines pre-NS equilibration and component-wise direction construction with second-moment-based reference magnitude control. These update choices define the optimizer; PP-N:M provides the support and representation on which it operates.

Permutation for N:M pruning. Pool & Yu (2021) redistribute important weights among N:M groups with channel permutations and compatible changes to adjacent layers. Plug-and-Play (Zhang et al., 2024a) uses importance-and-activation scoring and assignment refinement for post-training language-model pruning. HiNM combines hierarchical pruning with GPU-oriented permutations (Yu et al., 2024). These uses of permutation improve the assignment of weights to pruning patterns. PP-N:M instead defines a support family through jointly balanced, packable components that remain closed throughout NS. Magnitude-guided selection chooses a member of this family; packing then preserves that matrix's row–column relations, rather than arbitrarily reshaping its nonzero entries. Unlike Monarch's product of structured factors (Dao et al., 2022), the representation describes one sparse matrix directly.

Packed execution throughout training. Sparse arithmetic, compressed weights, and whole-step training memory are distinct system properties: gradients, optimizer states, activations, and temporary buffers also contribute to the peak. PP-linear connects the format to execution by sharing the packed parameter layout across forward computation, both backward products, and the optimizer. Fused indexing avoids reconstructing physical-size weights, and weight-gradient kernels write directly into active storage. The current kernels multiply dense packed components; their benefits arise from compact representation and execution, rather than direct use of N:M Sparse Tensor Cores. We therefore measure peak memory and throughput separately for each workload and update rule.

6 Conclusion

We introduced PP-N:M, a sparse support family that combines bidirectional N:M constraints with exact packing and closure under Muon's matrix products. This structure enables sparse-to-sparse matrix optimization without generating and discarding off-support updates. Building on it, S2S Muon separates direction construction through PRE and component-wise NS from second-moment-based magnitude control, while PP-linear preserves compact weights, gradients, and optimizer states throughout training. S2S improves ViT accuracy over projected Muon by 0.90 percentage points on ImageNet-1K and 3.76 points on CIFAR-100 at 50% sparsity, with complementary perplexity improvements in 350M and 1B language models. PP-linear reduces measured peak training memory for both single-momentum Muon+PP and complete S2S, with workload-dependent throughput trade-offs. Together, these results show the value of designing sparse connectivity, matrix updates, and training kernels jointly: a support compatible with the optimizer can make sparsity a property of the entire update process, not just the forward weights. Our construction restricts support flexibility; its broader applicability and remaining execution challenges are discussed in Appendix F.

AI use statement

We used generative AI tools, including ChatGPT and Codex, to assist with experiment implementation and debugging of experimental and analysis code, interpretation and cross-checking of results, literature search, preparation of tables and figures. Verification included cross-checking reported

values against experimental records, recomputing summary statistics, and numerical agreement tests for the packed implementation. These checks do not replace author judgment. The authors take responsibility for the final methodology, experimental results, citations, and claims, including all content prepared with AI assistance.

Ethics statement

This work studies sparse training methods using existing ImageNet-1K, CIFAR-100, and OpenWebText datasets. It does not involve recruiting human participants or collecting new personal data. Existing datasets, particularly web-derived text, may contain biases, personal information, or harmful content; improved training efficiency does not remove these risks. Dataset access and redistribution remain subject to their respective terms. Our experiments assess optimization and computational behavior, not the safety or fairness of downstream deployments. We report both memory savings and throughput trade-offs, rather than assuming that sparsity necessarily reduces overall computational or environmental cost.

Reproducibility statement

The support construction and optimizer are defined in Sections 3.1 and 3.3, with structural proofs in Appendix A and implementation details in Appendix B. Appendix C describes the model configurations, data processing, training settings, and method-specific selection procedures. Tables state the seed counts and report standard errors for multi-seed results. Appendices D.2 and D.4 describe the additional controls, while Appendix E specifies the execution comparisons and measurement conditions. We will release the code when paper is accepted.

References

Kwangjun Ahn, Noah Amsel, and John Langford. Dion2: A simple method to shrink matrix in muon. *arXiv preprint arXiv:2512.16928*, 2025.

Alexander R Block, Simina Brânzei, Hemanta K Maji, Himanshi Mehta, Tamalika Mukherjee, and Hai H Nguyen. p_4-free partition and cover numbers and application. *Cryptology ePrint Archive*, 2020.

Jesse Cai, Daniel Haziza, and Supriya Rao. Accelerating neural network training with semi-structured (2:4) sparsity. PyTorch technical blog, 2024. URL `https://pytorch.org/blog/accelerating-neural-network-training/`.

Da Chang, Qiankun Shi, Lvgang Zhang, Yu Li, Ruijie Zhang, Yao Lu, Yongxiang Liu, and Ganzhao Yuan. Muoneq: Balancing before orthogonalization with lightweight equilibration. *arXiv preprint arXiv:2603.28254*, 2026.

Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in neural information processing systems*, 34:19974–19988, 2021.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018.

Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. Accelerating transformer pre-training with 2: 4 sparsity. *arXiv preprint arXiv:2404.01847*, 2024a.

Yuezhou Hu, Jun Zhu, and Jianfei Chen. S-ste: Continuous pruning function for efficient 2: 4 sparse pre-training. *Advances in Neural Information Processing Systems*, 37:33756–33778, 2024b.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 34:21099–21111, 2021.

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6(3):4, 2024.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Yucheng Lu, Shivani Agrawal, Suvinay Subramanian, Oleg Rybakov, Christopher De Sa, and Amir Yazdanbakhsh. Step: learning n: M structured sparsity masks from scratch with precondition. In *International Conference on Machine Learning*, pp. 22812–22824. PMLR, 2023.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Mohammad Mozaffari, Amir Yazdanbakhsh, Zhao Zhang, and Maryam Mehri Dehnavi. Slope: Double-pruned sparse plus lazy low-rank adapter pretraining of llms. In *International Conference on Learning Representations*, volume 2025, pp. 20656–20685, 2025.

Jeff Pool and Chong Yu. Channel permutations for n: M sparsity. *Advances in neural information processing systems*, 34:13316–13327, 2021.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Ben S Southworth, Shuai Jiang, Daniel McBride, Eric C Cyr, and Stephen Thomas. Muon in vision transformers: Optimizer-recipe interactions and gradient spectra. *arXiv preprint arXiv:2605.24770*, 2026.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Qiao Xiao, Boqian Wu, Patrik Okanovic, Tomasz Sternal, Maurice van Keulen, Elena Mocanu, Mykola Pechenizkiy, Decebal Constantin Mocanu, and Torsten Hoefler. Memory-efficient llm training with dynamic sparsity: From stability to practical scaling. *arXiv preprint arXiv:2606.00888*, 2026.

Seungmin Yu, Xiaodie Yi, Hayun Lee, and Dongkun Shin. Toward efficient permutation for hierarchical n: M sparsity on gpus. *arXiv preprint arXiv:2407.20496*, 2024.

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 *International Conference on Learning Representations*, volume 2024, pp. 50490–50508, 2024a.

Yingtao Zhang, Jialin Zhao, Wenjing Wu, Alessandro Muscoloni, and Carlo Vittorio Cannistraci. Epitopological learning and cannistraci-hebb network shape intelligence brain-inspired theory for ultra-sparse advantage in deep learning. In *International Conference on Learning Representations*, volume 2024, pp. 46211–46239, 2024b.

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. *arXiv preprint arXiv:2102.04010*, 2021.

A Packing and Matrix-update Identities

Proposition 1 (Balanced PP-N:M construction)**.** *Under the assumptions in Section 3.1, the support has exactly N active entries per consecutive $1 \times M$ group in W and $W^\top$, contains mn/K active entries, and can be permuted into K dense diagonal blocks of shape $(m/K) \times (n/K)$.*

Proof. Fix $i \in R_k$. A consecutive group of M columns contains exactly N members of C_k, and these are precisely the retained coordinates in that row. Fixing a column instead proves transpose legality. Balance over all groups gives $|R_k| = m/K$ and $|C_k| = n/K$. Ordering rows by R_k and columns by C_k leaves nonzero entries only in matching blocks, with total size $K(m/K)(n/K) = mn/K$. □

The graph-theoretic equivalence between bipartite induced-P_4-free graphs and disjoint unions of bicliques is classical (Block et al., 2020). The following proposition connects it to a value-independent matrix-update condition and derives the additional balance required by bidirectional N:M sparsity.

Proposition 2 (N:M support compatibility with cubic matrix products)**.** *For a binary mask Ω, the following are equivalent: (i) $UU^\top U$ is supported on Ω for every real matrix U supported on Ω; (ii) after removing isolated rows and columns, the bipartite graph of Ω is a disjoint union of complete bipartite components. If Ω is additionally bidirectional N:M, with nonzero N and both dimensions divisible by M, these components have sizes $(mN/M) \times (nN/M)$, their number is M/N, and their memberships satisfy the balance conditions of Equation* (2)*.*

Proof. The block-product identity proves (ii)⇒(i). Conversely, an alternating path $i \to j \to i' \to j'$ in the support graph contributes $U_{ij}U_{i'j}U_{i'j'}$ to $(UU^\top U)_{ij'}$. Set all supported entries to one. If (i, j') were missing, the resulting positive off-support entry would contradict (i). Every length-three path must therefore have its endpoint edge. In a connected bipartite component, repeatedly shortening an odd path by this rule shows that every row vertex connects to every column vertex, proving (ii).

Bidirectional N:M implies that every row has degree nN/M and every column degree mN/M. In a complete bipartite component these are its column and row counts. Thus all components have the stated equal size, and there are M/N of them. The local N:M requirement forces exactly N members of a component in every consecutive group of M indices on either side. In particular, the number M/N must be an integer. □

This characterizes masks for a value-independent cubic-product property, not all possible sparse matrix optimizers. An optimizer using different operations or permitting projection need not satisfy the premise.

Which N:M masks are covered? A PP mask is a union of disjoint complete bipartite row–column components. Arbitrary row-wise N:M masks need not have this form. Even a mask legal in both directions need not be packable: an 8×8 mask with active entries at $j = i$ and $j = (i + 1) \bmod 8$ is bidirectional 2:8, but its bipartite graph is a connected cycle, rather than four disjoint 2×2 dense blocks. Thus the construction trades support flexibility for regular computation. The equal-block construction requires integer $K = M/N$; ratios such as 3:4 are not covered. Dimensions not divisible by M require separate treatment and are not used for eligible layers in our experiments.

NS with shared and independent normalization. For a fixed scalar $s > 0$ and polynomial $F(Z) = aZ + bZZ^\top Z + c(ZZ^\top)^2 Z$,

$$F(\mathrm{diag}(U_1/s, \ldots, U_K/s)) = \mathrm{diag}(F(U_1/s), \ldots, F(U_K/s)). \tag{11}$$

Repeated composition preserves this identity. Row and column permutations commute with these matrix products, so the same statement holds in physical coordinates after undoing the permutations. This establishes equivalence for a common normalization s, *not* between common normalization and independent choices $s_k = \|U_k\|_F$ at finite iteration count. No convergence assumption is needed for this algebraic statement.

Exact polar reference. For the canonical partial polar factor $\mathcal{P}(U) = (UU^\top)^{\dagger/2}U$, with the inverse square root taken on nonzero singular directions,

$$\mathcal{P}(\mathrm{diag}(U_1, \ldots, U_K)) = \mathrm{diag}(\mathcal{P}(U_1), \ldots, \mathcal{P}(U_K)). \tag{12}$$

For full-rank rectangular blocks this is the usual semi-orthogonal polar factor. The pseudoinverse convention avoids choosing an arbitrary completion in a null space. Positive scalar normalization does not change this exact factor, while row/column PRE generally does. Exact polar calculation is a reference for numerical accuracy, not a theoretical upper bound on validation accuracy.

B Algorithm and PP-linear Details

Matrix updates. Each component is transposed when needed to form the smaller Gram matrix, normalized by its Frobenius norm, and transformed by five iterations of

$$G = ZZ^\top, \qquad Z \leftarrow aZ + (bG + cG^2)Z, \qquad (a, b, c) = (3.4445, -4.7750, 2.0315). \tag{13}$$

These are the finite-step Muon coefficients (Jordan et al., 2024). S2S uses $(\beta_1, \beta_2) = (0.9, 0.95)$, Adam-reference $\epsilon = 10^{-8}$, PRE floor 10^{-8}, and normalization and final rescaling floors 10^{-12}. PRE computes column statistics after row normalization, unlike the original-input statistics in MuonEq (Chang et al., 2026). The second moment supplies the Adam-reference amplitude, not PRE itself.

Balanced support selection. At initialization, we approximately maximize $\sum_k \sum_{i \in R_k, j \in C_k} (W^{\mathrm{d}}_{ij})^2$ under local balance. For 2:4, column groups repeat one of the three unordered partitions of four offsets into pairs. For each partition, rows are ranked by retained-energy difference, assigning two of every four rows to each component; the highest-energy partition is selected. For 1:4, columns follow the four offset residue classes, and each four-row group tests all 24 assignments. Ties are resolved deterministically. The selected topology stays fixed, and retained weights are rescaled as in Equation (4).

Packed linear computation. The PP-linear backend stores active components contiguously and accesses physical channels through row/column index lists. Triton kernels fuse channel gathering with tiled dense products for forward, input gradients, and active weight gradients. The packed components are dense; this path does not invoke hardware sparse-matmul instructions. Forward saves the compute-dtype input, compact weights, and topology, rather than an additional permuted activation buffer. Products accumulate in FP32; forward and input-gradient outputs use the activation dtype, while packed weight gradients remain FP32. The measured implementation also allocates a BF16 weight cache, included in its memory accounting.

For an eligible $m \times n$ layer with s optimizer-state tensors, FP32 weights, gradients, and states occupy $4(2 + s)mn/K$ bytes before indices, caches, and workspaces. Table 5 separates these persistent tensors from the whole-model peaks measured later.

C Experimental Settings

C.1 Models, data, and reporting

Vision models. ImageNet uses ViT-S/16 with width 384, 12 blocks, six heads, MLP width 1,536, and 224×224 inputs. Its 72 attention/MLP matrices are sparsity-eligible; patch projection, classifier, biases, normalization, and tokens remain dense. CIFAR-100 uses the corresponding ViT-S/4 at 32×32 resolution. They use bias-free attention/MLP projections and no pretrained weights.

Table 5: Eligible tensor storage, in units of mn FP32 elements. Activation memory, indices, BF16 caches, and temporary workspaces are excluded.

Representation / optimizer	Weights	Gradients	Moments	Total
Dense / canonical Muon	1.00	1.00	1.00	3.00
Dense / AdamW	1.00	1.00	2.00	4.00
PP-2:4 / Muon	0.50	0.50	0.50	1.50
PP-2:4 / S2S	0.50	0.50	1.00	2.00
PP-1:4 / S2S	0.25	0.25	0.50	1.00

Language models. The decoders use RMSNorm, rotary position embeddings, SwiGLU, and grouped-query attention (Touvron et al., 2023), with the configurations in Table 6. Embeddings and the tied output head remain dense.

Table 6: Language model configurations. "350M" and "1B" denote model scales; exact dense parameter counts are listed.

Property	350M	1B
Layers	28	22
Model width	1,024	2,048
Feed-forward width	2,816	5,504
Query heads / KV heads	16 / 4	16 / 4
Vocabulary	16,384	16,384
Dense parameters	332,456,960	1,008,297,984
Eligible linear parameters	315,621,376	974,651,392
Eligible matrices	196	154
Context length	512	512

OpenWebText is exactly deduplicated and ordered by canonical-text hashes; 4,096 documents each form validation and test, with 8,005,577 for training. The vocabulary has 16,384 tokens. Each run consumes 1,007,616,000 training tokens in 61,500 updates of 32 sequences of length 512. The schedule warms up for 1,230 updates and decays to 10% of peak LR. Final PPL uses all 5,315,072 targets in complete, non-overlapping validation windows.

Uncertainty. Results report mean $\pm$ SE across independent training seeds, with $\mathrm{SE} = s/\sqrt{n}$ and sample standard deviation s using $n - 1$. PPL is computed per seed before aggregation.

C.2 Baseline definitions

Dense AdamW optimizes all weights; Top-N AdamW keeps an initialization-time, row-wise magnitude-selected mask. Dense and projected Muon use Nesterov momentum 0.95 and shape-based scale $0.2\sqrt{\max(m, n)}$. Projected Muon applies NS to a zero-filled physical momentum matrix and projects its output; Muon+PP applies independently normalized NS to PP components. Both use one momentum and no Adam-reference amplitude. S2S uses both moment statistics, PRE, and Adam-reference scaling. Table 7 summarizes these matrix-update differences.

Table 7: Sparse matrix-update definitions. Canonical shape scaling uses the physical layer dimensions. Auxiliary tensors use AdamW.

Choice	Projected Muon	Muon+PP	S2S Muon
Support	Row-wise Top-N	PP-N:M	PP-N:M
Momentum	Nesterov, 0.95	Nesterov, 0.95	Two moments, 0.90/0.95
NS domain	Physical matrix	PP components	PP components
PRE	No	No	Yes
Amplitude	Shape-based	Shape-based	Adam reference

SMET (Xiao et al., 2026) is adapted to row-wise N:M constraints with sparse Adam updates, support refresh every 100 updates, and a ten-update warm-up for newly active entries. The CIFAR adaptation

uses $\beta_2 = 0.999$ and zero matrix weight decay; at 1:4, turnover decays from 0.20 to 0.005 and the matrix LR is twice the reported base LR. ImageNet uses turnover 0.20 and its common auxiliary optimizer. These are structured adaptations, not PP-support controls.

C.3 IMAGENET TRAINING AND HYPERPARAMETERS

All methods train from scratch on 1,281,167 images for 300 epochs (375,300 updates), with global batch 1,024, five-epoch warm-up, cosine decay to 1% of peak LR, and weight decay 0.05. Augmentation comprises random resized crops, horizontal flips, RandAugment `rand-m9-mstd0.5-inc1`, color jitter 0.4, random erasing 0.25, Mixup $\alpha = 0.8$, CutMix $\alpha = 1$, and label smoothing 0.1. Evaluation uses bicubic resize to 256 and a 224 center crop on the official 50,000-image validation set. No distillation, EMA, or stochastic depth is used.

All arms clip the global L2 norm of actual optimized gradients at 1. They share the dense initialization within each seed and preserve physical weight RMS when sparsifying. Muon+PP and S2S share PP support and active initial values; projected Muon, Top2 AdamW, and SMET share the initial row-wise mask, with SMET subsequently updating its support. Gradient accumulation preserves global batch across two-/four-GPU runs.

LR selection holds out 50 images per class using split seed 20260920. Seven doubling-spaced candidates are screened with seed 20260923, then the best two are repeated with seed 20260924. Each trial runs 36,060 updates from the beginning of the 300-epoch schedule. Mean holdout CE across the two selection seeds determines the final LR. All selected rates are interior to the tested ranges; the 7+2 trials use 324,540 selection updates per method. Formal seeds are independent of selection. Table 8 lists the tested ranges and selected rates.

Table 8: ImageNet peak learning rates. Each range contains seven doubling-spaced candidates; the best two receive a second selection seed.

Method	Tested range	Selected LR	Trials
Dense AdamW	1.00e-4–6.40e-3	8.00e-4	7+2
Dense Muon	1.25e-4–8.00e-3	5.00e-4	7+2
Projected Muon (no PRE)	1.25e-4–8.00e-3	1.00e-3	7+2
Muon+PP-N:M	1.25e-4–8.00e-3	5.00e-4	7+2
S2S Muon	2.00e-4–1.28e-2	8.00e-4	7+2
Top2 AdamW	2.00e-4–1.28e-2	8.00e-4	7+2
SMET (adapted)	2.00e-4–1.28e-2	1.60e-3	7+2

C.4 CIFAR TRAINING AND HYPERPARAMETERS

The training split contains 45,000 images, with 5,000 held out for selection; the official 10,000-image test set is used only for final evaluation. Training lasts 600 epochs (52,800 updates), with batch 512, 5% warm-up, cosine decay to 1% of peak LR, Mixup $\alpha = 0.8$, CutMix $\alpha = 1$, and label smoothing 0.1.

The initial LR grids are $\{1, 2, 4\} \times 10^{-4}$ for dense AdamW, $\{8, 15, 30\} \times 10^{-4}$ for dense Muon, $\{2, 4, 8\} \times 10^{-4}$ for SMET, and $\{4, 8, 15\} \times 10^{-4}$ for other sparse methods. The initial funnel uses two selection seeds (206107/206109) and 5,280/10,560-update horizons. All six original methods follow the same preregistered boundary-extension rule: if the second-stage winner is an endpoint of its initial grid, one additional candidate is tested outward using the adjacent grid ratio. For sorted candidates $\eta_1 < \cdots < \eta_k$, this candidate is η_k^2/η_{k-1} at the upper boundary or η_1^2/η_2 at the lower boundary. It is evaluated on both selection seeds to 10,560 updates under the same 52,800-update schedule, then compared with the boundary winner using the same validation criterion before freezing the LR. This method-independent rule gives every original method the same opportunity for one extension, rather than reserving it for S2S. S2S, SMET, and Top-2 AdamW triggered the rule; only S2S selected the outward candidate. In particular, $(1.50 \times 10^{-3})^2/(8.00 \times 10^{-4}) = 2.8125 \times 10^{-3}$. The S2S 1:4 study uses the same outward candidate when its upper boundary wins. Dense AdamW additionally compares 2×10^{-4} and 4×10^{-4} at the full horizon. Thus the original extension rule is shared, but conditional extensions and subsequent validation additions do not imply identical total search budgets across all reported methods.

Table 9: CIFAR peak learning rates. Dense references have 0% sparsity. Rates exclude SMET's method-specific matrix multiplier.

Method	2:4	1:4
Dense Muon	1.50×10^{-3}	
Dense AdamW	4.00×10^{-4}	
S2S Muon	2.81×10^{-3}	2.81×10^{-3}
Muon+PP-N:M	2.81×10^{-3}	8.00×10^{-4}
Projected Muon (no PRE)	8.00×10^{-4}	2.81×10^{-3}
SMET (adapted)	8.00×10^{-4}	8.00×10^{-4}
Top-N AdamW	1.50×10^{-3}	8.00×10^{-4}

C.5 Language hyperparameters

Both scales use selection seeds 180001/180002 and 2,000 training updates per candidate from the beginning of the 61,500-update schedule. The 1B Dense AdamW, SMET, and Top2 AdamW additions score all 10,381. Final evaluation uses the same full validation set for all methods. Table 10 lists the candidate and selected rates.

Table 10: Language LR grids and selected peak learning rates.

	350M		1B	
Method	Candidates	Selected	Candidates	Selected
Dense Muon	$\{8, 15\} \times 10^{-4}$	1.50×10^{-3}	$\{8, 15\} \times 10^{-4}$	8×10^{-4}
Dense AdamW	$\{3, 6\} \times 10^{-4}$	6×10^{-4}	$\{3, 6\} \times 10^{-4}$	3×10^{-4}
S2S Muon	$\{4, 8\} \times 10^{-4}$	8×10^{-4}	$\{4, 8\} \times 10^{-4}$	4×10^{-4}
Projected Muon	$\{8, 15\} \times 10^{-4}$	8×10^{-4}	$\{4, 8\} \times 10^{-4}$	8×10^{-4}
SMET	$\{4, 8\} \times 10^{-4}$	8×10^{-4}	$\{4, 8\} \times 10^{-4}$	4×10^{-4}
Top2 AdamW	$\{8, 15\} \times 10^{-4}$	1.50×10^{-3}	$\{8, 15\} \times 10^{-4}$	8×10^{-4}

At 350M/1:4, S2S and projected Muon test $\{4, 8, 16\} \times 10^{-4}$ and select 8×10^{-4}; Top1 AdamW tests $\{8, 15, 30, 60\} \times 10^{-4}$ and selects 3×10^{-3}. The original 0.005 tie threshold is retained. Structured SMET tests $\{4, 8\} \times 10^{-4}$ and selects 8×10^{-4} with the full-validation selection protocol. All use two selection seeds and 2,000 updates per candidate.

D Additional Results and Analyses

D.1 ImageNet training trajectories

Figure 2 compares the augmented training objective over the full run and its early epochs.

At epoch 60, training CE is 3.98 for S2S, 3.97 for Muon+PP, and 4.19 for projected Muon. Both PP methods reduce this objective earlier than projected Muon at equal update counts; dense Muon remains lower. Figure 2 shows more details.

D.2 ImageNet support and optimizer ablations

The eight-arm study uses an independent 30-epoch schedule with three-epoch warm-up and the main ImageNet data/augmentation protocol. Each arm screens $\{4, 8, 16\} \times 10^{-4}$ for ten epochs on one selection seed; all select the upper candidate within this fixed budget. Two independent formal seeds then train on the full training set for 37,530 updates.

Random balanced PP samples locally legal assignments without magnitude scoring. Contiguous grouped support matches active count and component shapes but lacks local 2:4 balance. All arms share the dense initialization and physical-RMS rescaling; same-support arms share active initial values. Common normalization is a single Frobenius norm over a layer's active components. Both moment statistics and Adam-reference amplitude remain enabled when PRE or normalization is changed. Table 11 reports the eight configurations.

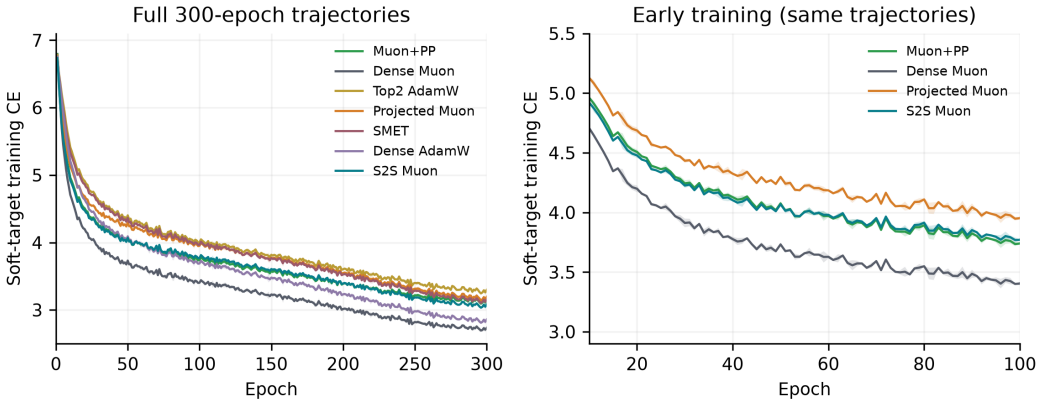

Figure 2: **ImageNet training trajectories.** Mean and shaded SE over two seeds. Each point averages losses logged every 25 updates within an epoch. The objective uses Mixup/CutMix and label smoothing; it is not clean-image training CE. Right: early detail from the same 300-epoch runs.

Table 11: **ImageNet ablations.** 30 epochs, mean ± SE over two seeds.

Configuration	Top-1 (%)	CE
Magnitude PP, full S2S	67.98 ± 0.11	1.47 ± 0.01
Random balanced PP, full S2S	67.85 ± 0.00	1.47 ± 0.00
Magnitude PP, S2S without PRE	67.67 ± 0.35	1.48 ± 0.01
Magnitude PP, S2S with common norm	67.88 ± 0.28	1.47 ± 0.01
Magnitude PP, common norm, no PRE	67.57 ± 0.33	1.48 ± 0.02
Contiguous grouped support, S2S	65.36 ± 0.63	1.58 ± 0.03
Magnitude PP, active AdamW	59.81 ± 0.06	1.87 ± 0.01
Random balanced PP, active AdamW	60.17 ± 0.10	1.85 ± 0.01

S2S exceeds active AdamW on magnitude-selected PP by 8.17 points in this original recipe comparison, which does not match update amplitudes; magnitude and random balanced support differ by only 0.12 points. PRE adds 0.31 points on average, with paired gains of 0.55/0.07 under component normalization and 0.35/0.26 under common normalization. Independent normalization is not uniformly better across seeds.

Long-horizon PRE control. One paired 300-epoch seed gives 78.21% for full S2S and 78.19% without PRE, holding the other optimizer components and LR fixed. This single pair uses different devices and does not establish a precise PRE gain.

Support versus routing. Random PP support changes the physical connectivity; random NS routing instead redistributes fixed active entries into work matrices before NS and inverts that assignment afterward. An exploratory two-seed 30-epoch routing run reaches 58.97% top-1; the different reference device precludes attributing the gap solely to routing.

D.3 Fixed-momentum direction and amplitude controls

To separate optimizer components, we hold magnitude-selected PP-2:4 support, initialization, and the bias-corrected first moment ($\beta_1 = 0.9$) fixed. Four NS configurations cross PRE on/off with Adam-reference or shape-based amplitude; all use component-wise FP32 NS5. For a physical $m \times n$ matrix, shape scaling uses $D_t = 0.2\sqrt{\max(m,n)}Q_t$ and stores no second moment. The fifth configuration replaces NS by the Adam direction $D_t = 2A_t$ without PRE. Both it and the Adam-reference NS arms use $\beta_2 = 0.95$ and target $\mathrm{RMS}_\Omega(D_t) = 2\,\mathrm{RMS}_\Omega(A_t)$ (away from numerical floors). This matches the amplitude rule, not the numerical updates along diverging training trajectories. The shape arms retain the same first-moment recurrence as S2S and are not the Nesterov Muon+PP baseline.

All five configurations train for 30 epochs (37,530 updates), with global batch 1,024, three-epoch warm-up, cosine decay, a fixed peak LR 1.60×10^{-3}, weight decay 0.05, and active-plus-auxiliary gradient clipping at 1. Auxiliary parameters use AdamW with $(0.9, 0.95)$. Table 12 reports final official-validation accuracy for seeds 20260925 and 20260926, separately from the original eight-arm study and the 300-epoch main comparison.

Table 12: **Fixed-momentum ImageNet controls.** PP-2:4, 30 epochs, mean $\pm$ SE over two seeds. All NS rows use the same FP32 NS5 precision.

Direction	Amplitude	PRE	Top-1 (%) $\uparrow$
NS (full S2S)	Adam-reference	Yes	67.59 ± 0.01
NS	Adam-reference	No	67.44 ± 0.37
NS	Shape-based	Yes	66.84 ± 0.23
NS	Shape-based	No	66.59 ± 0.10
Adam	Matched ($2A_t$)	No	60.73 ± 0.21

The no-PRE NS and matched-Adam runs use two H200 GPUs each; NS wins by 6.14/7.29 points across the two seeds. Full S2S and the shape arms run on RTX 5090: full S2S uses two GPUs, whereas each shape arm uses one GPU for the first seed and two for the second. Global batch and update counts are unchanged. Thus the full panel is not a hardware-identical factorial comparison. Adam-reference scaling improves on shape scaling in both seeds with or without PRE; PRE itself yields $+0.50/-0.21$ points under Adam-reference scaling and $+0.11/+0.37$ under shape scaling. These short-horizon observations do not establish a uniform PRE benefit.

Learning-rate response. An independent seed (20260927) evaluates all five configurations at $\{0.80, 1.60, 3.20\} \times 10^{-3}$ using ten-epoch prefixes of the 30-epoch schedule. These runs use a 50,000-image training-set holdout, not the official validation set, and do not select or change the fixed LR above. Table 13 reports every tested point.

Table 13: **Learning-rate response of the fixed-momentum controls.** Holdout top-1 (%), ten epochs, one independent seed; no cross-seed error bars.

Configuration	0.80×10^{-3}	1.60×10^{-3}	3.20×10^{-3}
Full S2S	52.44	52.81	50.50
NS, Adam-reference, no PRE	52.28	52.04	50.00
NS, shape, PRE	50.90	52.10	50.01
NS, shape, no PRE	50.87	51.43	49.96
Matched Adam, no PRE	43.16	41.11	38.33

The no-PRE NS and matched-Adam runs use the same RTX 5090 host, with NS ahead at all three tested rates.

D.4 Language support and update controls

The WikiText-103 study holds PP-2:4 support, initialization, data order, LR 4×10^{-4}, and five NS steps fixed for 5,000 updates. OpenWebText controls use LR 8×10^{-4}, the original 61,500-step cosine horizon, and endpoints at 130M/5K and 350M/10K. They share two-moment Adam-reference amplitude, and the auxiliary optimizer. Global normalization uses one Frobenius norm across a layer's active components, without adding inactive zeros. Table 14 compares PRE and normalization, followed by the support control.

In WikiText, PRE reduces PPL by 0.26 with global normalization and 0.15 with component normalization; both paired seeds improve. OpenWebText also favors PRE under both normalization choices. These results support PRE's contribution in language modeling, while the effect of independent normalization depends on the configuration.

NS precision. A paired 350M/5K projected-Muon RMS-repair control changes BF16-minus-FP32 PPL by -0.13 and $+0.13$ across two seeds, with no consistent advantage. It uses matched

Table 14: **Language update and support controls.** PPL, mean ± SE over two seeds. Top: fixed PP-2:4 support, varying PRE and normalization. Bottom: 350M/10K, common physical-coordinate NS and no PRE, varying support and its retained initialization.

Update	WikiText 350M/5K	OWT 130M/5K	OWT 350M/10K
Global, no PRE	22.96 ± 0.21	36.22 ± 0.10	28.37 ± 0.08
Component, no PRE	22.90 ± 0.22	36.25 ± 0.07	28.35 ± 0.11
Global + PRE	22.70 ± 0.19	35.55 ± 0.11	27.95 ± 0.06
Component + PRE	22.75 ± 0.25	35.62 ± 0.15	28.07 ± 0.10
Support control: 350M OpenWebText, 10K updates			
PP-2:4		28.37 ± 0.08	
Row-wise Top2		29.60 ± 0.01	

initialization, support and sequence order, LR 8×10^{-4}, and full validation. This short control does not establish equivalence of NS precision in every main-table recipe.

Polar-based reference. At 350M/15K, the PRE/NS5 recipe gives 21.78 ± 0.01 PPL versus 21.87 ± 0.00 for a regularized full-axis polar recipe (two seeds). Both use active moments and Adam-reference amplitude, but the polar reference additionally applies post-transform equilibration and an eigenvalue floor. This is a recipe comparison, not an isolated test of exact versus approximate polar computation.

D.5 Projection diagnostics

For the pre-projection NS output Q, define

$$e = \frac{\|(1-\Omega) \odot Q\|_F^2}{\|Q\|_F^2}, \qquad r = \frac{\|Q - \Omega \odot Q\|_F}{\|Q\|_F} = \sqrt{e}. \tag{14}$$

The 350M/5K control varies support and optimizer-visible gradients while retaining Adam-referenced scaling and NS5. Diagnostics are averaged over 196 eligible layers, then across two seeds. Table 15 compares the discarded energy and PPL for the two support families.

Table 15: **Projection diagnostics.** Mean ± SE across two seeds. Energy e and relative error r are averaged separately over layers.

Support	Gradient	PPL	Mean e (%)	Mean r
PP-2:4	Dense	32.31 ± 0.04	50.02 ± 0.00	0.71 ± 0.00
PP-2:4	Active only	32.14 ± 0.01	0.00 ± 0.00	0.00 ± 0.00
Row-wise 2:4	Dense	32.36 ± 0.05	50.02 ± 0.00	0.71 ± 0.00
Row-wise 2:4	Active only	33.53 ± 0.02	14.12 ± 0.04	0.37 ± 0.00

PP with active-only input produces exactly zero off-support output, whereas arbitrary row-wise support discards 14.12% mean energy. Dense-gradient inputs discard approximately 50% under either mask. This measures the update removed by projection, not a monotone predictor of final PPL. Since averaging and square root do not commute, $\bar{r}$ need not equal $\sqrt{\bar{e}}$.

D.6 Other N:M patterns

The short CIFAR study compares S2S with active AdamW on identical PP support, and projected Muon on pattern-matched row-wise support. Two seeds train for 4,000 updates at batch 512, with 400-step warm-up, cosine floor 0.01, and matrix weight decay 0.05; auxiliary AdamW uses LR 0.001 and coefficients $(0.9, 0.95)$. Results use all 5,000 validation images rather than the official test set.

The added projected controls select among $\{8, 16, 32\} \times 10^{-4}$ using 1,000 updates and 2,048 validation images at seed 160001; all choose the lower endpoint within this fixed budget. Formal seeds are 161001/162001. They use Nesterov 0.95, physical-shape scaling, and no PRE or Adam-reference amplitude. Table 16 compares the four N:M patterns under this short-run protocol.

Table 16: **N:M generalization.** CIFAR validation accuracy (%), mean ± SE over two seeds after 4,000 updates. Active AdamW and S2S are support-matched; projected Muon is pattern-matched.

Pattern	Active AdamW	Projected Muon	S2S Muon (Ours)
2:4	49.77 ± 0.07	61.64 ± 0.56	67.14 ± 0.42
1:4	49.94 ± 0.42	60.88 ± 0.14	64.74 ± 0.14
4:8	50.23 ± 0.33	62.16 ± 0.62	67.84 ± 0.34
2:8	50.59 ± 0.05	59.98 ± 0.36	65.04 ± 0.16

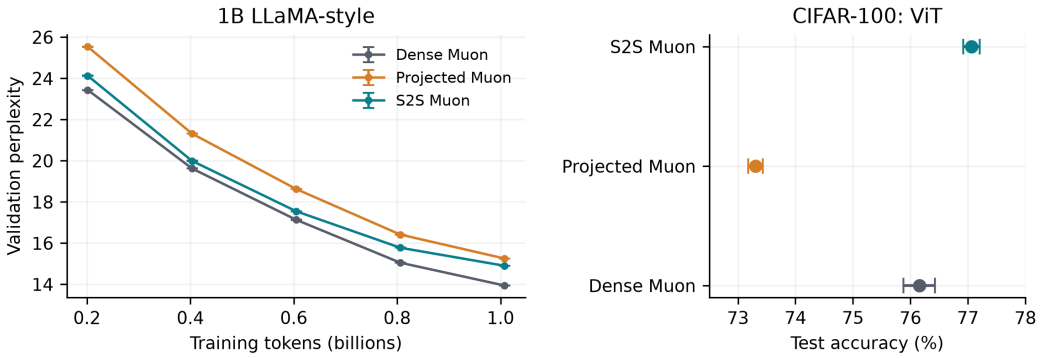

Figure 3: **Cross-domain results.** Left: 1B validation PPL at common training-token budgets (two seeds). Right: final CIFAR accuracy at 50% sparsity (three seeds). Error bars show SE.

PP supports both $K = 2$ and $K = 4$. Equal sparsity does not imply identical support constraints: 1:4 differs from 2:8, and 2:4 from 4:8.

D.7 Higher-sparsity language results and learning curves

The complete 350M/1:4 study uses the same 1.008B-token budget as the main language experiments. PP-1:4 retains one quarter of eligible weights. Its S2S PPL rises from 16.88 at 2:4 to 18.70, but remains below the projected-PRE and AdamW baselines at 1:4 (Table 17).

Table 17: **350M language modeling at 75% sparsity.** Final PPL after 1.008B tokens, mean ± SE over three seeds. The projected recipe includes PRE. Dense references are reproduced for context.

Method	Sparsity (%)	PPL
Dense Muon	0	15.90 ± 0.01
Dense AdamW	0	16.12 ± 0.04
S2S Muon (ours)	75	18.70 ± 0.01
Projected Muon	75	19.29 ± 0.01
SMET (1:4)	75	19.70 ± 0.07
Top1 AdamW	75	21.06 ± 0.06

Figure 3 complements the final language results with 1B learning curves and the CIFAR comparison at 50% sparsity.

E Systems Performance

E.1 Two-model execution comparison

Both workloads use the same RTX 5090 protocol: three fresh-process repetitions per arm in mirrored order, ten warm-up and thirty timed updates, BF16 forward/backward, FP32 master weights and gradients, disabled TF32, no activation checkpointing, and no CUDA graphs. ViTs use micro/global

batch 16/64; language models use 4/32 with context 512. Timing includes gradient reset, accumulation, both optimizers, and cache/scatter maintenance, but excludes input loading and compilation. Inputs are device-resident synthetic data. Memory is peak allocated memory after warm-up.

Comparison definitions. The S2S pair holds support and update rules fixed. Its dense+mask reference retains compact moments and an active shadow parameter; physical weights, gradients, masks, and gather/scatter costs are included. The Muon+PP pair instead compares PP components with row-wise projected Muon: both use one Nesterov momentum and BF16 NS without PRE. The projected reference already stores momentum on active coordinates. These two comparisons therefore have different meanings and denominators. Table 18 reports the absolute throughput and memory for both comparisons.

Table 18: **Two-model RTX 5090 execution.** Throughput is images/s for ViT and tokens/s for LM; ratios are packed/reference. Throughput and ratios report mean ± SE across three repetitions; memory columns are mean allocated peaks in GiB. Reference is dense+mask S2S in the upper panel and projected Muon below.

Model	Ref. throughput	PP throughput	Ratio	Ref. GiB	PP GiB	Save (%)
S2S Muon: PP-linear / dense+mask						
ViT-S/16	370.22 ± 7.15	469.45 ± 3.88	1.27 ± 0.01	1.02	0.87	15.17
LM 1B	10186.13 ± 0.75	8364.15 ± 5.70	0.82 ± 0.00	25.45	18.05	29.09
Muon+PP / Projected Muon						
ViT-S/16	325.08 ± 1.27	275.56 ± 5.81	0.85 ± 0.01	1.01	0.82	18.29
LM 1B	9551.63 ± 4.90	9197.47 ± 19.92	0.96 ± 0.00	20.51	12.26	40.22

Both comparisons save memory on both models. Full S2S speeds up ViT-S/16 by 26.86%, while the other workloads exhibit throughput trade-offs. Before timing, each eligible matrix shape is checked for forward, input-gradient, and active-weight-gradient agreement with the reference; relative L_2 errors are below 0.015, and identical supplied gradients produce identical active parameters after two updates.

Activation-memory sensitivity. An earlier framework-packed ViT control at microbatch 128 reduced persistent model/state storage but increased allocated peak memory by 1.81%. Unlike the fused path above, it retains explicitly permuted activations. Peak memory therefore depends on activation storage and microbatch size as well as packed parameter storage.

E.2 Where the time goes

A separate 1B RTX 5090 diagnostic profiles the forward/backward and matrix optimizer phases at micro/global batch 4/32. Each of three processes uses three warm-up and five instrumented updates; other full-step costs, including auxiliary AdamW, and cache refresh, are outside these phase totals.

The PP matrix update is $2.85\times$ faster (270.39 versus 769.33 ms) in this diagnostic, but its forward/backward path is slower. Reducing optimizer work alone therefore does not guarantee end-to-end acceleration.

Backend extensions. PP-N:M's physical weights are transposable sparse matrices, while its packed components are dense. A future physical-coordinate sparse backend could retain compact optimizer states, but must include layout conversion and weight-gradient costs. Transposable sparse training techniques (Hu et al., 2024a) offer complementary implementation directions; packing and sparse Tensor Core speedups cannot simply be multiplied because packing has already removed the inactive weights.

F LIMITATIONS

PP-N:M requires integer M/N and restricts support flexibility. Our quality results cover one ViT architecture per dataset and small language models; ImageNet uses two seeds, and long-horizon PRE evidence is limited to one pair. The modest visual PRE gains do not establish a universal benefit. PP-linear reduces memory in the tested workloads, but throughput depends on kernel efficiency and activation costs; broader architectures and backends remain to be evaluated.