TOWARDS 1:4 SEMI-STRUCTURED 75% SPARSITY VIA CANNISTRACI–HEBB N:M DYNAMIC SPARSE-TO-SPARSE TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving the computational and energy efficiency of large language models motivates semi-structured N:M sparsity, but preserving model quality becomes difficult when moving from 2:4 (50%) to 1:4 (75%) sparsity. We propose CHTsNM, a sparse-to-sparse pretraining framework that combines Cannistraci–Hebb topology evolution, topology-aware optimization, and contextually modulated low-rank compensation. CHTsNM directly trains the executed sparse backbone while evolving legal row-wise or transposable block-wise supports. Its Topology-Aware Nesterov Optimizer (TANO) stores moments on active coordinates, retains the history of surviving connections, and resets newly activated states. Contextually Modulated LoRA (CoMoLoRA) supplies input-adaptive residual directions through rank-space modulation. These components address connectivity, optimization, and the capacity lost under a sparse constraint. In the completed two-seed LLaMA main study, CoMoLoRA reduces mean perplexity in all twelve auxiliary/no-auxiliary configuration pairs, with larger gains at 1:4 than at 2:4. At 350M and 1:4, the gains are 2.124 and 2.423 perplexity points for row-wise and block-wise supports. Matched 130M controls show complementary benefits from TANO and residual compensation, while packed FP32 moments reduce eligible-matrix state storage by 75%. These results support combining effective optimization, compact optimizer state, and residual compensation as a practical route toward high-quality 1:4 sparse pretraining.

1 INTRODUCTION

Semi-structured sparsity imposes a local constraint on a neural network's connectivity: only N coordinates are active in each group of M. The familiar 2:4 format retains half of the targeted weights, while 1:4 retains one quarter. These constraints make the pattern regular enough to be useful for specialized execution, but they also restrict which features can interact within each local group (Mishra et al., 2021; Hubara et al., 2021). Learning a useful sparse model therefore requires more than selecting a mask at the end of dense training.

Dynamic sparse training makes the support itself part of learning. In the Cannistraci–Hebb family, topology is updated using connectivity information while the executed backbone remains sparse (Zhang et al., 2025a; Lyu et al., 2026). A refresh changes not only the function computed by a layer, but also the meaning of its optimizer state: some coordinates survive, some disappear, and some enter without an active optimization history. At the same time, a highly constrained backbone may lack directions that remain useful for particular inputs. These observations motivate two practical additions to a dynamic sparse backbone: an update rule with explicit state handling, and a small residual branch that supplies input-dependent capacity.

We develop CHTsNM by combining established Cannistraci–Hebb topology evolution with two additions for N:M pretraining. The *Topology-Aware Nesterov Optimizer* (TANO) applies preconditioning before Nesterov momentum and stores FP32 moments only for active connections. Its coordinate-based remapping retains surviving history and resets new states after each refresh. *Contextually Modulated LoRA* (CoMoLoRA) adds a small low-rank residual whose rank-space coefficients depend on the input. These components have distinct roles and can be evaluated separately.

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.

The empirical case rests on those distinct roles. At 130M, replacing Adam with TANO improves sparse-backbone quality; adding CoMoLoRA improves quality with either optimizer. A separate comparison keeps the update rule fixed and finds equal final PPL with dense and packed FP32 moments, while packing removes inactive state storage. Each component is assessed against its own quality or storage objective.

Our contributions are threefold. First, we specify TANO's active-coordinate update, state-remapping rule, and physical storage layout for row-wise and transposable block-wise N:M supports. Second, we introduce contextually modulated low-rank compensation and evaluate its effect alongside ordinary additive and alternative branches across auxiliary budgets. Third, we provide multi-seed evidence across model scales and sparsity layouts, paired component controls, architecture/context/horizon extensions, and platform-specific inference measurements and compiler-based estimates. Together, these studies establish the quality and storage benefits that remain useful as the backbone moves from 2:4 to 1:4 sparsity.

2 Related work

Learning semi-structured sparsity. STE-style approaches propagate surrogate gradients through discrete projections; SR-STE and S-STE improve this approach for N:M training (Bengio et al., 2013; Zhou et al., 2021; Hu et al., 2024). CAST studies a continuous sparsity-aware transition (Huang et al., 2025). CHTs and CHTs24 instead learn connectivity through sparse topology evolution (Zhang et al., 2025a; Lyu et al., 2026). CHTsNM builds on that topology mechanism and studies its interaction with optimization history and contextually modulated residual capacity. We distinguish the CAST-style AdamS pretraining adaptation used in some controls from the complete CAST retraining pipeline.

Optimization under changing support. Separating preconditioning from momentum is established in LaProp, and Nesterov-style adaptive updates have long been studied (Ziyin et al., 2020; Dozat, 2016). STEP emphasizes second-moment estimation when learning N:M masks (Lu et al., 2023); SMET studies regrowth cold starts, optimizer warm-up, density-aware learning rates, and active-only state storage (Xiao et al., 2026). These are direct antecedents of the issues considered here. TANO combines a particular preconditioned Nesterov rule with coordinate-preserving remapping under legal N:M refreshes. We make no claim of priority for its individual ingredients. Matrix optimizers such as Muon (Liu et al., 2025) motivate alternative update geometry, but matrix orthogonalization is not part of TANO.

Sparse and low-rank models. LoRA supplies a low-rank residual parameterization (Hu et al., 2022). Sparse-plus-low-rank pretraining and compensation have been explored by SLTrain, SLoPE, and CHTsL (Han et al., 2024; Mozaffari et al., 2025; Wu et al., 2026). Contextual modulation provides a related way to make residual behavior input dependent (Zhang et al., 2025b). CoMoLoRA modulates the low-rank coordinates before the output projection. Its purpose here is contextually modulated compensation during sparse pretraining, and its dense auxiliary compute is included in the method definition.

3 CHTsNM: dynamic sparsity, optimization, and compensation

3.1 Sparse backbone and contextually modulated residual

For a target weight matrix $W_t \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, let M_t be a binary legal N:M mask and $S_t = \text{supp}(M_t)$. A row-wise mask has exactly N active entries in each contiguous length-M row group. A block-wise mask additionally requires exactly N active entries in every column of each $M \times M$ block, making the local pattern transposable. We use $M = 4$ and $N \in \{1, 2\}$. The constraint applies to the attention and MLP projection matrices, not to embeddings, normalization parameters, or the auxiliary branch.

For a row-vector input x, the layer computes

$$y = x(W_t \odot M_t)^\top + \frac{\alpha}{r}\left[c_\theta(x) \odot (xA^\top)\right]B^\top, \tag{1}$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$, and $c_\theta(x) \in \mathbb{R}^r$. Omitting the second term gives the no-compensation variant. The backbone obeys N:M throughout execution; the complete layer is a sparse backbone plus a dense contextually modulated residual, so its effective function is not itself an N:M matrix.

3.2 Dynamic sparse backbone

We retain the CHTs24 topology mechanism: soft magnitude-based removal selects connections for replacement, and Cannistraci–Hebb connectivity scores guide legal regrowth (Lyu et al., 2026). Block-wise masks use transposable N:M motifs. For row-wise masks, motif pattern revisitation (MPR) optionally extends this refresh rule by restricting reselection of a group's previous motif; its definition and ablation are given in Appendix B.2.

Weight values at a refresh. State initialization and weight initialization are separate operations. With S^- and S^+ denoting the old and new active supports, the refreshed weights satisfy

$$W_i^+ = \begin{cases} W_i^-, & i \in S^- \cap S^+, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Thus genuinely new coordinates start at zero, and surviving coordinates retain their learned values. An inactive coordinate that reappears does not recover its former weight. The dense residual branch is unchanged by this operation. Appendix B.1 gives the initialization and refresh-budget details.

3.3 TANO: coordinate-consistent active-state optimization

State follows coordinates, not packed slots. Each sparse matrix stores its first and second moments only for active coordinates. When the support changes from S^- to S^+, let $K = S^- \cap S^+$ denote survivors and $G = S^+ \setminus S^-$ denote newly activated coordinates. Before the next update, TANO sets

$$(m_i^+, v_i^+) = \begin{cases} (m_i^-, v_i^-), & i \in K, \\ (0, 0), & i \in G, \end{cases} \qquad i \in S^+. \tag{3}$$

States on $S^- \setminus S^+$ are discarded. Matching uses each coordinate's position within its row group, so a surviving weight retains its history even if its packed slot changes. A coordinate that returns after being inactive is treated as newly activated. This defines a state policy; it does not guarantee that zero initialization eliminates cold-start transients.

For example, when a 2:4 group changes from active positions $[0, 2]$ to $[2, 3]$, coordinate 2 moves from packed slot 1 to slot 0. Its moments move with it; the new slot for coordinate 3 starts at zero. Copying buffers by slot instead would give coordinate 2 the history of coordinate 0. This distinction lets the optimizer preserve surviving history while storing only the currently active support. Algorithm 1 gives the complete interface.

Preconditioning before Nesterov momentum. For active gradient $g_{t,i}$, with remapping applied whenever needed, the update is

$$v_{t,i} = \beta_2 v_{t-1,i} + (1-\beta_2) g_{t,i}^2, \qquad q_{t,i} = \frac{g_{t,i}}{\sqrt{v_{t,i}} + \epsilon}, \tag{4}$$

$$m_{t,i} = \beta_1 m_{t-1,i} + (1-\beta_1) q_{t,i}, \qquad u_{t,i} = (1-\beta_1) q_{t,i} + \beta_1 m_{t,i}, \tag{5}$$

$$W_{t+1,i} = W_{t,i} - \eta_t s(W) u_{t,i}, \qquad s(W) = \sqrt{d_{\text{out}}/d_{\text{in}}}. \tag{6}$$

The implemented recipe has no moment bias correction, no Newton–Schulz branch, and zero weight decay. Moments use FP32; updates are cast to the BF16 parameter dtype, without an FP32 master weight copy. Untargeted dense parameters use dense moments and the same arithmetic, with $s = 1$ for vectors. CoMoLoRA parameters use their own Adam optimizer. These precision and scaling choices belong to the reported recipe and must be controlled in optimizer comparisons.

Storage and the execution boundary. If a targeted matrix has $D = d_{\text{out}} d_{\text{in}}$ coordinates and density $\delta = N/M$, its two FP32 moments occupy $8\delta D$ bytes rather than $8D$. The current implementation adds δD bytes of uint8 position metadata for $M = 4$. Thus moment storage alone

Algorithm 1 TANO active-state interface for one sparse matrix

Require: Legal mask M, packed coordinate indices I, FP32 moments m, v

On initialization or notification of a new support M':

Validate N active entries per row group; for block layout validate columns too
$I' \leftarrow$ sorted active positions within each row group of M'
Allocate zero FP32 buffers m', v' with shape I'
for each group g and active position $I'_{g,j}$ **do**
    if $I'_{g,j}$ occurs at old position $I_{g,k}$ **then**
        $m'_{g,j} \leftarrow m_{g,k}$; $v'_{g,j} \leftarrow v_{g,k}$
    end if
end for
$(I, m, v, M) \leftarrow (I', m', v', M')$; discard removed states

On an eligible optimizer update:

Reject nonfinite gradients; synchronize any pending support notification
Gather active gradient g in the order of I and convert to FP32
$v \leftarrow \beta_2 v + (1-\beta_2)g^2$; $q \leftarrow g/(\sqrt{v}+\epsilon)$
$m \leftarrow \beta_1 m + (1-\beta_1)q$; $u \leftarrow (1-\beta_1)q + \beta_1 m$
Cast u to parameter dtype; update active weights with $-\eta_t s(W)u$
Leave inactive coordinates outside the optimizer update

scales by δ, whereas moments plus indices scale by $9\delta/8$ relative to dense FP32 moments. Dense non-target state, model parameters, gradients, activations, and CoMoLoRA are additional costs. In particular, weights and gradients currently remain dense tensors: active-only optimizer storage does not imply an equal reduction in peak training memory or sparse-kernel execution.

3.4 CoMoLoRA: contextually modulated low-rank compensation

The gate in Equation 1 is

$$c_\theta(x) = \mathbf{1} + \gamma \tanh G_\theta(x), \qquad G_\theta(x) = \mathrm{SiLU}(xU_1^\top + b_1)U_2^\top + b_2. \tag{7}$$

Here $U_1 \in \mathbb{R}^{h \times d_{\text{in}}}$ and $U_2 \in \mathbb{R}^{r \times h}$. The final controller layer is zero-initialized, so $c_\theta(x) = \mathbf{1}$ initially and the branch begins as ordinary LoRA. The main configuration uses $r = 7$, $h = 2$, $\gamma = 0.5$, and $\alpha = 18\frac{2}{3}$. The gate lies in $(0.5, 1.5)$ per rank coordinate, allowing a bounded modulation around the additive low-rank branch.

The placement of this gate is consequential. Each column of B specifies a shared output direction, while $c_\theta(x)$ changes how strongly the input uses that direction. The branch can therefore supply compensation that depends on the input without predicting a full dense matrix. It cannot generally be merged into a fixed weight matrix, because the modulation depends on x. The approximately 1% auxiliary budget includes the controller as well as the low-rank factors; it is not a claim that the complete model has the backbone's sparsity ratio.

3.5 Roles of the components

CHTsNM builds on the CHTs24 topology engine and adds an active-state optimizer and an optional compensation branch. TANO changes the backbone update and state representation; CoMoLoRA adds input-dependent residual capacity. These changes can be enabled separately. We therefore evaluate their individual and combined effects, and separate improvements in final PPL from reductions in optimizer-state storage (Section 4.2).

4 Experiments

We study sparse language-model pretraining on OpenWebText (Gokaslan & Cohen, 2019), followed by architecture, context-length, and training-horizon comparisons. Model sizes and training budgets accompany the tables; full recipes are in Appendix A. Baseline hyperparameters are inherited from the published CHTs24 comparisons or the respective original papers, without additional per-method tuning for these comparisons.

Table 1: **Sparse pretraining across scales.** LLaMA models on OpenWebText; PPL (↓). CoMoLoRA uses a nominal 1% auxiliary budget.

LLaMA model size				60M	130M	350M
Training updates				10K	10K	20K
Sparsity		Method	Aux.	PPL	PPL	PPL
Dense		Adam	0%	32.63 ± 0.09	26.58 ± 0.05	20.39 ± 0.11
1:4	Row-wise	STE	0%	45.18 ± 0.09	34.97 ± 0.07	27.77 ± 0.42
		SR-STE	0%	42.81 ± 0.07	34.45 ± 0.26	26.75 ± 0.01
		CAST-style	0%	44.96 ± 0.22	35.83 ± 0.05	28.73 ± 1.28
		CHTs24	0%	42.13 ± 0.26	34.35 ± 0.08	27.34 ± 0.24
		CHTsNM	0%	34.49 ± 0.18	30.70 ± 0.62	24.57 ± 0.24
		CHTsNM	1%	$\mathbf{32.91 \pm 0.01}$	$\mathbf{28.18 \pm 0.46}$	$\mathbf{22.45 \pm 0.41}$
	Block-wise	STE	0%	53.91 ± 0.17	40.15 ± 0.17	31.27 ± 0.10
		SR-STE	0%	44.81 ± 0.08	35.92 ± 0.09	26.56 ± 0.20
		CHTs24	0%	42.85 ± 0.12	34.74 ± 0.04	27.22 ± 0.08
		CHTsNM	0%	35.33 ± 0.31	30.61 ± 0.28	24.92 ± 1.42
		CHTsNM	1%	$\mathbf{33.66 \pm 0.00}$	$\mathbf{28.41 \pm 0.23}$	$\mathbf{22.49 \pm 0.16}$
2:4	Row-wise	STE	0%	37.28 ± 0.15	29.72 ± 0.03	23.24 ± 0.31
		SR-STE	0%	36.39 ± 0.06	29.65 ± 0.05	22.98 ± 0.31
		CAST-style	0%	37.29 ± 0.13	30.04 ± 0.01	22.96 ± 0.35
		CHTs24	0%	36.61 ± 0.04	29.98 ± 0.05	23.24 ± 0.30
		S-STE	0%	38.36 ± 0.11	30.53 ± 0.05	24.31 ± 0.64
		CHTsNM	0%	31.00 ± 0.42	26.43 ± 0.05	21.04 ± 0.02
		CHTsNM	1%	$\mathbf{30.48 \pm 0.78}$	$\mathbf{26.08 \pm 0.36}$	$\mathbf{20.25 \pm 0.06}$
	Block-wise	STE	0%	40.81 ± 0.13	31.83 ± 0.04	24.58 ± 0.35
		SR-STE	0%	37.26 ± 0.05	30.14 ± 0.10	23.15 ± 0.05
		CHTs24	0%	37.15 ± 0.01	30.12 ± 0.04	23.14 ± 0.09
		CHTsNM	0%	31.95 ± 1.18	26.51 ± 0.40	20.79 ± 0.11
		CHTsNM	1%	$\mathbf{30.55 \pm 0.51}$	$\mathbf{25.66 \pm 0.18}$	$\mathbf{20.26 \pm 0.07}$

S-STE: three runs at 60M/130M, two at 350M. CHTsNM: two runs per cell; bold compares with and without CoMoLoRA. Training and evaluation details: Appendix A.

4.1 Main results: compensation under 75% sparsity

Consistent gains across layouts and scales. CoMoLoRA improves PPL in all twelve comparisons with the same sparse backbone without compensation (Table 1). Under 1:4 sparsity, the reduction ranges from 4.6% to 9.7%. The benefit holds for both row-wise supports and transposable blocks, showing that residual compensation remains useful under the additional column constraints.

Compensation matters more at 1:4. At each model size and layout, the relative PPL reduction is larger under 1:4 than under 2:4. At 350M, the row-wise and block-wise reductions are 8.6% and 9.7% at 1:4, compared with 3.8% and 2.6% at 2:4. This pattern connects the value of residual capacity to the severity of the local sparsity constraint. Table 1 reports all paired results.

4.2 Ablation: optimization and compensation

TANO and residual compensation provide complementary gains. Table 2 isolates their contributions on LLaMA-130M with the same initial backbone, sparse mask, and first input batch within each of two repetitions. Replacing Adam with TANO reduces PPL by 5.0%; adding CoMoLoRA to Adam reduces it by 8.7%; combining both reduces it by 12.4%. The combined recipe performs best among the four settings. Thus improving the update rule does not remove the benefit of residual capacity, and adding capacity does not remove the benefit of the update rule. Appendix A specifies the paired protocol.

Table 2: **Optimization and residual compensation: paired ablation.** LLaMA-130M, row-wise 1:4, OpenWebText, 10K updates; PPL (↓).

CoMoLoRA	TANO	PPL	Reduction vs. Adam
×	×	32.32 ± 0.07	–
×	✓	30.70 ± 0.62	5.0%
✓	×	29.52 ± 0.11	8.7%
✓	✓	$\mathbf{28.31 \pm 0.27}$	12.4%

Two matched repetitions per setting, with the same initial backbone, sparse mask, and data within each repetition. This dedicated paired study is distinct from the main sweep.

Table 3: **Optimizers on a dynamic sparse backbone.** LLaMA-130M, OpenWebText, 10K updates, no auxiliary branch; PPL (↓).

Optimizer	Row 2:4	Row 1:4
Adam	29.78 ± 0.12	32.40 ± 0.16
AdamS	29.83 ± 0.30	34.55 ± 0.07
SparseAdam	29.99 ± 0.02	35.77 ± 0.41
LaProp	27.92 ± 0.03	32.77 ± 0.46
SMET-style	29.40 ± 0.04	30.16 ± 0.27
TANO	26.43 ± 0.05	30.70 ± 0.62

Adam at row-wise 1:4 uses three runs; all other cells use two. Implementation and evaluation details: Appendix A.2.

4.3 TANO: UPDATE RULE AND STATE STORAGE

Update rule. On the 130M row-wise 1:4 backbone, TANO reduces PPL by 5.3% relative to Adam and by 6.3% relative to LaProp. SMET-style achieves the lowest mean in this comparison. AdamS and SparseAdam provide additional sparse-optimizer references in Table 3; their implementations and evaluation settings are described in Appendix A.2.

Table 4: **Physical optimizer-state storage.** LLaMA-130M, row-wise 1:4, no auxiliary branch. Both layouts use FP32 moments and the same reset update.

State component (MiB)	Dense layout	TANO packed layout
Eligible-matrix moments	648.00	162.00
Other-parameter moments	376.32	376.32
Coordinate metadata	81.00	20.25
Total optimizer state	1,105.32	558.57
Final PPL (two matched runs)	30.70 ± 0.62	30.70 ± 0.62

Dense metadata stores the previous Boolean mask; packed metadata stores uint8 active positions. Model weights, gradients, and activations are excluded.

Storing moments only for active connections. TANO allocates its two FP32 moment buffers only to active coordinates. Table 4 separates the resulting 75% reduction on eligible matrices from the unchanged moments on other parameters. Including coordinate metadata, total optimizer-state storage decreases by 49.5%. The reduction comes from allocating shorter buffers, rather than masking values inside dense buffers.

The paired dense-reset and packed implementations reach identical recorded final PPL in both repetitions. This comparison holds the update arithmetic and state-reset policy fixed, so it isolates the storage representation from the optimizer choice. Retaining FP32 moments while removing inactive entries therefore saves state memory without changing these measured endpoints. Appendix B.6 explains the storage accounting and auxiliary-optimizer costs.

Table 5: **Auxiliary-branch design with a shared Adam backbone.** LLaMA-60M, block-wise 1:4 CHTs24, OpenWebText, 10K updates; PPL (↓).

Budget	1%	2%	3%	4%	5%
WIDEN	42.21 ± 0.12	41.84 ± 0.27	41.53 ± 0.15	40.92 ± 0.26	40.5 ± 0.12
ADD	39.73 ± 0.02	38.74 ± 0.18	38.44 ± 0.31	37.58 ± 0.26	37.15 ± 0.03
ADD+ALIGN	$\mathbf{39.61 \pm 0.07}$	38.75 ± 0.18	38.34 ± 0.14	37.51 ± 0.05	37.28 ± 0.07
PROD-GATE	41.86 ± 0.49	41.2 ± 0.08	41.13 ± 0.35	40.87 ± 0.38	40.31 ± 0.13
COMOLORA	39.72 ± 0.22	$\mathbf{38.69 \pm 0.07}$	$\mathbf{37.77 \pm 0.3}$	$\mathbf{37.45 \pm 0.01}$	$\mathbf{36.85 \pm 0.06}$

All variants use Adam to isolate branch design from the proposed optimizer. Original reported means and dispersions are retained. Widen uses an active-parameter-equivalent budget; the other designs use auxiliary-parameter budgets.

4.4 AUXILIARY-BRANCH DESIGN

We compare widening the sparse backbone, additive low-rank compensation (ADD), alignment-regularized compensation (ADD+ALIGN), product-only modulation (PROD-GATE), and CoMoLoRA. All variants in Table 5 use Adam, separating the branch-design comparison from the proposed optimizer.

ADD+ALIGN has the lowest reported mean at 1%, while CoMoLoRA is best from 2% to 5%. Its reduction relative to ADD reaches 1.7%, and its advantage over widening reaches 9.1%. These results support contextual modulation when the auxiliary budget can support its controller. This branch-design comparison is specific to the shared Adam recipe.

4.5 BILLION-PARAMETER PRETRAINING AND GENERALIZATION

Table 6: **Billion-parameter pretraining.** LLaMA-1B, OpenWebText, 50K updates (∼3.2B tokens); PPL (↓).

Method	Layout	Aux.	Seeds	OWT PPL
Dense (Adam)	Dense	0%	2	14.798 ± 0.424
SR-STE	Block 2:4	0%	2	15.811 ± 0.236
SLoPE + Lazy-LoRA[‡]	Row 2:4	∼1%	2	19.022 ± 0.023
CHTsNM (TANO)	Block 2:4	0%	2	15.424 ± 0.043

[‡]Rank 14, two completed runs; the last 500 updates include Lazy-LoRA. Training details: Appendix C.4.

Scaling to 1B. Table 6 extends pretraining to a billion-parameter backbone. It includes two TANO block-wise 2:4 runs and two SLoPE + Lazy-LoRA runs. Appendix C.4 specifies the training budgets and branch configuration.

Table 7: **Sparse pretraining on GPT-NeoX-134M.** OpenWebText, 10K updates, context 256; two runs per cell. All sparse layouts are row-wise. PPL (↓).

Format	Method	GPT-NeoX-134M
Dense	Adam	27.853 ± 0.092
2:4	SR-STE	32.176 ± 0.008
2:4	CHTsNM (TANO)	27.061 ± 0.261
1:4	SR-STE	37.105 ± 0.091
1:4	CHTsNM (TANO)	31.059 ± 0.858
1:4	CHTsNM + CoMoLoRA (TANO)	30.053 ± 0.123

All entries use two repetitions. Training and evaluation details: Appendix A.

Transfer to GPT-NeoX. Table 7 compares methods on GPT-NeoX-134M at equal N:M formats. CHTsNM has lower reported mean PPL than the SR-STE reference at both sparsity levels. At 1:4,

CoMoLoRA further improves the TANO backbone by 3.2%, consistently across repetitions. These results extend the LLaMA study to GPT-NeoX (Appendix C.1).

Table 8: **7B retraining after pruning.** LLaMA2-7B, row-wise 2:4; PPL (↓).

Method / export	Format	Steps	FP32 master weight	WT2 PPL	OWT PPL
Original LLaMA2-7B	Dense	–	–	4.861	6.552
SparseGPT Pruning	2:4	–	–	12.895	16.376
CHTsNM + KD	2:4	3,815	no	8.237	8.970
CAST + KD, hard export	2:4	3,815	yes	18.061	27.988

CHTsNM and CAST are retrained on 500.04M tokens using the same teacher and knowledge-distillation objective. To address instability observed in CAST with BF16-only weight updates, we enable FP32 master weights while retaining BF16 computation. CAST maintains dense weights during retraining and applies a final 2:4 pruning step. Its WikiText-2 and OpenWebText PPLs increase from 5.122 and 6.312 before pruning to 18.061 and 27.988 afterward. This degradation may reflect an insufficient adaptation period: although CAST's AdamS optimizer progressively drives selected weights toward zero, the limited retraining budget may leave the model sensitive to their removal at final pruning.

Recovery after pruning at 7B. Table 8 evaluates LLaMA2-7B(Touvron et al., 2023) recovery after SparseGPT(Frantar & Alistarh, 2023) initialization. CHTsNM + KD reduces PPL relative to CAST + KD by 54.4% on WikiText-2 and 67.9% on OpenWebText. This extends the framework to retraining large pretrained models with distillation. Appendix C.5 details the training budgets and evaluation.

Longer contexts and longer training. CoMoLoRA improves all eight context/layout combinations from 128 to 2,048 tokens. At 2,048 tokens, the gains are 6.3% at 1:4 and 5.9% at 2:4, comparing methods at the same context and update budget (Appendix C.2). The gain persists over 30K updates, reaching 1.3% at row-wise 2:4 (Appendix C.3). These comparisons test compensation at fixed context lengths and training horizons.

4.6 Hardware acceleration

Table 9: **Inference acceleration on two hardware platforms.** Speedups use each platform's own dense baseline.

(a) NVIDIA RTX 5080: 2:4

Measured sparse-backbone inference

Model	Speedup	Energy saving
60M	1.197×	18.63%
130M	1.320×	25.28%
250M	1.391×	29.85%
350M	1.524×	39.83%
1B	1.652×	40.33%

(b) MOFFET: 1:4

Compiler-scheduled inference

Model	Length	Backbone	+ CoMoLoRA
130M	128	1.81×	1.32×
130M	256	2.25×	1.69×
350M	128	1.98×	1.43×
350M	256	2.41×	1.85×
1B	128	2.21×	1.89×
1B	256	2.67×	2.40×

(a) Batch 8, length 256; five counterbalanced blocks; GPU-board energy per full-forward token. (b) BF16, 700 MHz, four data-parallel cores with batch 1 each; last-token LM-head, I/O and DMA scheduling included. Panel (b) reports compiler estimates, not device measurements. Workloads differ across panels; ratios are within-platform comparisons.

Table 9 presents platform-specific benefits of 2:4 and 1:4 sparsity. NVIDIA native 2:4 inference reaches 1.65× measured throughput and 40.3% lower board energy at 1B. On MOFFET, compiler-scheduled 1:4 execution gives 1.81–2.67× backbone speedups and 1.32–2.40× with CoMoLoRA. The complete hybrid is also 1.28–1.69× faster than its 2:4 counterpart, showing that higher sparsity can accommodate residual compensation while retaining a latency benefit. Appendices D.1 and D.2 give the respective measurement and compilation protocols.

5 Discussion and limitations

Optimization and residual compensation provide complementary quality gains, while active-state packing reduces moment storage. The branch-design comparison uses a shared Adam backbone, so its conclusions are specific to the tested budgets and optimization recipe. Training still uses dense parameter tensors; activation memory and full-model serving costs require separate measurement. NVIDIA measurements and MOFFET compiler estimates characterize their respective inference workloads.

6 Conclusion

CHTsNM provides a sparse-to-sparse training framework for moving from 2:4 to 1:4 semi-structured sparsity while preserving useful model capacity. It retains Cannistraci–Hebb topology evolution and introduces two additions with distinct roles: TANO updates the sparse backbone with coordinate-consistent active-state storage, and CoMoLoRA supplies contextually modulated low-rank residual capacity. The matched component study shows that optimization and compensation each improve quality and that their combination performs best among the four tested settings. Separately, physically packing FP32 moments reduces eligible-matrix state storage by 75% at 1:4, with unchanged final PPL in the matched storage comparison.

Across the main LLaMA study, compensation improves all twelve layout-and-scale comparisons, with larger gains at 1:4 than at 2:4. GPT-NeoX, longer-context, and longer-training results extend this evidence beyond the main setting. Native NVIDIA 2:4 measurements and MOFFET 1:4 compiler estimates further connect the sparse representations to their platform-specific execution potential, including the cost of compensation in the latter. These findings support combining effective sparse-backbone optimization, compact optimizer state, and residual capacity as a practical route toward 75% backbone sparsity. Further work should evaluate the full training and serving pipeline with native 1:4 kernels and extend the study to larger models and more diverse training corpora.

Reproducibility statement

Appendix A specifies the training, evaluation, and statistical protocols. Appendix B details topology refinement, state remapping, and complexity, and Appendices C–D provide supplementary comparisons and measurement settings. Code, configurations, and per-run results accompany the paper.

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 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.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Timothy Dozat. Incorporating nesterov momentum into adam. *International Conference on Learning Representations*, 2016.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pp. 10323–10337. PMLR, 2023.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Jawanpuria, and Bamdev Mishra. Sltrain: a sparse plus low rank approach for parameter and memory efficient pretraining. *Advances in Neural Information Processing Systems*, 37:118267–118295, 2024.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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, 2024.

Weiyu Huang, Yuezhou Hu, Jun Zhu, and Jianfei Chen. Cast: Continuous and differentiable semi-structured sparsity-aware training for large language models. *arXiv preprint arXiv:2509.25996*, 2025.

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.

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.

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.

Jiaqing Lyu, Ruijie Wang, Kangyou Bao, Yingtao Zhang, and Carlo Vittorio Cannistraci. Cannistraci-hebb training with n: M semi-structured sparsity for pre-training and re-training. *Conference on Parsimony and Learning*, 2026.

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.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models, 2023. *URL https://arxiv. org/abs/2307.09288*, 2307, 2023.

Wenjing Wu, Yingtao Zhang, Jialin Zhao, and Carlo Vittorio Cannistraci. Alignment-enhanced integration of connectivity and spectral sparsity in dynamic sparse training of LLM. *International Conference on Learning Representations*, 2026.

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.

Yingtao Zhang, Diego Cerretti, Jialin Zhao, Wenjing Wu, Ziheng Liao, Umberto Michieli, and Carlo Vittorio Cannistraci. Brain network science modelling of sparse neural networks enables transformers and LLMs to perform as fully connected. *Advances in Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=OM0Qkq9xtY.

Yingtao Zhang, Wenqi Gu, Wen Hu, Jianguo Li, and Carlo Vittorio Cannistraci. From transformer to transponder: Introducing contextual modulation training for residual learning in llms. *Preprints*, September 2025b. doi: 10.20944/preprints202506.0120.v2. URL https://doi.org/10.20944/preprints202506.0120.v2.

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. *International Conference on Learning Representations*, 2021.

Liu Ziyin, Zhikang T Wang, and Masahito Ueda. Laprop: Separating momentum and adaptivity in adam. *arXiv preprint arXiv:2002.04839*, 2020.

A EXPERIMENTAL SETTINGS AND EVALUATION

A.1 PRETRAINING SETTINGS

The main study trains LLaMA-style models on OpenWebText with causal language modeling. Table 10 specifies the shared recipe. The sparse constraint applies to attention and MLP projection matrices; embeddings, normalization parameters, and auxiliary branches remain dense. The main CoMoLoRA experiments use no alignment penalty or distillation loss. Model size, N:M layout, and auxiliary-branch inclusion are varied as indicated in the main tables.

Table 10: Main pretraining settings for CHTsNM with TANO.

Item	Setting
Data and model	OpenWebText; LLaMA-style 60M/130M/350M
Tokenizer and context	Main-suite 32K vocabulary; 256 tokens
Split and order	Document split, test fraction 0.05, split seed = run seed; shuffle seed 42
Global batch	256 sequences
60M/130M schedule	10,000 updates; 1,000 warm-up updates; microbatch 64, accumulation 4
350M schedule	20,000 updates; 2,000 warm-up updates; microbatch 32, accumulation 8
Learning rate	10^{-3}; cosine restarts; minimum ratio 0.1; shape factor in Equation 6
TANO arithmetic	$\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-5}$; no bias correction; weight decay 0
Precision	BF16 parameters, FP32 TANO moments, no FP32 master copy
Target matrices	q/k/v/o and gate/up/down projections
Topology	CH2-L3n-soft regrowth; soft magnitude removal; adaptive $\zeta = 0.1$
Refresh schedule	Interval 100; iterative warm-up 10; topology freeze ratio 0.75
Row-wise MPR	$\rho = 0.75$ for 2:4; $\rho = 0.5$ for 1:4
CoMoLoRA	$r = 7$, $h = 2$, $\alpha = 18\frac{2}{3}$, rank gate $\gamma = 0.5$; no output-channel gate
Auxiliary optimizer	Separate Adam; BF16 state, matching the canonical auxiliary recipe
Randomness	Main exact-evaluation model/data/mask seeds 1 and 2
Final evaluation	Exactly 100M valid causal targets; FP32 CE, FP64 accumulation

A.2 BASELINES AND CONTROLLED COMPARISONS

The optimizer comparison in Table 3 uses a dynamic sparse backbone without an auxiliary branch. AdamS applies selective decay to pruned coordinates, whereas SparseAdam restricts moment updates to active coordinates. LaProp and SMET-style are adaptations within the same sparse training implementation. Their quality comparisons do not measure native sparse-kernel efficiency.

Hyperparameter provenance. Baselines already covered by the published CHTs24 comparison inherit those comparison configurations; other baselines inherit the defaults reported in their original papers. We do not conduct an additional method-specific learning-rate or optimizer-hyperparameter search for the comparisons reported here. Within the controlled optimizer study, shared training settings are fixed as in Table 10, while the defining update rules of each optimizer are retained. In particular, the sparse LaProp and SMET-style implementations are adaptations to this shared recipe, not reproductions of every setting in the original papers. Auxiliary-budget sweeps change the architecture budget, and component ablations change the stated mechanism; neither is a search used to select a separate best training recipe for each table entry.

For the branch comparison in Table 5, all backbones use Adam. WIDEN increases FFN width; ADD supplies an ordinary low-rank residual; ADD+ALIGN adds an alignment penalty with coefficient 0.3; and PROD-GATE modulates the sparse output without an additive residual. CoMoLoRA instead

modulates the low-rank coordinates. Widening matches active-parameter-equivalent capacity, while the other variants use auxiliary-parameter budgets. These budget definitions are kept distinct.

The matched 130M component comparison in Table 2 uses the common seeds 1 and 2. Within each repetition, the initial backbone, sparse mask, and first input batch agree across the four variants. This isolates the changes in optimizer and residual branch.

B METHOD AND COMPLEXITY DETAILS

B.1 WEIGHT INITIALIZATION AND REFRESH BUDGET

The LLaMA backbone is initialized using the model's Gaussian initializer (standard deviation 0.02), followed by a sampled legal N:M mask. Inactive weights are zeroed. During training, inactive weights receive no optimizer update. A topology refresh changes the mask and applies Equation 2: surviving values are kept, removed values are set to zero, and newly active values remain zero until subsequent gradient updates. No random reinitialization or historical weight recovery is used.

Refreshes occur every 100 update slots until $T_f = 0.75T$, where T is the training horizon. For a matrix with A_l active connections, the requested removal count at slot t is

$$\zeta_t = \frac{0.1}{2}\left[1 + \cos\left(\frac{\pi t}{T_f}\right)\right], \qquad k_l(t) = \lfloor A_l \zeta_t \rfloor. \tag{8}$$

The removal fraction is thus initially close to 10% of active connections, 5% halfway to the topology-freeze point, and zero at that point. For 10K-slot runs, the first refresh is at slot 100 and the freeze point is 7,500; for 20K-slot runs the freeze point is 15,000. Refresh slots skip the weight update, while synchronizing the mask and optimizer state; iterative learning-rate warm-up spans ten slots after refresh.

The requested count is distinct from realized turnover after legal regrowth. A removed connection may be selected again, and block-wise motif completion may change which coordinates survive. The realized replacement fraction is $|S^+ \setminus S^-|/|S^-|$, or half the support Hamming distance divided by $|S^-|$ when density is fixed. MPR acts on affected row groups with probability ρ, not on a fraction ρ of all weights. Equation 8 specifies the implemented schedule; per-refresh realized-turnover traces were not retained for the adopted main runs, so these schedule values are not reported as measured turnover.

B.2 OPTIONAL ROW-WISE MOTIF REVISITATION

MPR extends the legal regrowth rule used by the CHTs24 backbone; it is an optional topology refinement rather than a separate optimizer or compensation mechanism. For row-wise groups, define the legal pattern set $\mathcal{R}_N = \{p \in \{0,1\}^4 : \sum_i p_i = N\}$. A group selected for refresh may be assigned its previous pattern again. MPR changes the candidate set to

$$\mathcal{C}_g = \begin{cases} \mathcal{R}_N \setminus \{p_g^{\text{old}}\}, & \text{with probability } \rho, \\ \mathcal{R}_N, & \text{otherwise.} \end{cases} \tag{9}$$

The existing score-guided regrowth rule then chooses a legal pattern within $\mathcal{C}_g$. MPR acts only on groups already selected for refresh; ρ is not a global refresh fraction. We use $\rho = 0.75$ for row-wise 2:4 and 0.5 for row-wise 1:4. Block-wise updates select legal transposable motifs without applying this row-specific rule.

Standalone effect at 60M. Table 11 isolates the refinement on the CHTs24 backbone with Adam, without a compensation branch. At row-wise 1:4 and fixed initial $\zeta = 0.15$, constant $\rho = 0.5$ lowers mean PPL by 0.22%, with an improvement in both repetitions. At row-wise 2:4 and initial $\zeta = 0.10$, the best tested constant setting by mean PPL, $\rho = 0.75$, lowers mean PPL by 0.19%, also improving both repetitions. The intermediate setting $\rho = 0.5$ does not improve mean PPL at 2:4. Thus MPR provides a small, setting-dependent refinement, rather than a uniform gain from increasing motif turnover.

Table 11: Standalone MPR study: 60M CHTs24 with Adam and no auxiliary branch, OpenWebText, context 256, 10K updates. Each row uses the same two seeds. Positive reductions denote lower PPL relative to disabling MPR within the same layout and refresh budget.

Layout	Initial ζ	Constant ρ	PPL $\downarrow$	Reduction
Row 1:4	0.15	0	41.838 ± 0.039	–
Row 1:4	0.15	0.5	41.744 ± 0.125	+0.22%
Row 2:4	0.10	0	36.580 ± 0.107	–
Row 2:4	0.10	0.1	36.563 ± 0.044	+0.05%
Row 2:4	0.10	0.25	36.560 ± 0.052	+0.06%
Row 2:4	0.10	0.5	36.627 ± 0.094	-0.13%
Row 2:4	0.10	0.75	36.510 ± 0.035	+0.19%

Mean $\pm$ sample SD, $n = 2$. Adam uses $\beta_1 = 0$, $\beta_2 = 0.99$, learning rate 10^{-3}, and $\epsilon = 10^{-5}$. The two layout panels use different initial refresh fractions; comparisons are made within a panel.

B.3 Support changes and optimizer state

Algorithm 1 remaps states whenever the topology scheduler changes the support, including refreshes at which a weight update is skipped. A surviving coordinate retains its moments; removed coordinates release their state; newly active coordinates start from zero. A coordinate returning after an inactive interval is treated as newly active. The remapping policy is the same with and without CoMoLoRA.

At fixed support, gathering active coordinates, applying the same recurrence, and scattering the updates is a storage transformation of the dense-layout update. The paired dense-reset and packed implementations have equal recorded final PPL in both common repetitions. This provides a direct control for the storage representation; the comparison does not change the optimizer arithmetic or reset policy.

B.4 Parameter and computation budgets

For input width d_{in}, output width d_{out}, rank r, and controller width h, the CoMoLoRA branch has

$$P_{\text{branch}} = r(d_{\text{in}} + d_{\text{out}}) + hd_{\text{in}} + h + rh + r \tag{10}$$

parameters, including controller biases. Its leading matrix-vector work per token is proportional to $r(d_{\text{in}}+d_{\text{out}})+hd_{\text{in}}+hr$, followed by the nonlinearities and elementwise gate. Budget comparisons include the controller rather than counting only the low-rank factors.

Integer ranks introduce small differences between nominal and realized budgets. For example, the 3% branch sweep uses rank 20 for ADD and rank 22 for ADD+ALIGN.

B.5 Scope of the optimizer decomposition

The current comparisons distinguish changing Adam to the full TANO update from changing dense to packed state with fixed arithmetic and reset. LaProp and SMET-style provide additional optimizer comparisons. The experiments do not independently toggle preconditioning, Nesterov momentum, matrix-shape scaling, or bias correction inside the current TANO implementation. Consequently, the update-rule gain is attributed to the specified recipe as a whole; it is not assigned to any one of these arithmetic choices. All current TANO controls omit Newton–Schulz corrections.

B.6 Optimizer-state storage

For D eligible weights and active density $d = N/M$, two dense FP32 moment buffers occupy $8D$ bytes; packed buffers occupy $8dD$. Moment storage therefore falls by 75% at 1:4 and 50% at 2:4. At 130M and 1:4, this is 162MiB rather than 648MiB for the eligible matrices.

Table 4 includes non-sparse moments and coordinate metadata, giving a 49.5% reduction in total optimizer state. The corresponding TANO hybrid occupies 563.86MiB, compared with 558.57MiB

without the branch. The SMET-style adaptation occupies 1,429.32MiB under this implementation. These totals describe allocated optimizer tensors; weights, gradients, activations, and peak device memory are separate quantities.

Packing reduces storage without requiring a sparse matrix-multiplication kernel. Conversely, inference acceleration requires a supported execution format and includes the work of the dense residual branch. Appendix D reports that execution cost separately.

C GENERALIZATION AND SCALING

C.1 TRANSFER TO GPT-NEOX

The GPT-NeoX-134M comparison in Table 7 uses OpenWebText, context 256, 10K updates, and two training repetitions per configuration. All sparse layouts are row-wise. At 1:4, adding CoMoLoRA improves the backbone in both repetitions. This study tests transfer beyond the LLaMA architecture used in the main experiments.

C.2 CONTEXT LENGTH

Table 12: Context-length sweep with TANO on OpenWebText: 130M, 10K updates; two runs.

Format	Method	128	512	1,024	2,048
1:4	CHTsNM	33.293 ± 0.126	27.612 ± 0.412	27.308 ± 0.631	28.210 ± 0.001
1:4	CHTsNM + CoMo	31.267 ± 0.069	26.838 ± 0.649	25.961 ± 0.179	26.439 ± 0.188
2:4	CHTsNM	29.356 ± 0.168	25.374 ± 0.489	23.907 ± 0.444	25.059 ± 0.466
2:4	CHTsNM + CoMo	28.675 ± 0.127	24.657 ± 1.054	23.503 ± 0.097	23.588 ± 0.206

All runs use 10K updates and 256 sequences/update. L128 evaluates exactly 50M valid targets; other lengths use 100M. The 256-token main results use seeds 1/2 and appear in Table 1. Context length changes the processed token budget; compare methods within each length.

CoMoLoRA improves all eight format/context combinations in Table 12. At length 2,048, the reductions are 6.3% at 1:4 and 5.9% at 2:4. Each run uses 10K updates and a global batch of 256 sequences. Longer contexts therefore process more training tokens; the comparisons isolate compensation within a fixed length, rather than the effect of length at a fixed token budget.

C.3 LONGER TRAINING

Table 13: Longer-horizon 130M row-wise 2:4 pretraining with TANO.

Configuration	Updates	n	Final PPL
No branch	10K	2	26.431 ± 0.046
CoMoLoRA	10K	2	26.079 ± 0.355
No branch	30K	2	21.874 ± 0.107
CoMoLoRA	30K	2	21.591 ± 0.050

All values use exact 100M-token final evaluation. The 10K suite uses seeds 1/2 and its original schedule; the 30K suite uses seeds 0/1 and a 30K schedule (including its MPR setting). These are horizon-specific training recipes, not one continued learning curve.

The benefit persists at 30K updates: both repetitions improve, with a mean PPL reduction of 1.3%. The 10K and 30K entries use their respective full training schedules and seed sets. They show results at two training budgets, rather than checkpoints on a single continued learning curve.

C.4 BILLION-PARAMETER PRETRAINING

Table 6 uses 50K updates, approximately 3.2B training tokens. The two TANO block-wise 2:4 runs use no auxiliary branch and give 15.424 ± 0.043 under exact 100M-target evaluation. The SLoPE + Lazy-LoRA baseline uses rank 14, 13.077M branch parameters, and two pretraining seeds. Its

branch is activated at update 49,501, so the 500 Lazy-LoRA updates are included in the 50K budget. It uses a fixed data split with seed 0 and a trainable EOS token in packed sequences. The two merged exports give 19.022 ± 0.023 under exact 100M-target evaluation. PPL is evaluated on the merged model; the sparse-plus-low-rank factors are not assessed separately in this evaluation.

C.5 Retraining after pruning at 7B

The LLaMA2-7B experiments in Table 8 start from SparseGPT initialization and use knowledge distillation for CHTsNM and CAST. They measure recovery of a pretrained model, rather than training from scratch. CHTsNM and CAST process 500.04M tokens. Evaluation uses context 2,048, stride 512, the full WikiText-2 test split, and a fixed 10M-token OpenWebText set.

D Efficiency measurements

D.1 Native 2:4 inference on NVIDIA

Table 14: Measured native 2:4 full-forward inference on one RTX 5080, batch 8 and sequence length 256. Each entry uses five counterbalanced dense/2:4 blocks; uncertainty is the 95% Student-t confidence-interval half-width.

Model	Throughput ratio	Dense J/token	Sparse J/token	Saving	CI half-width
60M	1.197×	0.000377	0.000307	18.63%	0.38 pp
130M	1.320×	0.000935	0.000699	25.28%	1.19 pp
250M	1.391×	0.001830	0.001284	29.85%	0.75 pp
350M	1.524×	0.002820	0.001697	39.83%	1.55 pp
1B	1.652×	0.008991	0.005365	40.33%	0.41 pp

Measured backbone inference without CoMoLoRA; energy covers the GPU board only.

The RTX 5080 measurements compare dense and native 2:4 backbone inference at batch 8 and sequence length 256. Five counterbalanced measurement blocks provide the mean energy savings and 95% Student-t confidence-interval half-widths. This uncertainty describes measurement blocks, not training seeds. At 1B, throughput increases by $1.652\times$ and GPU-board energy per full-forward token falls by 40.33%.

These measurements exclude CoMoLoRA. GPU-board energy excludes CPU/host consumption, cooling, and datacentre overhead. The workload is full-forward inference; autoregressive decode and KV-cache costs are not measured.

D.2 Compiler-scheduled 1:4 inference on MOFFET

Table 15: MOFFET compiler-scheduled inference. Latencies in ms; dense is the shared backbone baseline without CoMoLoRA. Sparsity is row-wise on projection weights.

Model	Length	CoMoLoRA	Dense	2:4	1:4	Dense / 1:4	2:4 / 1:4
130M	128	No	12.918	10.035	7.132	1.811×	1.407×
130M	128	Yes	12.918	13.055	9.772	1.322×	1.336×
130M	256	No	23.023	14.237	10.220	2.253×	1.393×
130M	256	Yes	23.023	17.501	13.641	1.688×	1.283×
350M	128	No	37.875	31.195	19.160	1.977×	1.628×
350M	128	Yes	37.875	38.837	26.397	1.435×	1.471×
350M	256	No	68.504	43.881	28.479	2.405×	1.541×
350M	256	Yes	68.504	51.351	36.930	1.855×	1.390×
1B	128	No	132.463	108.882	59.906	2.211×	1.818×
1B	128	Yes	132.463	118.513	69.950	1.894×	1.694×
1B	256	No	260.796	154.841	97.678	2.670×	1.585×
1B	256	Yes	260.796	164.233	108.742	2.398×	1.510×

The MOFFET compiler estimates cover 130M, 350M, and 1B at lengths 128 and 256. Sparse models with and without CoMoLoRA share a dense-backbone baseline. Settings are BF16, four data-parallel cores (batch one each), 700MHz, and DDR bandwidth of 12 bytes/cycle. I/O and DMA scheduling are included; the LM-head processes only the last token. The 1B length-256 entries use nominal sparsity and performance-only compilation without numerical output validation.

CoMoLoRA adds 11.3%–37.8% to 1:4 latency, retaining 1.32–$2.40\times$ speedups over dense and 1.03–$1.56\times$ over the uncompensated 2:4 backbone.

Ratios are within-platform: NVIDIA reports device measurements and MOFFET compiler estimates under different workloads. Sparsity applies to projection weights, excluding the auxiliary branch, embeddings, and output head. Neither panel measures sparse training speed.