BLADE: BLOCK-LEVEL ADAPTIVE MASK DISCOVERY WITH ERROR COMPENSATION

Anonymous authors
Paper under double-blind review

ABSTRACT

N:M sparsity is the pattern hardware accelerates, but selecting its masks well is either cheap or good, not both: one-shot pruners leave a large quality gap to end-to-end learned masks, while the learned reference, MaskLLM, needs thousands of GPU-hours and hundreds of gigabytes of GPU memory at 2:4, and runs out of memory beyond it. We present BLADE (Block-Level Adaptive mask Discovery with Error compensation), which reaches learned-mask quality at a small fraction of end-to-end cost by recasting whole-model N:M compression as a sequence of block-local solves: each block takes its inputs from the already-compressed stream and trains against the dense model's outputs, so upstream error is absorbed rather than compounded, and each solve runs a differentiable Gumbel-softmax search over the block's per-group keep-patterns followed by re-optimization of the surviving weights under the frozen mask. Block-local mask learning runs on a gradient signal that vanishes as the relaxation hardens; we show that with a fair per-optimizer search, every optimizer still converges where it is tuned, but only the sign-based Lion transfers, carrying one configuration, tuned once on a single block of the smallest model, unchanged across blocks and models. Under a unified evaluation harness spanning one-shot, lightweight-learned, and end-to-end-learned baselines on three models, BLADE matches MaskLLM's 2:4 downstream accuracy with roughly $20\times$ less memory and $56\times$ less compute on a single workstation GPU, and at 4:8, where no end-to-end learned masks exist, it attains the best accuracy, perplexity, and KL divergence in every comparison, 2.4% to 4.0% points above the strongest baseline, with its 4:8 models surpassing even MaskLLM's 2:4 accuracy.

1 INTRODUCTION

Large language models are remarkably capable, but serving them is expensive: inference cost is dominated by both the arithmetic over model weights and the memory footprint and bandwidth needed to store and stream them. The recent open-weight Kimi K3 comprises 2.8 trillion parameters (Moonshot AI, 2026), 1.5 TB even in the natively quantized 4-bit form of its public release. At production scale, weight compression is therefore not an optimization but a deployment necessity.

Model compression rests on three main pillars that have matured unevenly: sparsity (Han et al., 2015), quantization (Frantar et al., 2023), and low-rank approximation (Wang et al., 2025). Quantization now offers near-lossless 4-bit deployment as a matter of routine (Frantar et al., 2023; Lin et al., 2024), whereas sparsity at a matched compression ratio still exacts a steep quality tax. Sparsity remains a major open pillar, and closing its quality gap is the subject of this paper.

The structure a sparsity pattern imposes determines both its hardware efficiency and its achievable quality: unstructured sparsity (Han et al., 2015; Frantar & Alistarh, 2023) preserves quality best but its irregular memory access defeats practical acceleration, while structured pruning of channels or heads (Ma et al., 2023) accelerates readily but degrades severely at LLM scale. Semi-structured N:M sparsity, which keeps N nonzeros in every group of M consecutive weights, occupies the acceleratable middle ground: selection remains fine-grained while the pattern stays regular enough for efficient kernels, and 2:4 executes directly on GPU sparse tensor cores (Mishra et al., 2021; Zhou et al., 2021). Higher values of M offer more mask options (at a fixed 50% ratio every 2:4 mask is a valid 4:8 mask, not conversely), and FPGAs, custom accelerators, and GPU kernels such

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.

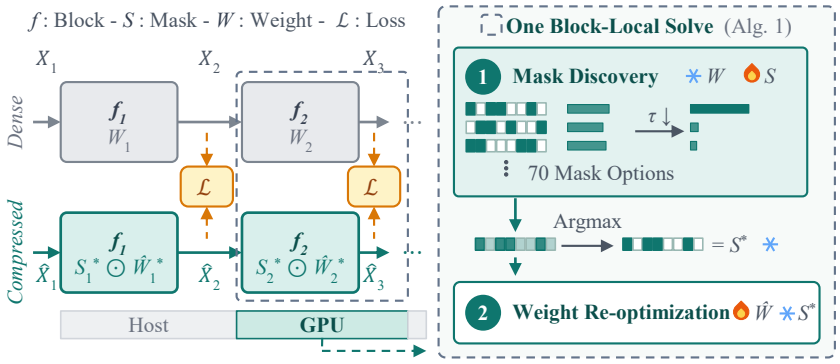


Figure 1: Overview of BLADE. End-to-end N:M compression becomes a sequence of block-local solves; only the block being solved is GPU-resident (grey/teal band). Block i draws its inputs $\widehat{\boldsymbol{X}}_i$ from the compressed stream but trains against dense-stream targets $\boldsymbol{X}_{i+1}$ through the block loss $\mathcal{L}$, absorbing upstream error rather than compounding it (equations 2 and 3). Each solve runs two stages (inset; Algorithm 1; snowflakes frozen, flames trainable): a Gumbel-softmax search over the $\binom{M}{N}$ keep-patterns of every M-group (70 at the illustrated 4:8), annealed by τ and κ and baked by argmax into the hard mask $\boldsymbol{S}^\star$; then re-optimization of the surviving weights under the fixed mask.

as V:N:M (Zhao et al., 2024) increasingly reward patterns richer than 2:4. The practical question is therefore how to select high-quality N:M masks at a cost and at values of M that deployment actually demands.

Existing approaches to N:M mask selection trade quality against compression cost. At the cheap end, one-shot methods select masks with closed-form criteria, ranging from weight magnitude (Han et al., 2015) to calibration-aware saliency scores and closed-form weight updates (Frantar & Alistarh, 2023; Sun et al., 2024; Ilin & Richtárik, 2025; Meng et al., 2024; Boža, 2024; Das et al., 2023; Dong et al., 2024); they prune a 7B model in minutes but leave a substantial quality gap to the dense model. Learned masks close this gap while climbing the cost curve. MaskLLM sets the quality reference by learning per-group mask distributions end to end via Gumbel-softmax sampling (Fang et al., 2024), but at a steep price: roughly 330 GB of GPU memory across eight A100s, 520k training samples, and over 1200 GPU-hours for a 7B model, as measured in the MaskPro and HyperPrune papers (Sun et al., 2025; Sun & Sakuma, 2026). Worse, because it enumerates candidate masks, its logit memory scales as $\mathcal{O}\left(\binom{M}{N} \cdot d/M\right)$ for a layer with d weights, and the approach could not be instantiated at 4:8 even on eight A100s, where Sun et al. (2025) measure out-of-memory beyond 640 GB.

MaskLLM's expense has motivated cheaper successors, but the two recent ones neither recover its quality nor move convincingly beyond 2:4. MaskPro replaces backpropagation with forward-only policy gradients over a linear-space categorical parameterization (Sun et al., 2025), and HyperPrune trains a small shared context-aware hypernetwork that emits per-group masks layer by layer (Sun & Sakuma, 2026). Both, however, keep weights frozen in their headline configurations, and both leave a large quality gap to MaskLLM: MaskPro concedes it in its own comparisons, while HyperPrune did not report comparisons with MaskLLM. Beyond 2:4, HyperPrune reports no learned results, and MaskPro's gains over one-shot baselines are modest; under our unified re-evaluation, the affordable learned methods cluster near one-shot quality while MaskLLM-level quality stays out of reach (Table 1). No existing method delivers the four properties this setting demands: MaskLLM-level quality, single-GPU cost far below end-to-end training, mask and weight co-optimization, and graceful scaling in M.[1]

We close this gap with **BLADE** (**B**lock-**L**evel **A**daptive mask **D**iscovery with **E**rror compensation), a framework that pursues end-to-end-learned mask quality at block-local cost by pairing differentiable mask discovery with gradient-based weight re-optimization. BLADE dissects end-to-end compression into a sequence of block-local reconstruction problems with GPTQ-style error prop-

[1] Additional related work is discussed in Appendix A.

agation (Frantar et al., 2023): each block is conditioned on the actual compressed activations of its predecessors, which keeps every block-local objective aligned with the end-to-end one. Each block is then solved in two stages of fundamentally different character: (i) a differentiable $N{:}M$ mask search over directly parameterized per-group mask distributions, minimizing block-output reconstruction error, and (ii) gradient-based re-optimization of all remaining weights in the block. To our knowledge, BLADE is **the first method to combine directly parameterized per-group mask distributions with gradient-based block-level weight re-optimization under sequential error compensation**, precisely the ingredients the affordable successors trade away (Sun et al., 2025; Sun & Sakuma, 2026). A second finding concerns how this search must be driven. The obvious combination, MaskLLM-style mask learning applied block by block, **suffers from vanishing mask gradients**: as the relaxed distributions harden over a flat block-local loss landscape, the gradient reaching the mask logits shrinks by orders of magnitude, which makes hyperparameter tuning a fragile task. BLADE names this pathology and **drives the mask search with a magnitude-invariant, sign-based optimizer** (Lion, Chen et al., 2023), consistent with an observation GSQ makes in the quantization pillar (Dadgarnia et al., 2026). Decoupling the step from the gradient's scale is what lets us tune every hyperparameter exhaustively on a single transformer block of Qwen2.5-0.5B and transfer the result unchanged across our model suite (Section 3.1). The design maps directly onto the gap: block locality yields a single-block memory footprint with no end-to-end backpropagation, hence a small fraction of end-to-end cost, and the two-stage split combined with sign-based search is what makes larger M tractable, because the real obstacle at 4:8 is per-candidate signal dilution rather than raw combinatorics.

Our experiments bear this design out. Because reported results in this area are separated by mismatched calibration and evaluation conventions, **we contribute a unified evaluation harness** that reproduces one-shot, lightweight-learned, and end-to-end-learned baselines under a single pipeline (Table 1). Under this harness, BLADE **matches MaskLLM's downstream accuracy at 2:4 sparsity with roughly $20\times$ less memory and $56\times$ less compute**. At 4:8, where MaskLLM cannot run, BLADE achieves the best quality in every cell, 2.4 to 4.0% points above the strongest baseline on the eight-task average, and its 4:8 models surpass even MaskLLM's 2:4 accuracy on all three models (Section 3.1).

2 METHODOLOGY

Figure 1 summarizes the design. BLADE converts end-to-end mask learning into a sequence of block-local, error-compensated solves: blocks are visited in order, each calibrated on the compressed model's own activations against the dense model's targets (Section 2.1), and only the block being solved is ever resident on the GPU (Section 3.1). Each solve is attacked in two stages of fundamentally different character: a discrete search over per-group mask distributions (Section 2.2), whose gradients vanish as the relaxation hardens and which is therefore driven by sign-based updates (Section 2.3), followed by a smooth re-optimization of the surviving weights under the baked mask (Section 2.4).

2.1 BLOCK-LOCAL RECONSTRUCTION WITH SEQUENTIAL ERROR COMPENSATION

Consider a pretrained transformer whose decoder consists of blocks $f_1, \dots, f_B$; throughout, a block is one or more consecutive transformer layers, and we use one layer per block unless stated otherwise (block granularity is ablated in Appendix H). $N{:}M$ compression must select, for every linear weight matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ in these blocks, a binary mask $\boldsymbol{S}$ that keeps exactly N of every M consecutive entries along the input dimension, and may additionally adjust the surviving weights. Each group of M weights admits $\binom{M}{N}$ valid keep-patterns, six at 2:4 and seventy at 4:8, and we write $\mathcal{S}_{N:M}$ for the set of admissible masks of a weight matrix. The ideal objective is end to end:

$$\min_{\{\boldsymbol{S}^{(j)},\, \widehat{\boldsymbol{W}}^{(j)}\}} \mathbb{E}_{x\sim\mathcal{D}}\Big[\mathcal{L}_{\text{LM}}\big(x;\, \{\boldsymbol{S}^{(j)} \odot \widehat{\boldsymbol{W}}^{(j)}\}\big)\Big] \qquad \text{s.t.} \quad \boldsymbol{S}^{(j)} \in \mathcal{S}_{N:M} \text{ for every linear layer } j, \tag{1}$$

where $\mathcal{L}_{\text{LM}}$ is the language-modeling loss of the compressed model on data $\mathcal{D}$. MaskLLM optimizes a differentiable relaxation of exactly this objective with the full model in the loop (Fang et al., 2024), which makes its masks both the quality reference and prohibitively expensive (Section 1).

BLADE instead minimizes, one block at a time and in order, the distance between the compressed block's outputs and the dense model's outputs on a small calibration set (Section 3). Let $\boldsymbol{X}_i$ denote the dense model's input activations to block i on the calibration data: $\boldsymbol{X}_1$ contains the embedded calibration tokens, and $\boldsymbol{X}_{i+1} = f_i(\boldsymbol{X}_i; \boldsymbol{W}_i)$, where $\boldsymbol{W}_i$ collects the dense parameters of block i. Block i is solved by minimizing

$$\mathcal{L}\big(\boldsymbol{S}_i, \widehat{\boldsymbol{W}}_i\big) \;=\; \frac{1}{|\Omega_i|} \Big\| f_i\big(\widehat{\boldsymbol{X}}_i;\, \boldsymbol{S}_i \odot \widehat{\boldsymbol{W}}_i\big) - \boldsymbol{X}_{i+1} \Big\|_F^2, \tag{2}$$

where $\boldsymbol{S}_i$ and $\widehat{\boldsymbol{W}}_i$ collect the masks and parameters of all linear layers in the block ($\boldsymbol{S}_i \odot \widehat{\boldsymbol{W}}_i$ masks only the linear weights; biases and normalization parameters are never masked), $\widehat{\boldsymbol{X}}_i$ is the compressed model's input to the block, defined in equation 3 below, and $|\Omega_i|$ is the number of entries of the target tensor. Each block solve proceeds in two stages: a mask search with the block's weights frozen (Section 2.2), then a re-optimization of the surviving weights with the mask fixed (Section 2.4).

This granularity is a deliberate middle ground between the pruning literature's two classical extremes, end-to-end optimization in the lineage of Optimal Brain Damage (LeCun et al., 1989) and layer-wise optimization in the lineage of Optimal Brain Surgeon (Hassibi & Stork, 1992) (rationale in Appendix C), and it has independent theoretical support: Sun & Sakuma (2026) show that maximizing the mutual information between a layer's dense and compressed outputs is approximately equivalent to minimizing their mean squared distance (their Theorem 1).

The two activation streams in equation 2 are the heart of the design. Targets come from the dense model, but inputs come from the compressed model's own forward pass: once block i is solved, with solution $(\boldsymbol{S}_i^\star, \widehat{\boldsymbol{W}}_i^\star)$, the compressed stream advances through it,

$$\widehat{\boldsymbol{X}}_1 = \boldsymbol{X}_1, \qquad \widehat{\boldsymbol{X}}_{i+1} = f_i\big(\widehat{\boldsymbol{X}}_i;\, \boldsymbol{S}_i^\star \odot \widehat{\boldsymbol{W}}_i^\star\big), \tag{3}$$

so block $i{+}1$ is calibrated on the activations it will actually receive at inference time (the first equality holds because embeddings stay dense, Section 3), and each local objective absorbs the accumulated upstream compression error instead of compounding it: every block steers the compressed trajectory back toward the dense one. This sequential error compensation carries the within-layer error propagation of GPTQ and SparseGPT (Frantar et al., 2023; Frantar & Alistarh, 2023), which GPTAQ names asymmetric calibration (Li et al., 2025), from weight columns to whole blocks.

2.2 Stage One: Differentiable Mask Discovery over Per-Group Distributions

Stage one searches for the masks $\boldsymbol{S}_i$ with the block's weights and biases frozen; only the mask logits are trainable. Every group of M consecutive input-dimension weights carries its own learnable categorical distribution over the valid keep-patterns of the group. Let $\mathcal{A}_{N:M} = \big\{\boldsymbol{c} \in \{0,1\}^M : \sum_{m=1}^{M} c_m = N\big\}$ denote this pattern set, where a one marks a kept weight and a zero a pruned one, and let $K = |\mathcal{A}_{N:M}| = \binom{M}{N}$. We fix an arbitrary enumeration $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$ of $\mathcal{A}_{N:M}$, once per (N, M), and stack it as the rows of the candidate matrix $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K]^\top \in \{0,1\}^{K \times M}$, whose k-th row is the k-th valid keep-pattern. We then equip a weight matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ with a logit tensor $\boldsymbol{Z} \in \mathbb{R}^{d_{\text{out}} \times G \times K}$, where $G = d_{\text{in}}/M$ is the number of groups per output row and $\boldsymbol{z}_{r,g} \in \mathbb{R}^K$ denotes the slice for group g of row r. The parameterization is generic in (N, M): nothing but K changes between 2:4 ($K{=}6$) and 4:8 ($K{=}70$), which Section 3.1 exercises.

During mask learning the forward pass applies a Gumbel-softmax relaxation (Jang et al., 2017; Maddison et al., 2017). At training step t with temperature τ_t, each group draws

$$\boldsymbol{p}_{r,g} = \operatorname{softmax}\!\left(\frac{\kappa_t\, \boldsymbol{z}_{r,g} + \boldsymbol{\gamma}_{r,g}}{\tau_t}\right), \qquad \gamma_{r,g,k} = -\log\big(-\log u_{r,g,k}\big), \quad u_{r,g,k} \sim \operatorname{Uniform}(0,1), \tag{4}$$

where $\kappa_t > 0$ is a scheduled logit scale, ramped from κ_{start} to κ_{end} over the mask-learning steps; its rise compounds the temperature anneal's sharpening, and scheduling the scale is standard practice in learned-mask and learned-quantization training (Fang et al., 2024; Hourri et al., 2026; Dadgarnia et al., 2026). Each group then forms the soft mask

$$\tilde{\boldsymbol{m}}_{r,g} = \boldsymbol{C}^\top \boldsymbol{p}_{r,g} \in [0,1]^M. \tag{5}$$

Because $\tilde{\boldsymbol{m}}_{r,g}$ is a convex combination of rows of $\boldsymbol{C}$, each of which keeps exactly N of M entries, the block always executes an exact-N:M mixture: the keep-mass of every group sums to N at every step, while gradients flow to the logits. The forward pass uses $\boldsymbol{W} \odot \widetilde{\boldsymbol{M}}$ with $\boldsymbol{W}$ frozen, and the loss is equation 2. The temperature anneals and the logit scale rises along mirrored geometric schedules over the T_{m} mask-learning steps (equations 7 and 8, Appendix D), hardening the relaxation as training progresses. The tuning search of Appendix D carried a straight-through forward variant and a magnitude-regularizer weight as knobs, and every winning configuration selected the soft forward mask and a zero regularizer; prior learnable-sparsity methods, including MaskLLM and PATCH (Fang et al., 2024; Hourri et al., 2026), rely on such a weight-promoting regularization term to train well, while BLADE needs no such term. When mask learning ends, the distribution is discretized: every group bakes the pattern of its noise-free argmax, $k^{\star}_{r,g} = \arg\max_k z_{r,g,k}$, into a hard mask.

The logits are initialized from a one-shot prior, the only mechanism by which the prior enters training: we run the one-shot solver on the block, set the logit of each group's matching candidate to a strength $\rho > 0$, and set all other logits to zero. By default the prior is SparseGPT (Frantar & Alistarh, 2023), whose per-weight saliencies also supply the per-pattern scores that drive the restricted search of Section 2.3; the choice is immaterial in quality terms (Section 3.1). There is no supervised imitation stage and no persistent prior term in the loss, in contrast to HyperPrune (Sun & Sakuma, 2026): the prior is a starting point, not an anchor, and a block whose learned mask ends worse than its prior in hard-mask block error reverts to it, so stage one is never worse than its one-shot initialization (Appendix C). Stage one also changes the economics of calibration data: one-shot solvers plateau at on the order of 128 sequences (Frantar & Alistarh, 2023; Sun et al., 2024; Meng et al., 2024), while a learned mask keeps converting data into quality once given a few thousand, with nothing near the hundreds of thousands of samples of MaskLLM's end-to-end recipe (Fang et al., 2024) needed; BLADE defaults to 4096 sequences, and Section 3.1 maps the full curves.

2.3 Vanishing Mask Gradients and the Case for Sign-Based Updates

Mask learning on a block-local objective runs on a vanishing signal: block-reconstruction gradients are small to begin with, and the two sharpening schedules that make the mask commit, the temperature anneal and logit-scale ramp of Section 2.2, drive them exponentially toward zero, a decay GSQ bounds explicitly (Dadgarnia et al., 2026) (bound restated in Appendix D.1). In practice, on the first block of Qwen2.5-0.5B the mask-logit gradient norm falls from about 10^{-4} into the 10^{-7} to 10^{-6} band under every optimizer we tested (Figure 2). The vanishing signal does not prevent convergence: given a fair per-optimizer search, every optimizer we tested converges on the block it is tuned on (Section 3.1). What it decides is the price and the portability of convergence: an optimizer that keys its step to the gradient's magnitude must make up the vanishing signal with its learning rate, and a rate chosen this way is tied to the gradient scale it was tuned at, which is a property of the model and spans orders of magnitude across models. Carried unchanged to another model, the Adam-family and SGD configurations lose most or all of their gain, AdamW's to the point of reverting to the one-shot prior, and carried across the blocks of the very model they were tuned on they degrade with depth (Section 3.1).

The remedy is to stop keying the step size to the gradient's magnitude. Lion (Chen et al., 2023) updates parameters with the sign of an interpolated momentum,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \, \operatorname{sign}\big(\beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\, \boldsymbol{g}_t\big), \qquad \boldsymbol{m}_t = \beta_2 \boldsymbol{m}_{t-1} + (1-\beta_2)\, \boldsymbol{g}_t, \tag{6}$$

so every coordinate moves by exactly η no matter how small $\boldsymbol{g}_t$ has become, and a learning rate tuned at one gradient scale means the same thing at another. With Lion as the mask optimizer, at a constant learning rate and with no schedule beyond the temperature anneal and logit-scale ramp of Section 2.2, mask learning converges in every block of Qwen2.5-0.5B, and the single configuration tuned on that model's first block is used for every block of every model in this paper: a learning-rate sweep on the first block of LLaMA-3.1-8B places its own optimum within 0.1% of the transferred value (Appendix D.1), the property on which the tune-once recipe of Section 2.4 rests. GSQ independently reports that Lion is much less sensitive to the vanishing gradients of saturated Gumbel-softmax relaxations (Dadgarnia et al., 2026). The same diagnosis explains why scaling M has been hard: at 4:8 each group spreads the same learning signal over seventy candidates instead of six, so the per-candidate gradient is diluted roughly tenfold before saturation and underflow act at all, and the binding constraint at block scale is signal dilution, not memory. BLADE holds logits only for the block currently being solved and frees them when it is finished, whereas enumeration

across the whole model was measured beyond 640 GB of logit memory at 4:8 for a 7B model (Sun et al., 2025); sign-based search keeps the diluted per-candidate signal usable (Section 3.1).

For large models, even one block's 4:8 logits can strain device memory, so BLADE supports a restricted search: each group's candidate set is cut to the K' patterns ranked highest by the one-shot prior's scores (always including the prior's own pick), and at 4:8 the K'=32-of-70 restriction recovers 99.5% and 94.7% of full-search mask decisions on Qwen2.5-0.5B and LLaMA-3.1-8B at matched held-out block error (Section 3.1; Section 3 lists which runs use it). Enumeration has its own horizon: the per-weight logit store grows from 1.5 values at 2:4 and 8.75 at 4:8 to roughly 800 at 8:16, so block locality removes the depth factor from MaskLLM's memory wall but not the combinatorial one, and BLADE's enumeration is practical exactly through the patterns deployment currently rewards. Beyond that boundary, linear-space parameterizations in the style of MaskPro (Sun et al., 2025) compose naturally with the two-stage design (Appendix C).

2.4 Stage Two: Weight Re-Optimization and the Practical Recipe

With the mask baked in, the character of the problem changes from discrete search to smooth optimization, and BLADE re-optimizes every remaining parameter of the block, the surviving weights, the biases, and the normalization gains, against the same error-compensated objective: equation 2 is minimized over $\widehat{\boldsymbol{W}}_i$ with $\boldsymbol{S}_i$ fixed at $\boldsymbol{S}_i^\star$. This stage adopts the block-wise weight update rule of BEAM (Anonymous, 2025). Pruned coordinates are pinned to exactly zero throughout: the forward pass multiplies the weights by the hard mask, and a straight-through estimator masks their gradients (Bengio et al., 2013), so no pruned weight drifts away from zero. Because the stage reads its mask off the weights' zero pattern, it composes with any pruner; run on top of the one-shot prior alone, with no learned mask, it yields a mask-free BLADE$^{\text{Weight}}$ variant.

The weight stage is a continuous problem and benefits from structure-aware optimization rather than magnitude invariance. BLADE runs Muon (Jordan et al., 2024) on matrix-shaped parameters, with AdamW handling vectors such as biases and normalization gains, under a cosine learning-rate schedule with linear warmup; the same configuration is used for every model and sparsity pattern. The two stages thus want different things from their optimizers, magnitude-blind steps for the mask search against preconditioning for the weight solve (Muon, tested in the mask stage as well, transfers worse than Lion; Section 3.1), and the separation itself is load-bearing: joint optimization is unstable and yields a worse model even where it wins the block objective (Appendix F). Full hyperparameters appear in Appendix D.

Stage two carries a keep-best guard that, combined with stage one's revert-to-prior rule, enforces block by block that the pipeline never ends worse than its one-shot prior in calibration block error; under a properly tuned recipe the guards essentially never fire (Appendix C).

The practical recipe keeps tuning cost off the critical path. All hyperparameters of both stages are tuned once, with Optuna (Akiba et al., 2019), on the first block of the smallest model in our suite, and the winning configuration is frozen for every later block and reused unchanged across models: ten-trial re-searches on the first block of LLaMA-3.1-8B reproduce the Qwen2.5-0.5B winners at both 2:4 and 4:8, consistent with the hyperparameter transfer reported for learned-mask training (Hourri et al., 2026) (Appendix D). This transfer is a property of the optimizer as much as of the architecture: the same tune-once recipe under a magnitude-normalized mask optimizer does not survive the move to another model or to deeper blocks (Section 3.1). Algorithm 1 (Appendix D) summarizes the per-block procedure.

3 Experiments

We first describe the experimental setup, then turn to the results.

Models. We evaluate three models, LLaMA-2-7B, LLaMA-3.1-8B, and Qwen2.5-0.5B, chosen for direct comparison with prior work: MaskLLM reference masks exist for all three (released by the MaskLLM authors for LLaMA-2-7B and LLaMA-3.1-8B (Fang et al., 2024), and by PATCH for Qwen2.5-0.5B (Hourri et al., 2026)).

Baselines. One-shot baselines are Wanda (Sun et al., 2024), SparseGPT (Frantar & Alistarh, 2023), and ADMM (Boža, 2024), each run at its recommended and at BLADE's calibration budget,

reporting the better of the two (selection rule and budget matching in Appendix B). Learned-mask baselines are MaskPro (Sun et al., 2025) and HyperPrune (Sun & Sakuma, 2026), reproduced from their authors' official implementations on our calibration corpus and module scope at their own prescribed training budgets; neither release supports 4:8, so we extended both to it, and Appendices I and J document the reproductions. MaskLLM appears through its authors' released reference masks on all three models (Fang et al., 2024), its training cost ruling out reproduction; where our reproduction of a published number diverges, we show both as reported / ours.

Calibration data. Calibration uses SlimPajama (Soboleva et al., 2023), as in PATCH (Hourri et al., 2026) (sampling and cropping mechanics in Appendix B). All methods we run ourselves, one-shot and learned alike, calibrate on the same data; the one exception is MaskLLM, which enters through released masks trained on their own original data, so it shares the evaluation but not the calibration.

Metrics and tasks. On every model we report WikiText-2 and C4 perplexity and, as a distributional measure against the dense model, KL divergence on WikiText-2, C4, and the calibration set itself; the calibration set is deliberately not held out, so drift on it exposes calibration overfitting (Section 3.1). Perplexity, however, is the weakest of these signals: models with the same pre-training loss can differ markedly downstream (Liu et al., 2023), so we treat KL divergence and downstream accuracy as the primary measures of compression quality and perplexity as a secondary one. We report the eight-task downstream suite of PATCH (Hourri et al., 2026) exactly: MMLU (the 57-subtask aggregate), PIQA, ARC-Easy, ARC-Challenge, WinoGrande, OpenBookQA, RACE, and HellaSwag (task citations in Appendix K). Result tables aggregate these measures: accuracy as the mean over the task suite, and perplexity and KL divergence as the mean over the two held-out corpora (the calibration set is excluded from the average, since it favors the methods calibrated on it); the evaluation software and protocol, per-task and per-corpus results, and the agreement of our harness with PATCH's published numbers all appear in Appendix K.

BLADE configurations. A block is one transformer layer throughout (granularity ablated in Appendix H). Both stages use the Appendix D configurations: mask discovery at 16×4800 batch–steps under Lion, weight re-optimization at 32×2400 under Muon with AdamW on vector parameters, per-pattern winners frozen across all blocks and models. The calibration budget is 4096 sequences of 2048 tokens. The full candidate set is searched at both patterns; the restricted search of Section 2.3 is validated in Section 3.1 but not needed at these scales.

Finally, a note on naming: every table reports two BLADE variants alongside the baselines, **BLADE$^{\text{Mask}}$** (stage one only: the learned mask applied to untouched weights) and **BLADE$^{\text{Full}}$** (both stages). The decomposition attributes every gain: BLADE$^{\text{Mask}}$ isolates what mask learning buys, and its gap to BLADE$^{\text{Full}}$ what weight re-optimization adds.

3.1 Quality Results, Overheads, and Ablations

Table 1 reports the aggregate quality metrics of Section 3 for the three models at 2:4 and 4:8, expanded per task and per corpus in Appendix K. The reproduced learned-mask baselines do not separate from the one-shot methods: on every model and pattern, HyperPrune and MaskPro land at or below the one-shot range of the eight-task average (at most 0.1% points above it, within noise), and each model's stronger of the two sits 1.5 to 6% points below BLADE$^{\text{Mask}}$, with the anomalous HyperPrune 4:8 cell on Qwen2.5-0.5B 8.4 points below (reproduction protocols and the anomaly in Appendices I and J).

At 2:4, BLADE$^{\text{Full}}$ reaches MaskLLM's downstream accuracy at a small fraction of its cost. Its eight-task average lands within 0.25% points of MaskLLM on Qwen2.5-0.5B and LLaMA-3.1-8B and 0.9% points ahead on LLaMA-2-7B, and BLADE$^{\text{Mask}}$ alone, which trains no weights, already clears the best one-shot method by 2.5 to 6% points. Each eight-task average carries a sampling standard error of roughly 0.4% points, so a difference of two carries about 0.6 (Appendix K); the two deficits are under one such error and the 0.9-point advantage about 1.4, so we read none of the three gaps as decisive. MaskLLM keeps the lower perplexity and KL divergence at 2:4, the imprint of its 520k-sample end-to-end training; BLADE delivers the matching downstream quality from 4096 block-local sequences at $56\times$ lower cost (Section 3.1).

Table 1: Main results. Eight-task zero-shot average accuracy (%, ↑), perplexity averaged over WikiText-2 and C4 (↓), and KL divergence to the dense model averaged over the same two held-out corpora (nats/token, top-512 lower bound, ↓). MaskLLM enters through its authors' released masks, which exist only at 2:4. MaskPro and HyperPrune are our reproductions (Appendices I and J). Best compressed method per column in **bold**, the second best underlined; per-task, per-corpus, and calibration-stream results in Appendix K.

		Qwen2.5-0.5B			LLaMA-2-7B			LLaMA-3.1-8B		
	Method	Acc. (↑)	PPL (↓)	KLD (↓)	Acc. (↑)	PPL (↓)	KLD (↓)	Acc. (↑)	PPL (↓)	KLD (↓)
	Dense	46.06	32.9	–	54.58	14.0	–	60.59	12.8	–
2:4	Wanda	32.72	285.2	1.807	43.47	39.8	0.758	42.34	66.1	1.303
2:4	SparseGPT	33.48	97.8	0.930	44.55	30.2	0.557	44.88	36.2	0.862
2:4	ADMM	35.01	85.2	0.804	45.61	31.2	0.574	45.71	36.7	0.863
2:4	HyperPrune	32.23	233.9	1.808	44.81	35.5	0.657	45.71	38.1	0.900
2:4	MaskPro	32.67	219.3	1.640	45.45	30.3	0.536	43.90	42.8	0.957
2:4	MaskLLM	**39.13**	**39.8**	**0.283**	48.71	**19.8**	**0.273**	**52.98**	**18.1**	**0.341**
2:4	BLADE$^{\text{Mask}}$	37.51	52.8	0.403	48.81	23.5	0.368	51.71	23.9	0.517
2:4	BLADE$^{\text{Full}}$	38.95	47.9	0.318	**49.57**	22.6	0.296	52.75	22.7	0.461
4:8	Wanda	35.13	137.3	1.213	47.02	25.1	0.419	47.79	34.4	0.787
4:8	SparseGPT	35.79	67.9	0.605	47.76	23.3	0.370	49.51	26.3	0.605
4:8	ADMM	36.85	64.4	0.555	48.70	23.6	0.368	50.66	26.6	0.601
4:8	HyperPrune	30.64	490.8	2.214	48.47	24.8	0.398	50.74	27.5	0.632
4:8	MaskPro	34.97	114.5	1.097	47.87	23.9	0.362	49.60	28.8	0.649
4:8	BLADE$^{\text{Mask}}$	39.01	46.7	0.305	49.96	**21.0**	**0.259**	53.75	21.4	0.416
4:8	BLADE$^{\text{Full}}$	**39.63**	**44.8**	**0.252**	**51.05**	22.9	0.271	**54.71**	**20.5**	**0.376**

At 4:8, where MaskLLM released no masks and cannot be instantiated even on eight A100s (Sun et al., 2025), **BLADE$^{\text{Full}}$ is the best method in every cell**: 2.4 to 4.0% points above the strongest baseline on the eight-task average, with held-out KL divergence 1.3 to 2.2× lower. The extra mask freedom is real: on all three models, **BLADE$^{\text{Full}}$ at 4:8 surpasses even MaskLLM's 2:4 accuracy**, so where hardware rewards the richer pattern (Section 1), BLADE converts it into quality 2:4-bound learned masks do not reach.

The two stages compose as designed: BLADE$^{\text{Full}}$ improves on BLADE$^{\text{Mask}}$ in every cell of the eight-task average and lowers held-out KL divergence everywhere except LLaMA-2-7B at 4:8 (Appendix K).

Compression cost: wall-clock and peak memory. We measure the wall-clock and peak resource cost of both BLADE variants on a single NVIDIA RTX PRO 6000 workstation GPU (256 GiB host RAM) at the full 4096-sequence calibration budget, one-time setup excluded (protocol and analysis in Appendix G): per Table 17 (Appendix G), BLADE$^{\text{Full}}$ compresses the 0.5B model in two hours and the 7B to 8B models in about a day. This cost sits between the one-shot methods, minutes to a couple of hours, and MaskLLM's reported $20 \times 64 = 1{,}280$ A100 GPU-hours for LLaMA-2-7B (Fang et al., 2024); the hardware differs, so GPU-hour comparisons are approximate, but the gap is not close: **BLADE$^{\text{Full}}$ compresses LLaMA-2-7B in 22.8 hours on one workstation GPU, roughly 56× fewer GPU-hours than MaskLLM's 7B figure**, and needs no multi-GPU node at any scale evaluated. Memory follows the same pattern, because exactly one block is resident on the GPU at a time while the rest of the model is parked in host memory: peak device memory is block-local and independent of model depth, from 13.4 GiB (LLaMA-2-7B at 2:4) to 41.4 GiB (LLaMA-3.1-8B at 4:8) on the same workstation GPU, with host-side activation stores of 123 to 387 GiB (Table 17; streaming mechanics in Appendix G). For contrast, MaskLLM's end-to-end training was measured at roughly 330 GB of GPU memory across eight A100s at 2:4, and out of memory beyond 640 GB at 4:8 (Sun et al., 2025; Sun & Sakuma, 2026).

Prior mask and restricted search. Two robustness checks confirm that stage one's cheap ingredients carry no hidden weight. First, the one-shot prior that initializes the mask logits (Section 2.2) is immaterial: compressing every block of Qwen2.5-0.5B under four priors (SparseGPT, magnitude, Wanda, and ADMM) spans 0.24% points of eight-task accuracy at 2:4 and 0.81% points at 4:8 with no consistent winner; **the learned mask, not the prior, determines quality** (Table 9,

Appendix D.2). Second, the top-32 restriction of Section 2.3 is faithful where the full search still fits: learned full-search 4:8 patterns lie within the prior's top-32 candidates for 99.5% of groups on Qwen2.5-0.5B and 94.7% on LLaMA-3.1-8B, at matched held-out block error (Tables 10 and 3, Appendix D.2).

Mask-stage optimizer and gradient magnitude. Section 2.3 argues that the mask stage should be driven by a sign-based optimizer, and this ablation tests it on equal footing: AdamW, Adam, Muon, and SGD each get their own Optuna search on the first block of Qwen2.5-0.5B at 2:4, and each winner is then frozen and carried unchanged to the first block of LLaMA-3.1-8B and to all 24 blocks of Qwen2.5-0.5B (Table 4, Appendix D.1). On the tuning block all five optimizers converge within 3.7 points of one another, and the mask-logit gradient vanishes under all of them, Lion included (Figure 2). What separates them is transfer: on LLaMA-3.1-8B AdamW, the best performer on Qwen, reverts to the prior and the spread across optimizers widens from 3.7 to 63.6 points, and across the 24 blocks of Qwen2.5-0.5B, Lion's BLADE$^{\text{Mask}}$ model reaches a KL divergence 38% below AdamW's. A first-block search therefore does not select the optimizer that produces the best model; **the tune-once recipe of Section 2.4 works because Lion's configuration is the one that survives being frozen** (protocol, search spaces, and gradient-norm traces in Appendix D.1).

Calibration-set size and overfitting. Calibration-set size is a knob for BLADE and nearly a no-op for one-shot pruners. From 128 to 4096 sequences on Qwen2.5-0.5B at 2:4, the held-out KL divergence of ADMM, SparseGPT, and Wanda moves by 1 to 8% on C4, whereas **BLADE is the only method that keeps converting calibration data into quality**: BLADE$^{\text{Full}}$'s held-out KL falls by 49% and BLADE$^{\text{Mask}}$'s by 19%, leaving BLADE$^{\text{Full}}$ 2.7 to $6.5\times$ below the one-shot methods at 4096 sequences (Figure 3, Appendix E). Below a few thousand sequences BLADE memorizes its calibration set, the ratio of held-out to calibration KL reaching $10\times$ at 128 sequences against a one-shot floor of 1.1 to $1.3\times$, and training less does not repair small data (full study in Appendix E).

The appendix carries the full versions of everything summarized above: per-task and per-corpus results for every method, model, and pattern (Appendix K), the MaskPro and HyperPrune reproduction protocols (Appendices I and J), the hyperparameter search spaces and winners together with the complete optimizer study (Appendix D), and the calibration-size, joint-optimization, and compression-cost studies (Appendices E to G). Two further studies live there in full. **Joint optimization wins the objective it is tuned on and loses the model**: trained jointly, mask and weights reach a lower first-block error than BLADE$^{\text{Full}}$ yet end 23% worse in held-out KL divergence, because the weights compensate a mask that is still moving and softened (Appendix F). And in the block-granularity study, **coarser units buy a modest, monotone distributional gain** over the one-layer default while the eight-task average moves within sampling noise, so the default stays at one layer per unit (Appendix H).

4 Conclusion and Limitations

BLADE turns end-to-end $N{:}M$ mask learning into a sequence of block-local two-stage solves, a differentiable mask search followed by weight re-optimization, trained on compressed-stream inputs against dense-stream targets so block errors are compensated rather than compounded. Driving the search with a sign-based optimizer makes the recipe practical: one configuration, tuned on one block of the smallest model, transfers unchanged across blocks, models, and patterns. Under a unified evaluation harness, BLADE matches MaskLLM's 2:4 downstream accuracy with roughly $20\times$ less memory and $56\times$ less compute on a single workstation GPU, and at 4:8 it is the best method in every comparison, its 4:8 models surpassing even MaskLLM's 2:4 accuracy.

These results come with limits, each discussed in full in Appendix L. The block solves are sequential and greedy, so no downstream signal can improve an earlier block's mask. Enumeration is practical through 4:8 but not at 8:16, and the restricted search is validated only where the full search fits. Masks and weights are calibrated on one data distribution, with no guarantee under shift. The tune-once recipe is verified only under Lion on the evaluated families. Compression takes a day at the 7B scale rather than one-shot minutes, quality beyond the evaluated scales is unverified, and realized 4:8 speedup depends on still-uneven kernel support outside 2:4 tensor cores.

AI USE STATEMENT

In this work, we used generative AI tools to assist with drafting and editing text, with literature search, and with implementing and editing the codebase. All AI-assisted text was reviewed and edited by the authors, all AI-assisted code was reviewed and verified by the authors, and all citations and reported numbers were verified against the original sources by the authors. Beyond these uses, we have not used generative AI tools for research ideation, experimental design, or analysis of results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops methods for compressing existing, publicly released language models; it involves no new data collection, no human subjects, and no personally identifiable information, and all models and datasets are used under their published licenses. The societal effects of the method largely track those of the models it compresses: compression reduces the memory, energy, and monetary cost of serving, which broadens access to capable models and reduces the environmental footprint of inference, while the same economics apply to beneficial and harmful uses alike. Compressed models are not behaviorally identical to their dense sources: sparsification can shift capabilities and error patterns unevenly across domains, so compressed checkpoints should be re-evaluated against a deployment's quality and safety bar rather than inheriting the dense model's assessments. Our evaluation protocol is specified precisely (Section 3) to make such re-evaluation reproducible.

REPRODUCIBILITY STATEMENT

All models are publicly available checkpoints, identified by name in Section 3; MaskLLM reference masks come from their authors' public releases (the MaskLLM team for LLaMA-2-7B and LLaMA-3.1-8B; PATCH for Qwen2.5-0.5B). Calibration and evaluation data are public, with the exact source, sampling seed, sequence count, and sequence length given in Section 3; the evaluation stack pins a specific `lm-evaluation-harness` version, task set, and decoding protocol, and per-item outputs are logged for every configuration. Every BLADE hyperparameter appears in Appendix D, produced by the first-block search of Section 2.4 and frozen thereafter; Algorithm 1 specifies the per-block procedure, and Appendix D contains the full search spaces and winning configurations. Baseline methods are run from their authors' official implementations[2] with recommended configurations, and where our reproduction differs from a reported number, both are shown. Hardware and software stack: single NVIDIA RTX PRO 6000 GPUs in `bfloat16` (Kalamkar et al., 2019) throughout, with `lm-evaluation-harness v0.4.10`, `torch v2.13.0+cu130`, and `transformers v5.15.1` (Appendix K); compression hardware and cost are detailed in Section 3.1.

[2]Wanda: `https://github.com/locuslab/wanda`; SparseGPT: `https://github.com/IST-DASLab/sparsegpt`; ADMM: `https://github.com/fmfi-compbio/admm-pruning`; MaskPro: `https://github.com/woodenchild95/Maskpro`; HyperPrune: `https://github.com/futuresun912/HyperPrune`.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

Anonymous. BEAM: Blockwise error minimization for one-shot compression of LLMs. Reference withheld for double-blind review, 2025. Blog post.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 560–569, 2018.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Vladimír Boža. Fast and effective weight update for pruned large language models. *Transactions on Machine Learning Research*, 2024.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. EfficientQAT: Efficient quantization-aware training for large language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, Kaokao Lv, and Yi Liu. Optimize weight rounding via signed gradient descent for the quantization of LLMs. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Alireza Dadgarnia, Soroush Tabesh, Mahdi Nikdan, Michael Helcig, Eldar Kurtic, Maximilian Kleinegger, and Dan Alistarh. GSQ: Highly-accurate low-precision scalar quantization for LLMs via Gumbel-Softmax sampling. *arXiv preprint arXiv:2604.18556*, 2026.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models. *arXiv preprint arXiv:2311.04902*, 2023.

Xin Ding, Xiaoyu Liu, Zhijun Tu, Yun Zhang, Wei Li, Jie Hu, Hanting Chen, Yehui Tang, Zhiwei Xiong, Baoqun Yin, and Yunhe Wang. CBQ: Cross-block quantization for large language models, 2023.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. MaskLLM: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. LQ-LoRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *International Conference on Learning Representations (ICLR)*, 2024a.

Song Guo, Fan Wu, Lei Zhang, Xiawu Zheng, Shengchuan Zhang, Fei Chao, Yiyu Shi, and Rongrong Ji. EBFT: Effective and block-wise fine-tuning for sparse LLMs, 2024b.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5 (NIPS)*, pp. 164–171. Morgan Kaufmann, 1992.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. Presented at the NeurIPS 2014 Deep Learning and Representation Learning Workshop.

Younes Hourri, Mohammad Mozaffari, and Maryam Mehri Dehnavi. PATCH: Learnable tile-level hybrid sparsity for LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

Weiyu Huang, Yuezhou Hu, Guohao Jian, Jun Zhu, and Jianfei Chen. Pruning large language models with semi-structural adaptive sparse training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Ivan Ilin and Peter Richtárik. Thanos: A block-wise pruning algorithm for efficient large language model compression. *arXiv preprint arXiv:2504.05346*, 2025.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

Keller Jordan, Yuchen Jin, Vlado Boža, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024. Blog post.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of bfloat16 for deep learning training, 2019.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2 (NIPS)*, pp. 598–605. Morgan Kaufmann, 1989.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations (ICLR)*, 2021.

Yuhang Li, Ruokai Yin, Donghyun Lee, Shiting Xiao, and Priyadarshini Panda. GPTAQ: Efficient finetuning-free quantization for asymmetric calibration. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning (ICML)*, 2023.

Hongyi Liu, Rajarshi Saha, Zhen Jia, Youngsuk Park, Jiaji Huang, Shoham Sabach, Yu-Xiang Wang, and George Karypis. ProxSparse: Regularized learning of semi-structured sparsity masks for pretrained LLMs. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025a.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations. In *International Conference on Learning Representations (ICLR)*, 2025b.

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 *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22812–22824, 2023.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

Xiang Meng, Kayhan Behdin, Haoyue Wang, and Rahul Mazumder. ALPS: Improved optimization for highly sparse one-shot pruning for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=0lBx844upd.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

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.

Moonshot AI. Kimi k3. https://huggingface.co/moonshotai/Kimi-K3, July 2026. Open-weight model release; 2.8T-parameter mixture-of-experts.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R. Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627b token cleaned and deduplicated version of RedPajama. `https://huggingface.co/datasets/cerebras/SlimPajama-627B`, 2023. Cerebras blog post and dataset release.

Yupeng Su, Ziyi Guan, Xiaoqun Liu, Tianlai Jin, Dongkuan Wu, Zhengfei Chen, Graziano Chesi, Ngai Wong, and Hao Yu. LLM-Barber: Block-aware rebuilder for sparsity mask in one-shot for large language models. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

Lu Sun and Jun Sakuma. Learning semi-structured sparsity for LLMs via shared and context-aware hypernetwork. In *International Conference on Learning Representations (ICLR)*, 2026. URL `https://openreview.net/forum?id=lqjQs2lVNm`.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Wei Sun, Aojun Zhou, Sander Stuijk, Rob Wijnhoven, Andrew Oakleigh Nelson, Hongsheng Li, and Henk Corporaal. Dominosearch: Find layer-wise fine-grained N:M sparse schemes from dense neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL `https://openreview.net/forum?id=IGrC6koW_g`.

Yan Sun, Qixin Zhang, Zhiyuan Yu, Xikun Zhang, Li Shen, and Dacheng Tao. MaskPro: Linear-space probabilistic learning for strict (N:M)-sparsity on LLMs. *arXiv preprint arXiv:2506.12876*, 2025.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations (ICLR)*, 2025.

Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. BESA: Pruning large language models with blockwise parameter-efficient sparsity allocation. In *International Conference on Learning Representations (ICLR)*, 2024.

Yifan Yang, Kai Zhen, Bhavana Ganesh, Aram Galstyan, Goeric Huybrechts, Markus Müller, Jonas M. Kübler, Rupak Vignesh Swaminathan, Athanasios Mouchtaris, Sravan Babu Bodapati, Nathan Susanj, Zheng Zhang, Jack FitzGerald, and Abhishek Kumar. Wanda++: Pruning large language models via regional gradients. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Kumar Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Yuxin Zhang, Mingbao Lin, Zhihang Lin, Yiting Luo, Ke Li, Fei Chao, Yongjian Wu, and Rongrong Ji. Learning best combination for efficient N:M sparsity. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Yunyun Sun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse LLMs. In *International Conference on Learning Representations (ICLR)*, 2024.

Kang Zhao, Tao Yuan, Han Bao, Zhenfeng Su, Chang Gao, Zhaofeng Sun, Zichen Liang, Liping Jing, and Jianfei Chen. Beyond 2:4: Exploring V:N:M sparsity for efficient transformer inference on GPUs. *arXiv preprint arXiv:2410.16135*, 2024.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning N:M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021.

Lancheng Zou, Shuo Yin, Zehua Pei, Tsung-Yi Ho, Farzan Farnia, and Bei Yu. PermLLM: Learnable channel permutation for N:M sparse large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Appendix

Table of Contents

A Related Work **16**

A.1 Pruning for Large Language Models 17

A.2 Learned N:M Masks 17

A.3 Block-Level Reconstruction and Error Compensation 18

A.4 Complementary Compression Pillars 18

A.5 Optimization Machinery for Discrete Mask Search 18

B Experimental-Setup Details **19**

C Method Details **19**

D Hyperparameter Settings **20**

D.1 Mask-Stage Optimizer Study 20

D.2 Prior-Mask and Restricted-Search Tables 24

E Calibration-Set Size Study **24**

F Joint Versus Two-Stage Optimization **27**

G Compression-Cost Details **29**

H Block-Granularity Study **30**

I MaskPro Reproduction **32**

J HyperPrune Reproduction **33**

K Per-Task and Per-Corpus Results **34**

L Extended Limitations **35**

A Related Work

Model compression research divides across three pillars: sparsity, quantization, and low-rank approximation. The pillars are not siloed: techniques proven in one repeatedly transfer to another, and three such transfers are load-bearing for this paper: block-level reconstruction granularity and asymmetric calibration between the compressed and dense streams (both Section A.3), and sign-based optimization of hardening categorical relaxations (Section A.5). We first survey the sparsity pillar in depth (Sections A.1–A.3), then the complementary pillars (Section A.4), and close with the optimization machinery underlying learnable masks (Section A.5).

A.1 Pruning for Large Language Models

Pruning removes redundant parameters while preserving model quality, and the classical literature already spans the two extremes that frame modern work. At the local extreme, Optimal Brain Surgeon uses second-order information to select weights and compute compensating updates for the survivors (Hassibi & Stork, 1992), an idea made tractable layer by layer at scale by Optimal Brain Compression (Frantar et al., 2022). At the global extreme, Optimal Brain Damage scores saliency against the end-to-end loss through a diagonal Hessian approximation (LeCun et al., 1989), and WoodFisher extends this direction with block-structured Fisher approximations (Singh & Alistarh, 2020). LLM-scale one-shot methods descend mainly from the local line: SparseGPT casts layer-wise pruning as sparse regression with efficient column-block OBS-style updates and native N:M support (Frantar & Alistarh, 2023); Wanda drops weight updates entirely in favor of a magnitude-times-activation criterion (Sun et al., 2024); Thanos improves the closed-form updates with multi-column block approximations, where block denotes weight sub-blocks within a layer rather than the transformer blocks of this paper (Ilin & Richtárik, 2025); Boža (2024) computes the post-mask layer-wise update with a single matrix inversion and a handful of ADMM iterations, extended with gradual mask re-selection, and is the method our experiments label ADMM; ALPS likewise brings ADMM-based optimization guarantees to one-shot pruning (Meng et al., 2024); and gradient-informed or evolved saliency metrics refine the scoring function (Das et al., 2023; Dong et al., 2024). A complementary axis allocates how much to prune where, through outlier-aware layer-wise ratios (Yin et al., 2024) and differentiable per-block sparsity budgets (Xu et al., 2024), without selecting the N:M masks themselves. A further refinement re-selects the mask after the fact: DSnoT iteratively prunes and grows weights with no training at all, choosing each swap by its anticipated reduction in reconstruction error (Zhang et al., 2024), and LLM-Barber rebuilds the mask in one shot across whole attention and MLP blocks using a weight-times-gradient score (Su et al., 2025). Both remain score-driven re-selection over frozen weights: neither learns a mask distribution nor re-optimizes the survivors, the two ingredients BLADE combines. Wanda++ moves one step toward the present work by injecting decoder-block-level regional gradients and light regional optimization into one-shot scoring, though its masks remain score-derived rather than learned (Yang et al., 2025). These methods prune a 7B model in minutes, but at hardware-friendly semi-structured patterns they leave a substantial quality gap, and their pruning decisions remain local and largely heuristic.

A.2 Learned N:M Masks

A separate line learns the masks themselves. During pretraining, SR-STE trains N:M-sparse networks from scratch with straight-through estimation (Zhou et al., 2021), Zhang et al. (2022) learn the best per-group combination directly, STEP stabilizes from-scratch mask learning with preconditioning (Lu et al., 2023), and DominoSearch searches mixed layer-wise N:M schemes (Sun et al., 2021). For post-training compression of pretrained LLMs, MaskLLM (Fang et al., 2024) sets the quality reference: it places a categorical distribution over the full candidate set of each M-group and trains it end to end through Gumbel-softmax sampling against the language-modeling loss, producing transferable masks, at the enumeration cost documented in the introduction (as measured in the MaskPro and HyperPrune papers; Sun et al., 2025; Sun & Sakuma, 2026). Successors attack that cost from several directions. ProxSparse performs regularized proximal mask selection over frozen weights at calibration scale (Liu et al., 2025a). MaskPro collapses the candidate enumeration into a linear-space categorical parameterization trained with forward-only policy gradients from a prior-mask initialization, extending to 4:8 and 8:16 patterns (Sun et al., 2025). HyperPrune amortizes mask generation through a small shared, context-aware hypernetwork applied layer-sequentially with continual-pruning and feature-outlier regularizers, stabilized by a supervised stage that imitates SparseGPT masks; its headline results keep weights frozen, with closed-form SparseGPT-style compensation available as an optional configuration (Sun & Sakuma, 2026). Retraining-based approaches such as AST recover quality through semi-structural adaptive sparse training at correspondingly higher cost (Huang et al., 2025). PATCH learns tile-level hybrid sparsity, selecting per tile between dense and 2:4 execution to realize continuous sparsity ratios between 0% and 50% (Hourri et al., 2026). PermLLM optimizes an axis the methods above leave fixed: which channels share an M-group. It learns channel permutations through differentiable Sinkhorn-relaxed permutation matrices so that N:M groups retain more important weights (Zou et al., 2025); permutation is orthogonal to selecting the mask within a group and composes with BLADE. Across this line, no existing method co-optimizes directly parameterized per-group mask distributions with gradient-

based weight updates at block granularity under sequential error compensation; that combination is BLADE's contribution.

A.3 Block-Level Reconstruction and Error Compensation

Between purely local and fully end-to-end optimization lies a middle ground that this paper deliberately occupies, and its two ingredients have separate lineages. Error compensation descends from Optimal Brain Surgeon (Hassibi & Stork, 1992): GPTQ propagates quantization error across columns within a layer (Frantar et al., 2023), and SparseGPT does the same for pruning (Frantar & Alistarh, 2023); conditioning each unit on the actual compressed outputs of its predecessors is what keeps local objectives aligned with the end-to-end one. GPTAQ makes the asymmetry explicit in the quantization pillar, naming and analyzing the scheme that calibrates compressed-stream computation against dense-stream targets (Li et al., 2025); BLADE carries this asymmetric calibration from within-layer weight columns to whole blocks. Reconstruction granularity was settled first in the quantization pillar: BRECQ showed that block-level reconstruction dominates layer-wise reconstruction for post-training quantization (Li et al., 2021), an insight BLADE transports to learnable sparsity. LLM-scale post-training quantizers have since standardized on that granularity: OmniQuant learns clipping and equivalent-transformation parameters under block-wise error minimization (Shao et al., 2024), CBQ widens the reconstruction window across several blocks to curb error accumulation (Ding et al., 2023), and EfficientQAT trains all parameters of one block at a time before a light end-to-end pass (Chen et al., 2025). AutoRound is doubly relevant: it drives block-wise rounding and clipping with signed gradient descent (Cheng et al., 2024), prefiguring in quantization the magnitude-discarding update family that Section A.5 argues for and BLADE adopts for masks. Gradient-based block-wise weight re-optimization also has sparsity-side precedent: EBFT fine-tunes a pruned model block by block against reconstruction error on a small calibration set (Guo et al., 2024b), a prior instance of BLADE's mask-free BLADE$^{\text{Weight}}$ configuration, and BEAM (Anonymous, 2025) supplies the update rule BLADE adopts in its second stage. BLADE combines these ingredients with a differentiable mask search, dissolving the local/global dichotomy: block-local cost with error-propagated, end-to-end-aligned objectives.

A.4 Complementary Compression Pillars

Sparsity is one of three compression pillars alongside quantization and low-rank approximation, and the pillars have matured unevenly. Quantization is the most mature: GPTQ (Frantar et al., 2023) and AWQ (Lin et al., 2024) made near-lossless 4-bit weight compression routine, and rotation-based outlier suppression extends it to harder regimes (Ashkboos et al., 2024; Liu et al., 2025b). Low-rank approximation compresses weight matrices directly, as in SVD-LLM (Wang et al., 2025) and LQ-LoRA (Guo et al., 2024a). Knowledge distillation offers a further orthogonal route through teacher-student transfer (Hinton et al., 2015). BLADE targets the sparsity pillar, and its block-local formulation leaves the other pillars' standard toolchains untouched, enabling rather than obstructing composition.

A.5 Optimization Machinery for Discrete Mask Search

Learning discrete masks relies on continuous relaxations, straight-through estimation (Bengio et al., 2013) and the Gumbel-softmax and Concrete distributions (Jang et al., 2017; Maddison et al., 2017), whose gradients can become vanishingly small as the relaxed distributions harden. Sign-based methods discard gradient magnitude: signSGD established the family (Bernstein et al., 2018), and Lion was discovered by program search as a practical magnitude-invariant optimizer (Chen et al., 2023). Notably, this remedy has independent support in the quantization pillar: GSQ observes the same rescue in the quantization pillar, with Lion keeping traction after its Gumbel-softmax relaxation saturates (Dadgarnia et al., 2026), evidence that sign-based updates address a pathology of hardening categorical relaxations rather than an artifact of any single compression task. For the continuous weight stage, structure-aware optimizers apply: Muon orthogonalizes momentum for hidden-layer weight matrices (Jordan et al., 2024). BLADE's finding, that block-local mask learning under magnitude-normalized optimizers converges only at learning rates that fail to transfer across models and blocks, while sign-based updates transfer unchanged, corroborating in the sparsity pillar what GSQ observed in quantization, connects this machinery directly to the semi-structured pruning problem.

B EXPERIMENTAL-SETUP DETAILS

This appendix carries the setup mechanics summarized in Section 3.

Budget matching and baseline selection. All budgets are matched in sequences and tokens alike: 4096 sequences of 2048 tokens (8.4M tokens) against the field-standard 128×2048 (262k tokens). Each one-shot baseline is run at both its recommended 128-sequence calibration budget and at BLADE's 4096-sequence budget, and every table reports whichever of the two attains the lower KL divergence to the dense model on WikiText-2; this per-model selection can only favor the baseline.

Calibration sampling. SlimPajama sequences are drawn from the 6B-token subset's train split in streaming mode under a seeded shuffle, and each calibration sequence is a random 2,048-token crop of a randomly selected document, never a prefix of the corpus.

C METHOD DETAILS

Block granularity as a middle ground. The block-local objective of Section 2.1 sits between the pruning literature's two classical extremes, end-to-end optimization in the lineage of Optimal Brain Damage (LeCun et al., 1989) and layer-wise optimization in the lineage of Optimal Brain Surgeon (Hassibi & Stork, 1992), and improves on each in one direction: block outputs are the sole channel through which compression error reaches the loss, so keeping them small, with the sequential compensation of Section 2.1, keeps the end-to-end error controlled, and unlike a layer-wise solver the block objective accounts for the interactions among the block's linear layers when choosing masks and updates.

Restricted search. For large models, even one block's 4:8 logits can strain device memory. BLADE therefore supports a restricted search: each group's candidate set is cut from all $\binom{M}{N}$ patterns to the K' patterns ranked highest by the one-shot prior's own scores; the prior's selected pattern is the top-ranked candidate by construction, so the restriction always contains the initialization. Concretely, a pattern's score is the sum of the prior's per-weight saliencies over the weights the pattern keeps: each group's weights are sorted by saliency, and patterns are ranked by that sum in the sorted coordinates, which places the prior's own pick first. At 4:8 we use $K' = 32$ of 70, which recovers 99.5% and 94.7% of full-search mask decisions on Qwen2.5-0.5B and LLaMA-3.1-8B respectively, at matched held-out block error; Section 3.1 reports the full analysis, and Section 3 lists which runs use the restriction.

The enumeration horizon. Per weight, the logit store costs $\binom{M}{N}/M$ values: 1.5 at 2:4 and 8.75 at 4:8, but roughly 800 at 8:16, so even a single block's logits would occupy hundreds of gigabytes for a 7B model. Block locality removes the depth factor from MaskLLM's memory wall; it does not repeal the combinatorial one. BLADE's enumeration is therefore practical exactly through the patterns deployment currently rewards, 2:4 and 4:8, and beyond that boundary linear-space parameterizations in the style of MaskPro (Sun et al., 2025) compose naturally with the two-stage block-local design; we leave that composition to future work. Top-K restriction extends enumeration's reach in the same direction: it trades candidate coverage for memory, and we validate that trade only at 4:8.

Prior anchoring and stage guards. There is no supervised imitation stage and no persistent prior term in the loss, in contrast to HyperPrune's 12,000-step supervised stage imitating SparseGPT masks plus prior logit biases (Sun & Sakuma, 2026): the prior is a starting point, not an anchor. Two guards protect the pipeline instead. In stage one, a block whose learned mask ends with a higher hard-mask block error than its prior reverts to the prior, so mask learning is never worse than its one-shot initialization. In stage two, the block is scored periodically on a fixed calibration subset and the best snapshot is restored at the end, so weight re-optimization never leaves a block worse than stage one left it. Combined, the guards enforce, block by block, that the pipeline never ends worse than its one-shot prior in calibration block error. Under a properly tuned recipe they essentially never fire; they are stability insurance against out-of-tune hyperparameters or an ill-suited optimizer (Section 3.1), not load-bearing components.

Algorithm 1 One block-local solve of BLADE (block i)

Input: dense inputs $\boldsymbol{X}_i$ and targets $\boldsymbol{X}_{i+1}$; compressed inputs $\widehat{\boldsymbol{X}}_i$; block parameters $\boldsymbol{W}_i$; pattern (N, M)

Output: baked masks $\boldsymbol{S}_i^\star$; re-optimized parameters $\widehat{\boldsymbol{W}}_i^\star$; next-block inputs $\widehat{\boldsymbol{X}}_{i+1}$

compute the one-shot prior (SparseGPT by default) on the block and set the matching logits to ρ ▷ Section 2.2
for $t = 0, \ldots, T_\mathrm{m} - 1$ **do** ▷ stage one: mask discovery, weights frozen
    sample soft masks by equations 4 and 5 at τ_t from equation 7
    update logits $\boldsymbol{Z}$ with a Lion step on $\mathcal{L}$ ▷ equation 6
end for
bake $\boldsymbol{S}_i^\star$ by per-group argmax; revert to the prior mask if the hard-mask error worsened
for $t = 1, \ldots, T_\mathrm{w}$ **do** ▷ stage two: weight re-optimization, mask fixed
    update $\widehat{\boldsymbol{W}}_i$ with a Muon/AdamW step on $\mathcal{L}$; snapshot on improvement
end for
restore the best snapshot ▷ keep-best guard
$\widehat{\boldsymbol{X}}_{i+1} \leftarrow f_i\big(\widehat{\boldsymbol{X}}_i;\, \boldsymbol{S}_i^\star \odot \widehat{\boldsymbol{W}}_i^\star\big)$ ▷ error compensation, equation 3

D Hyperparameter Settings

The mask stage's temperature follows a geometric anneal over the T_m mask-learning steps,

$$\tau_t = \tau_{\max}\big(\tau_{\min}/\tau_{\max}\big)^{\,t/(T_\mathrm{m}-1)}, \tag{7}$$

and the logit scale rises along the mirrored geometric law over the same steps,

$$\kappa_t = \kappa_{\mathrm{start}}\big(\kappa_{\mathrm{end}}/\kappa_{\mathrm{start}}\big)^{\,t/(T_\mathrm{m}-1)}. \tag{8}$$

Table 2 lists the Optuna (Akiba et al., 2019) search spaces of both stages, and Table 3 the studies and their winning configurations. All values are selected by the first-block search of Section 2.4 and frozen for every later block. Both stages search with Optuna's tree-structured Parzen estimator sampler. Mask-stage trials each run the full step budget, with no early stopping, and are ranked on held-out hard-mask block error; weight-stage trials run under an asynchronous successive-halving schedule and may terminate early on a validation subsample.

D.1 Mask-Stage Optimizer Study

This section expands the optimizer ablation of Section 3.1: the protocol, the gradient-norm and loss trajectories (Figure 2), the discussion, and the search details.

GSQ makes the vanishing precise (Dadgarnia et al., 2026): for logits $\boldsymbol{a} \in \mathbb{R}^K$ with gap Δ between the two largest entries, temperature τ, and a loss whose gradient with respect to the relaxed output has ℓ_1 norm at most G, their Lemma B.1 gives

$$\big\|\nabla_{\boldsymbol{a}}\,\ell\big\|_\infty \;\le\; \frac{G}{\tau}\,\big(1 - y_{i^\star}\big) \;\le\; \frac{(K-1)\,G}{\tau}\,e^{-\Delta/\tau}, \tag{9}$$

where $i^\star = \arg\max_i a_i$ and $y_{i^\star}$ is the probability of the leading candidate; the logit-scale ramp raises Δ while the temperature anneal lowers τ, so the exponent grows on both counts. The literature's other workarounds are consistent with the same underlying weak signal: HyperPrune requires its supervised imitation stage plus prior biases to train stably, and MaskPro requires learning rates of 50 to 100 to make progress (Sun & Sakuma, 2026; Sun et al., 2025).

We compare Lion against AdamW, Adam, Muon, and SGD. For each of the four alternatives we run an Optuna search on the first block of Qwen2.5-0.5B at 2:4 over the shared mask-stage knobs (temperature schedule, logit-scale schedule, prior strength, and regularizer weight) jointly with the optimizer's own (learning rate, weight decay, and the Adam betas or the momentum coefficient), 80 to 130 trials per optimizer under a three-hour cap and 457 trials in total, selecting by held-out block error exactly as in the main search. Lion uses the published configuration of Table 3, itself a 343-trial search on the same block. Learning-rate windows are per optimizer, since a shared window

Table 2: Hyperparameter search spaces. Float ranges marked "log" are sampled log-uniformly. Batch size and steps are tied: the token budget is fixed and steps are rescaled so that steps × batch is constant, so every trial costs about the same.

Stage	Hyperparameter	Space	Notes
Mask	learning rate	float, log, $10^{-3} \to 0.3$	
	reg_weight	$\{0.0, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$	magnitude regularizer
	temp_start	float, $1.0 \to 8.0$	$\tau_{\max}$ of equation 7
	temp_end	float, log, temp_start $\times$ $(0.002 \to 0.5)$	fraction, so the schedule always cools
	prior_strength	float, $0.5 \to 8.0$	ρ
	logit_scale_start	float, log, $0.3 \to 20.0$	κ_{start}
	logit_scale_ramp	float, log, $1.0 \to 10.0$	end/start ratio; 1.0 = constant; κ_{end} derived
	batch size	{16, 32, 64, 128, 256}	token budget fixed; steps rescaled
	optimizer	{lion, adamw}	
	weight decay	$\{0.0, 10^{-5}, 10^{-4}, 10^{-3}\}$	
	hard	{False, True}	straight-through
Weight	learning rate	float, log, $5 \times 10^{-4} \to 5 \times 10^{-2}$	
	batch size	{16, 32, 64, 128, 256}	token budget fixed; steps rescaled
	scheduler	{none, cosine, linear, step}	
	warmup ratio	$0.0 \to 0.3$	
	min-LR ratio	$0.0 \to 0.5$	
	step_frac	$0.05 \to 0.5$	step scheduler only
	scheduler_gamma	$0.1 \to 0.9$	step scheduler only
	weight decay	$\{0.0, 10^{-5}, 10^{-4}, 10^{-3}\}$	
	muon_momentum	$0.80 \to 0.99$	

Table 3: Tuning studies and winning configurations. All studies ran on block 0 of Qwen2.5-0.5B; each pattern's winning configuration is frozen across all blocks and all models. Ten-trial re-searches on the first block of LLaMA-3.1-8B reproduced the winning configuration at both 2:4 and 4:8. The weight-stage study ran on 2:4 with the mask frozen, Muon only.

	Mask-stage studies		
	2:4 all-6	4:8 all-70	4:8 top-32
trials	343	236	204
winner trial	257	221	173
held-out block-0 MSE	1.4192×10^{-3}	1.1202×10^{-3}	1.1159×10^{-3}
learning rate	7.6298×10^{-3}	3.7113×10^{-3}	1.1498×10^{-2}
temperature $\tau_{\max} \to \tau_{\min}$	$4.336 \to 0.3178$	$5.346 \to 0.3336$	$3.098 \to 0.4061$
logit scale $\kappa_{\text{start}} \to \kappa_{\text{end}}$	$1.418 \to 1.557$	$2.744 \to 5.836$	$0.399 \to 1.402$
prior strength ρ	6.101	4.683	4.574
batch × steps	16×4800	16×4800	16×4800
optimizer	lion	lion	lion
weight decay	0.0	10^{-3}	10^{-5}
reg_weight	0.0	0.0	0.0
hard	False	False	False

	Weight-stage study (2:4, mask frozen, Muon only)
trials	163
learning rate	3.4461×10^{-3}
batch × steps	32×2400
scheduler	linear (warmup ratio 0.1427, min-LR ratio 0.0909)
weight decay	10^{-5}
muon_momentum	0.9482

would manufacture the result: SGD's step is the raw gradient, and its optimum lies eight orders of magnitude from Lion's. Each optimizer's winner is then frozen and used unchanged, first on the first block of LLaMA-3.1-8B, then on all 24 blocks of Qwen2.5-0.5B, where the resulting BLADE$^{\text{Mask}}$ model is scored end to end by KL divergence to the dense model (Table 4).

Table 4: Mask-stage optimizer study at 2:4. Each optimizer is tuned on the first block of Qwen2.5-0.5B and then frozen. Block-0 columns: reduction in held-out block-reconstruction error relative to the SparseGPT prior (%, ↑; 0.00 marks a block reverted to the prior by the guard). All-blocks columns: mean per-block reduction over the 24 blocks of Qwen2.5-0.5B, number of reverted blocks (Rev.), and exact KL divergence of the resulting BLADE$^{\text{Mask}}$ model to the dense model on C4 and WikiText-2 (↓; estimator of Appendix E).

		Block 0 (%, ↑)		All blocks, Qwen2.5-0.5B			
Optimizer	lr	Qwen2.5	LLaMA-3.1	Mean (%, ↑)	Rev.	C4 (↓)	Wiki. (↓)
Lion	7.6×10^{-3}	70.85	**63.63**	**26.82**	0	**0.377**	**0.452**
AdamW	7.6×10^{-1}	**71.17**	0.00	21.32	0	0.604	0.703
Adam	5.7	70.17	37.86	17.46	0	0.659	0.798
Muon	3.2×10^{-2}	69.37	57.48	23.65	0	0.464	0.559
SGD	7.2×10^{7}	67.49	32.18	9.52	7	0.614	0.733

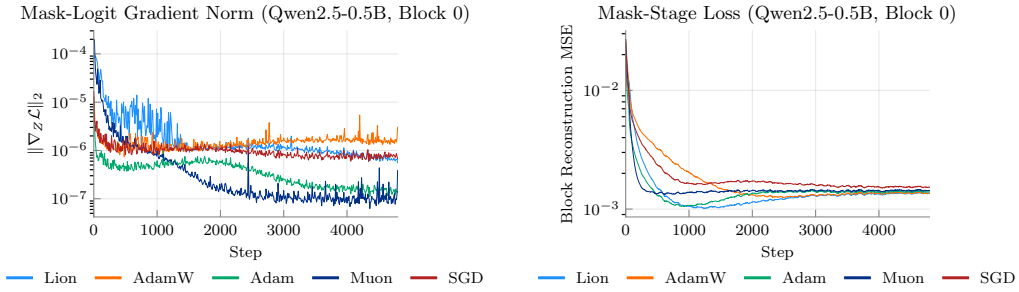


Figure 2: Mask stage on the first block of Qwen2.5-0.5B at 2:4, each optimizer at its own tuned configuration. Left: ℓ_2 norm of the mask-logit gradient, sampled every ten steps; it falls by two to three orders of magnitude under every optimizer, and the configuration with the smallest gradients (Muon) converges as well as the rest. Right: training micro-batch loss of the soft-masked block (50-step rolling mean). The dip and subsequent rise are not overfitting but the relaxation gap: at intermediate temperature the soft mask spreads keep-mass over more than N weights per group and reconstructs the block better than any hard N:M mask can, and as the anneal hardens it the loss climbs back to the hard-mask value (held-out hard-mask errors in Table 6).

On the block it is tuned on, every optimizer converges. The first column of Table 4 shows that all five optimizers cut the first block's error by 67.5 to 71.2% relative to the SparseGPT prior, a spread of 3.7 points, and AdamW lands 0.3 points ahead of Lion, a gap we do not read as a ranking. Vanishing gradients thus do not prevent convergence. What distinguishes the optimizers is what convergence costs: AdamW and Adam need learning rates of 0.76 and 5.7 against Lion's 0.0076, two to three orders of magnitude, and SGD needs 7×10^7, nine. GSQ describes the same experience in quantization, where AdamW must converge early, before the gradients shrink, through a learning rate large enough to make the search unstable and hard to tune (Dadgarnia et al., 2026).

Gradients vanish for every optimizer, Lion included, so gradient magnitude by itself explains nothing. In the left panel of Figure 2, Lion's mask-logit gradient falls from 8.5×10^{-5} to 5.6×10^{-7} over training, a $152\times$ decay, and Adam's and Muon's fall $90\times$ and $525\times$; Muon ends with the smallest gradients of any optimizer, 3.8×10^{-7}, and still matches Adam and AdamW on the final mask. AdamW's gradient, by contrast, decays only $4\times$, and SGD's $25\times$. The optimizer with the smallest gradients converges and the one with the largest final gradients is the one that fails to transfer, so the size of the gradient reaching the logits, which equation 9 bounds identically for every optimizer, is not what separates them; the rule that turns that gradient into a step is.

What separates the optimizers is transfer. Carried unchanged to the first block of LLaMA-3.1-8B, whose first-block gradients are two orders of magnitude smaller than Qwen's, Lion recovers 63.6% of the prior's error, Muon 57.5%, Adam 37.9%, and SGD 32.2%, while AdamW, the best performer on Qwen, learns nothing: its mask ends worse than the prior, the revert guard of Section 2.2 fires,

and its gradient norm grows $2.1\times$ over training rather than decaying, the signature of a mask drifting away from a good solution. The spread across optimizers widens from 3.7 points on the tuning model to 63.6 points on the transfer target, and Lion is the only one that is never worse than $1.00\times$ its own error anywhere. Transfer across depth tells the same story inside a single model: on all 24 blocks of Qwen2.5-0.5B, Lion's mean per-block error reduction is 26.8%, against 23.7% for Muon, 21.3% for AdamW, 17.5% for Adam, and 9.5% for SGD, which reverts seven blocks; end to end, Lion's BLADE$^{\text{Mask}}$ model reaches a KL divergence of 0.377 on C4 and 0.452 on WikiText-2, 38% below AdamW's 0.604 and 0.703, the optimizer that had beaten it on the first block. A first-block search therefore does not select the optimizer that produces the best model, and the tune-once recipe of Section 2.4 is viable because Lion's configuration is the one that survives being frozen; the learning-rate sweep of Table 8 confirms the same robustness from the other side, placing LLaMA-3.1-8B's own optimum within 0.1% of the transferred value.

Search details. All optimizers share the fixed setting of the mask stage (2:4, all linear layers, 4096 calibration sequences of 2048 tokens, SparseGPT prior, soft mask, 16×4800 batch–steps, seed 0) and select by held-out block-0 error on a 10% held-out split. Table 5 lists the knobs searched for every optimizer and the optimizer-specific ones; Table 6 the winning configurations, trial counts, and stopping conditions. Lion is not re-searched: its configuration is the 2:4 winner of Table 3, whose 343-trial study covered the same block and a superset of these knobs, and a ten-trial probe of its β_1 moved the block error by 0.02%. Adam's and AdamW's ϵ is fixed at the default 10^{-8} on purpose: in an earlier 70-trial AdamW search the winning ϵ wandered from 3.9×10^{-15} to 3.3×10^{-12} while the error stayed flat, and at the default AdamW still reached $0.98\times$ Lion's error, so it is not the operative knob.

Table 5: Search space of the optimizer study. The shared knobs are searched identically for every optimizer, so none inherits a sharpness schedule tuned for another update rule; learning-rate windows are per-optimizer and log-uniform. Lion is not searched: it uses the 2:4 column of Table 3 with $(\beta_1, \beta_2) = (0.9, 0.99)$.

	Hyperparameter	Space
Shared	temp_start	float, $1.0 \to 8.0$
	temp_end	float, log, temp_start $\times$ $(0.002 \to 0.5)$
	logit_scale_start	float, log, $0.3 \to 20.0$
	logit_scale_ramp	float, log, $1.0 \to 10.0$
	prior_strength ρ	float, $0.5 \to 8.0$
	reg_weight	$\{0.0, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$
	weight decay	$\{0.0, 10^{-5}, 10^{-4}, 10^{-3}\}$
AdamW, Adam	learning rate	float, log, $10^{-4} \to 10$
	β_1	float, $0.5 \to 0.99$
	β_2	float, $0.9 \to 0.9999$
Muon	learning rate	float, log, $10^{-4} \to 1$
	momentum	float, $0 \to 0.995$
SGD	learning rate	float, log, $10^{-2} \to 10^{8}$
	momentum	float, $0 \to 0.995$

Table 7 gives the ℓ_2 norm of the mask-logit gradient at the first step, its median over the 481 samples taken every ten steps, and its value at the last step, for each optimizer on the first block of both models. Two features stand out: gradient scale is a property of the model, spanning more than two orders of magnitude between LLaMA-3.1-8B and Qwen2.5-0.5B at the same step of the same schedule, and AdamW is the only optimizer whose gradient grows on LLaMA-3.1-8B, the block on which it reverted.

Finally, Table 8 is the learning-rate sweep cited in Sections 2.3 and 3.1: Lion's Qwen-tuned learning rate scaled by $\{0.25, 0.5, 1, 2, 4\}$ on the first block of LLaMA-3.1-8B at 2:4, everything else held at the transferred configuration. The optimum is interior, ties the transferred value within 0.06%, and the whole sixteenfold band spans 9.2% in held-out block error.

Table 6: Winning configurations of the optimizer study on block 0 of Qwen2.5-0.5B at 2:4, with held-out block error and search budget. AdamW's search stopped early on reaching Lion's error, so its value is an upper bound on its attainable error; SGD's learning rate was still rising against the window ceiling of 10^8. Held-out MSE is measured on this study's validation split (the last 10% of the calibration set); Table 3 reports the same Lion configuration on the original search's smaller split (the last 6.25%), hence the slightly different value. Every optimizer selected reg_weight $= 0$. The last column is (β_1, β_2) for the Adam family and Lion and the momentum coefficient for Muon and SGD.

Optimizer	Trials	Time	Held-out MSE	lr	wd	$\tau_{\max} \to \tau_{\min}$	$\kappa_{\text{start}} \to \kappa_{\text{end}}$	ρ	betas / mom.
Lion	343	–	1.3751×10^{-3}	7.63×10^{-3}	0	$4.34 \to 0.318$	$1.42 \to 1.56$	6.10	$(0.9, 0.99)$
AdamW	80	116 min	1.3571×10^{-3}	0.758	10^{-3}	$6.12 \to 0.460$	$0.481 \to 2.93$	7.81	$(0.71, 0.9927)$
Adam	129	184 min	1.4119×10^{-3}	5.72	0	$7.13 \to 0.0529$	$0.598 \to 1.14$	4.91	$(0.88, 0.9446)$
Muon	118	184 min	1.3797×10^{-3}	3.20×10^{-2}	10^{-4}	$1.58 \to 0.136$	$1.92 \to 2.22$	0.764	0.505
SGD	130	181 min	1.4615×10^{-3}	7.18×10^{7}	0	$5.08 \to 0.0260$	$0.532 \to 1.05$	4.77	0.643

Table 7: Mask-logit gradient ℓ_2 norm on block 0 (first step, median, last step) and the first-to-last decay factor, per optimizer and model, each optimizer at its Qwen-tuned configuration.

	Qwen2.5-0.5B				LLaMA-3.1-8B			
Optimizer	first	median	last	decay	first	median	last	decay
Lion	8.5×10^{-5}	1.1×10^{-6}	5.6×10^{-7}	152×	5.1×10^{-7}	3.6×10^{-9}	2.8×10^{-9}	180×
AdamW	1.3×10^{-5}	1.5×10^{-6}	3.0×10^{-6}	4.3×	8.1×10^{-8}	5.0×10^{-8}	1.7×10^{-7}	0.5×
Adam	1.3×10^{-5}	4.0×10^{-7}	1.4×10^{-7}	90×	8.5×10^{-8}	1.6×10^{-8}	2.9×10^{-8}	3.0×
Muon	2.0×10^{-4}	1.4×10^{-7}	3.8×10^{-7}	525×	1.2×10^{-6}	1.7×10^{-8}	2.4×10^{-8}	49×
SGD	1.7×10^{-5}	9.3×10^{-7}	6.7×10^{-7}	25×	1.1×10^{-7}	4.8×10^{-8}	9.9×10^{-8}	1.1×

D.2 Prior-Mask and Restricted-Search Tables

Tables 9 and 10 carry the full numbers behind the robustness checks of Section 3.1: the end-to-end prior-mask comparison and the per-prior top-32 coverage of learned full-search 4:8 patterns.

E Calibration-Set Size Study

This section expands the calibration-size ablation of Section 3.1: the protocol, the measurement, the full tables behind Figure 3, and the analysis of the two scaling policies.

Protocol. Qwen2.5-0.5B is compressed to 2:4 on all linear layers from 128, 512, 2048, and 4096 SlimPajama calibration sequences of 2048 tokens. The calibration windows are drawn deterministically per size (seed 0) and are byte-identical across methods at a given size, so every method at a size sees exactly the same data; the four sizes are independent draws, not nested subsets. BLADE runs with the tuned 2:4 configuration of Table 3 unchanged, as BLADE$^{\text{Mask}}$ and BLADE$^{\text{Full}}$, under two scaling policies. Under *strong scaling* the step budget is pinned at its 4096-sequence value (4800 mask steps, 1280 weight steps), so a smaller set is revisited more often, up to 600 mask epochs at 128 sequences. Under *weak scaling* the epoch count is pinned instead (18.75 mask epochs, 20 weight epochs), so the step budget falls with the data, to 150 mask and 40 weight steps at 128 sequences; the temperature and logit-scale schedules are defined on the fraction of steps completed, so a weak-scaling run traverses the full schedule in its smaller budget rather than truncating it. At 4096 sequences the two policies coincide, and a single run serves both. The one-shot pruners Wanda, SparseGPT, and ADMM are run at every size; they have no iteration budget, so the scaling distinction does not apply to them.

Measurement. Every model is scored by the exact full-vocabulary KL divergence $\mathrm{KL}(p_{\text{dense}} \,\|\, p_{\text{sparse}})$ in nats per token on three sets: the full WikiText-2 test split (298,792 tokens), the first 20,000 documents of the C4 validation split (9.39M tokens), and each run's own calibration windows, scored from the dumped token ids. The C4 subset is converged: across 1k, 4k, 10k, 20k, and 50k documents the estimate moves non-monotonically within a 1.2% band, and every checkpoint is scored on the identical subset, so between-run differences carry no subsampling error.

Table 8: Lion learning-rate sweep on block 0 of LLaMA-3.1-8B at 2:4, around the learning rate transferred from Qwen2.5-0.5B (7.63×10^{-3}). Held-out block-reconstruction MSE (↓). Note the 10^{-5} scale: LLaMA-3.1-8B's first-block reconstruction errors sit two orders of magnitude below Qwen2.5-0.5B's (Table 3), consistent with its two orders of magnitude smaller first-block gradients (Appendix D.1).

lr multiple	$0.25\times$	$0.5\times$	$1\times$ (transferred)	$2\times$	$4\times$
Held-out MSE ($\times 10^{-5}$)	1.2880	1.2407	1.2414	1.2751	1.3543

Table 9: Prior-mask ablation on Qwen2.5-0.5B: every block compressed with the tuned configuration under each prior. Eight-task zero-shot average (↑) and KL divergence to the dense model on WikiText-2 (↓). Per-task results: Appendix K.

	2:4		4:8	
Prior	Avg. acc. (↑)	KLD (↓)	Avg. acc. (↑)	KLD (↓)
SparseGPT (default)	37.80	0.404	39.31	0.304
Magnitude	37.95	0.400	39.60	0.303
Wanda	38.04	0.408	38.79	0.310
ADMM	37.97	0.397	39.25	0.304

The noise floor from two independent runs of the identical 4096-sequence configuration is 0.54% on C4 and 0.85% on WikiText-2; nothing below it is called a result.

Only BLADE benefits from more data. Table 11 holds the values plotted in Figure 3. From 128 to 4096 sequences the one-shot methods move by -0.8% (Wanda), -5.0% (ADMM), and -7.7% (SparseGPT) on C4, while BLADE$^{\text{Full}}$ improves by 49% and BLADE$^{\text{Mask}}$ by 19%. At 4096 sequences BLADE$^{\text{Full}}$'s C4 KL of 0.294 is $2.7\times$ below ADMM (0.785), $2.9\times$ below SparseGPT (0.847), and $6.5\times$ below Wanda (1.924). WikiText-2 is uniformly harsher than C4 for every method, by 15 to 30%: calibration is SlimPajama, so C4 is the nearer held-out distribution and WikiText-2 the stricter test, but the two corpora agree on every ordering.

Memorization. The calibration-set column of Table 11 runs opposite to the held-out columns for every trained run: as the set grows, BLADE$^{\text{Mask}}$ fits it worse (0.112 to 0.281) and generalizes better (0.466 to 0.378). Table 12 reports the ratio of C4 KL to calibration-set KL, which is one for a model that treats its calibration data like any other text. The one-shot methods sit at 1.07 to $1.26\times$ at every size, since they use the data only for activation statistics; against that floor BLADE$^{\text{Full}}$ reaches $10.2\times$ and BLADE$^{\text{Mask}}$ $4.2\times$ at 128 sequences, falling to $1.63\times$ and $1.35\times$ at 4096. Model, pattern, evaluation sets, and calibration windows are identical across rows, so the span isolates memorization as the mechanism. The weak-scaling BLADE$^{\text{Mask}}$ run reads $1.18\times$ at 128 sequences not because it is robust but because it learned nothing: all 24 of its blocks reverted to the SparseGPT prior, so it is an untrained model sitting on the one-shot floor.

Block reconstruction is an inverted proxy along the data axis. Under strong scaling, held-out KL and mean per-block reconstruction error rank the eight BLADE runs in exactly opposite orders with no ties: the mean block-error reduction over the prior grows as the set shrinks, from 27.3% to 40.4% for BLADE$^{\text{Mask}}$ and from 30.3% to 46.1% for BLADE$^{\text{Full}}$ between 4096 and 128 sequences, while held-out KL worsens throughout. The 128-sequence BLADE$^{\text{Full}}$ run has the best reconstruction of all eight and the worst model. Along the compute axis, by contrast, the two agree: weak-scaling runs have both worse reconstruction and worse KL than their strong-scaling counterparts. The precise statement is therefore that less data at fixed compute buys a better fit and a worse model, whereas less compute at fixed data makes both worse. The optimizer study of Section 3.1 reached the same warning about block error as a selection criterion from a different direction.

The two stages want opposite scaling policies. Table 13 compares the two policies. Mask learning is iteration-bound, not data-bound: the three weak-scaling BLADE$^{\text{Mask}}$ runs all ran exactly 18.75 epochs and produced C4 KLs of 0.392, 0.532, and 1.992, a fivefold spread driven purely by step count; halving the steps at 2048 sequences costs 3.5%, an eightfold cut at 512 costs 34%, and at 150 steps the keep-better guard rejects the learned mask in all 24 blocks, so the checkpoint is the

Table 10: Share of groups (%, ↑) whose learned full-search 4:8 pattern lies within each prior's top-32 saliency-ranked candidates. Headline numbers quote the SparseGPT column, the default prior.

Model	Wanda	SparseGPT	ADMM
Qwen2.5-0.5B	99.50	99.48	99.44
LLaMA-3.1-8B	95.03	94.65	95.00

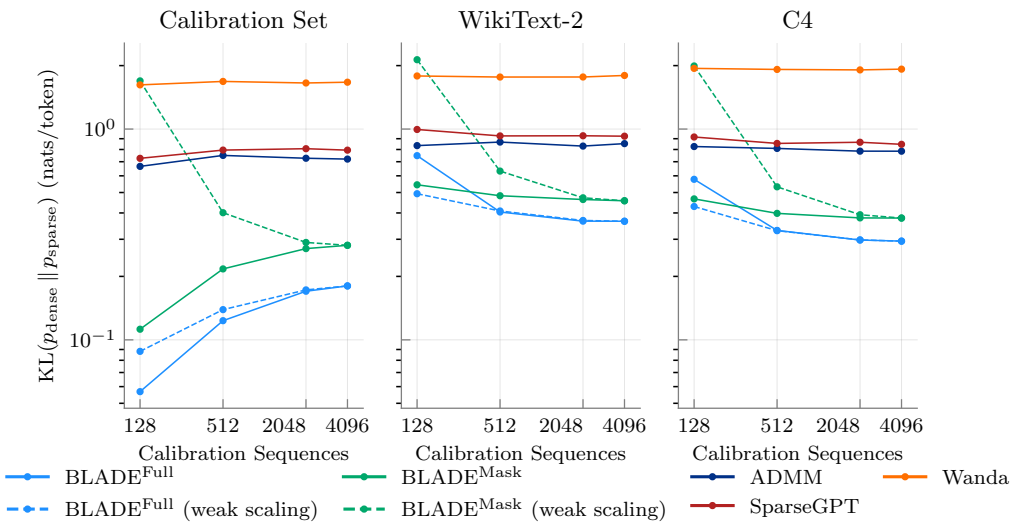


Figure 3: KL divergence to the dense model versus calibration-set size on Qwen2.5-0.5B at 2:4, evaluated on each model's own calibration windows (left) and on two held-out corpora (center, right); lower is better. Solid BLADE lines hold the step budget fixed as the set shrinks (strong scaling); dashed lines hold epochs fixed instead (weak scaling); the two coincide at 4096. One-shot methods are flat on every panel. BLADE improves with data on the held-out corpora while its fit to the calibration windows loosens, the signature of memorization at small sizes.

untrained prior. The cliff lies between 150 and 600 steps and is bracketed, not resolved. Weight re-optimization is the exact inverse, iteration-cheap and overfit-prone: 640 steps match 1280 within noise at 2048 sequences, and at 128 sequences cutting from 1280 to 40 steps *improves* the model by 25.7% on C4 and 34.0% on WikiText-2. The strong-scaling collapse of BLADE$^{\mathrm{Full}}$ at 128 sequences is therefore 640 epochs of weight fitting, not a property of the weight stage on small data. At 128 sequences the best policy learns no mask at all: the reverted prior mask plus 40 weight steps (C4 0.429) beats 4800 steps of mask learning alone (0.466) and the full-budget two-stage recipe (0.578). On a small calibration set, compute spent on weights dominates compute spent on masks.

The 4096-sequence recipe is exactly provisioned. Strong and weak scaling coincide at 4096 sequences by construction. Halving the set to 2048 costs $+0.16\%$ (BLADE$^{\mathrm{Mask}}$) and $+1.22\%$ (BLADE$^{\mathrm{Full}}$) on C4, at or below the noise floor, while halving activation memory and setup time; every further reduction along either axis costs something clearly measurable. The recipe's 4800/1280 split sits where mask learning has just enough iterations and weight re-optimization has not yet begun to memorize.

Scope. One model, one sparsity pattern, and one seed per cell. The 128-sequence rows are the most dramatic and the least sampled. The thresholds above are Qwen2.5-0.5B numbers until tested on a second model.

Table 11: KL divergence to the dense model (↓, nats/token) on Qwen2.5-0.5B at 2:4, for every method and calibration-set size, on the three evaluation sets of Figure 3. Weak-scaling BLADE entries coincide with the strong-scaling runs at 4096 sequences.

	Calibration set				WikiText-2				C4			
Method	128	512	2048	4096	128	512	2048	4096	128	512	2048	4096
BLADE$^{\text{Full}}$	0.0568	0.1233	0.1701	0.1802	0.7480	0.4041	0.3657	0.3653	0.5776	0.3305	0.2976	0.2940
BLADE$^{\text{Full}}$, weak scaling	0.0882	0.1391	0.1726	0.1802	0.4935	0.4087	0.3684	0.3653	0.4292	0.3296	0.2979	0.2940
BLADE$^{\text{Mask}}$	0.1123	0.2170	0.2710	0.2809	0.5440	0.4827	0.4632	0.4569	0.4664	0.3982	0.3789	0.3783
BLADE$^{\text{Mask}}$, weak scaling	1.6925	0.4010	0.2898	0.2809	2.1334	0.6319	0.4720	0.4569	1.9922	0.5321	0.3922	0.3783
ADMM	0.6652	0.7498	0.7270	0.7203	0.8346	0.8675	0.8304	0.8529	0.8262	0.8093	0.7856	0.7852
SparseGPT	0.7264	0.7948	0.8070	0.7942	0.9956	0.9279	0.9294	0.9251	0.9170	0.8548	0.8658	0.8466
Wanda	1.6203	1.6825	1.6543	1.6684	1.7859	1.7652	1.7662	1.7958	1.9403	1.9182	1.9084	1.9243

Table 12: Memorization ratio, C4 KL divided by calibration-set KL, per method and calibration-set size. A ratio near one means the model fits its calibration windows no better than unseen text.

Method	128	512	2048	4096
BLADE$^{\text{Full}}$	10.17×	2.68×	1.75×	1.63×
BLADE$^{\text{Full}}$, weak scaling	4.87×	2.37×	1.73×	1.63×
BLADE$^{\text{Mask}}$	4.15×	1.83×	1.40×	1.35×
BLADE$^{\text{Mask}}$, weak scaling	1.18×	1.33×	1.35×	1.35×
ADMM	1.24×	1.08×	1.08×	1.09×
SparseGPT	1.26×	1.08×	1.07×	1.07×
Wanda	1.20×	1.14×	1.15×	1.15×

F JOINT VERSUS TWO-STAGE OPTIMIZATION

Joint optimization wins the objective it is tuned on and loses the model. Training the mask logits and the block weights simultaneously, with its own 168-trial search on the first block of Qwen2.5-0.5B at 2:4, reaches a 0.31% lower first-block error than BLADE$^{\text{Full}}$, then wins that block and no other (BLADE$^{\text{Full}}$ is lower on 23 of 24 blocks) and ends 23% worse in held-out KL divergence (0.362 against 0.293 on C4), within a few percent of BLADE$^{\text{Mask}}$, which trains no weights at all; it is also unstable, with 34% of searched configurations ending worse than plain mask-only. The mechanism is that the weights are fit to a mask that is both moving and softened, so the compensation they learn holds only for the soft mixture and is lost when the mask is baked. This section details the implementation, the search, the measurement basis, and the full results.

Joint mode trains the mask logits and the block's weights in one loop against the block-reconstruction loss of equation 2, in two parameter groups: the logits under Lion at the mask learning rate, the weights under a separately searched optimizer and learning rate. The soft mask of equation 5 multiplies the weights in the forward pass exactly as in stage one, so the weights see the relaxed mask throughout training and the hard mask only at bake time. Three details were necessary for the comparison to be genuine. First, the block's weights are frozen at two independent points in the two-stage code path (by the caller for stage one, and by the masked-linear wrapper whenever a mask module is attached); lifting only the first leaves the optimizer holding the two normalization layers, 1,792 of 14,912,384 elements, while every log line looks healthy, so joint mode counts its trainable weight elements and aborts if the weight group is empty. Second, trials mutate weights in place, so every search trial restores a pristine snapshot of the block before training and the winning trial's weights are captured together with its mask. Third, the revert-to-prior guard of Section 2.2 rolls the weights back alongside the mask, since a reverted mask over jointly-fitted weights is worse than the prior it falls back to.

The joint configuration is tuned at least as thoroughly as the two-stage recipe it is compared against. It receives a 168-trial, four-hour Optuna search on the first block of Qwen2.5-0.5B at 2:4, ranked on a 6.25% held-out split so that a configuration that merely memorizes its training windows cannot win. Table 15 lists the space: the six mask-stage knobs are re-searched rather than frozen at the two-stage optimum, because a moving weight tensor changes what the mask should do and freezing them would handicap the joint search; the four joint-only knobs are the weight learning rate, weight

Table 13: Strong versus weak scaling of the BLADE budget on Qwen2.5-0.5B at 2:4. Steps and implied epochs of each stage, held-out C4 KL (↓), and the number of blocks whose learned mask reverted to the prior.

		BLADE$^{\text{Mask}}$			BLADE$^{\text{Full}}$		
Sequences	Scaling	Steps (epochs)	KL C4	Reverted	Weight steps (epochs)	KL C4	Reverted
128	strong	4800 (600)	0.4664	0/24	1280 (640)	0.5776	0/24
128	weak	150 (18.75)	1.9922	24/24	40 (20)	0.4292	0/24
512	strong	4800 (150)	0.3982	0/24	1280 (160)	0.3305	0/24
512	weak	600 (18.75)	0.5321	0/24	160 (20)	0.3296	0/24
2048	strong	4800 (37.5)	0.3789	0/24	1280 (40)	0.2976	0/24
2048	weak	2400 (18.75)	0.3922	0/24	640 (20)	0.2979	0/24
4096	both	4800 (18.75)	0.3783	0/24	1280 (20)	0.2940	0/24

Table 14: Joint versus two-stage optimization on Qwen2.5-0.5B at 2:4. The joint configuration is tuned on block 0 with a 168-trial search; all rows use 4096 calibration sequences. Block-0 error is the hard-mask reconstruction MSE (↓); blocks won counts the blocks on which each row attains the lower error; KL divergence to the dense model is exact (↓); the last column is the ratio of C4 KL to KL on its own calibration windows.

	Block-0 MSE	Blocks won	Mean drop (%, ↑)	KL C4	KL Wiki.	C4 / calib.
BLADE$^{\text{Mask}}$	1.397×10^{-3}	–	27.3	0.378	0.454	1.34×
Joint	$\mathbf{9.270 \times 10^{-4}}$	1/24	32.5	0.362	0.459	1.63×
BLADE$^{\text{Full}}$	9.299×10^{-4}	**23/24**	**35.5**	**0.293**	**0.372**	1.63×

decay, gradient clipping, and weight optimizer. Everything else is held at the two-stage recipe (4800 steps, batch 16, Lion for the logits, SparseGPT prior, 4096 calibration sequences), so only the joint flag and the tuned knobs differ from the controls.

Table 15: Search space and winning configuration of the joint search on block 0 of Qwen2.5-0.5B at 2:4 (168 trials). The last column gives the range of each knob over the 111 trials that at least matched plain mask-only.

	Hyperparameter	Space	Winner	Range over usable trials
Mask	learning rate	float, log, $10^{-3} \to 0.3$	8.271×10^{-3}	$1.53 \times 10^{-3} \to 0.2$
	reg_weight	$\{0, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$	10^{-7}	$0 \to 10^{-4}$
	temp_start	float, $1.0 \to 8.0$	7.263	$1.99 \to 8$
	temp_end	float, log, temp_start × $(0.002 \to 0.5)$	$\times 0.0667$	$\times(0.00212 \to 0.451)$
	prior_strength ρ	float, $0.5 \to 8.0$	7.612	$0.729 \to 7.99$
	logit_scale_start	float, log, $0.3 \to 20.0$	2.368	$0.319 \to 15.8$
Weights	learning rate	float, log, $10^{-7} \to 10^{-3}$	1.291×10^{-4}	$1.1 \times 10^{-5} \to 6.78 \times 10^{-4}$
	weight decay	$\{0, 10^{-5}, 10^{-4}, 10^{-3}\}$	10^{-3}	$0 \to 10^{-3}$
	gradient clip	{off, 0.1, 1.0, 10.0}	1.0	off → 10
	optimizer	{AdamW, Adam, SGD}	SGD	all three

The search landscape is itself the instability result. The best of 168 trials reaches a held-out block-0 error of 8.789×10^{-4}, the first quartile 1.005×10^{-3}, the median 1.202×10^{-3}, and the worst 2.333×10^{-2}. Of the 168 trials, 57 (34%) are worse than plain mask-only and 5 are worse than the untrained prior, the worst by a factor of five. The tree-structured Parzen sampler barely helped: the first trial, a random draw, led for 153 trials, and the eventual winner improved on it by 0.3%. The usable weight learning rates span barely over a decade inside a four-decade window, but the failed trials span the whole window and overlap the usable band completely, so no single knob separates success from failure; the interaction is what makes joint training hard to tune. Gradient clipping does not change this: a clip of 1.0 gives the best single trial, but unclipped trials have the better median and the most aggressive clip (0.1) is worst on both counts, so the failure mode is not an occasional gradient explosion. The winning weight optimizer, SGD, is the least-sampled option and not the most reliable one: usable-trial rates are 68% for AdamW (40 of 59), 69% for Adam (50 of 72), and 57% for SGD (21 of 37), so one winning trial supports a higher ceiling for SGD, not a safer floor.

The comparison must be made on a common measurement basis, and the search's own numbers are not. The search trains on 3840 sequences and reports block error on the 256 held-out ones, whereas the two-stage controls train and report on all 4096; comparing across that difference inflates the joint run's block-0 win from 0.31% to 5.48%, since even the untrained prior reads 4.7206×10^{-3} on the held-out split against 4.7904×10^{-3} on the full set. The deliverable joint run therefore reproduces the controls' conditions exactly, full 4096 sequences and no held-out split, changing only the joint flag and the tuned knobs. Both controls, BLADE$^{\text{Mask}}$ and BLADE$^{\text{Full}}$, are fresh runs and reproduce the calibration study's reference values to 0.02% on block 0, inside the 0.04% noise floor.

Like for like, the joint run's block-0 error is 9.270×10^{-4} against 9.299×10^{-4} for BLADE$^{\text{Full}}$ and 1.3966×10^{-3} for BLADE$^{\text{Mask}}$, a 0.31% win over BLADE$^{\text{Full}}$ that is about eight times the noise floor. Over all 24 blocks, BLADE$^{\text{Full}}$ has the lower error on 23, with the joint run's error exceeding it by a median factor of 1.18 and at worst 1.33; the mean per-block error reduction over the prior is 35.5% for BLADE$^{\text{Full}}$, 32.5% for the joint run, and 27.3% for BLADE$^{\text{Mask}}$. The joint run beats BLADE$^{\text{Mask}}$ on all 24 blocks, so it sits cleanly between the two stages: training weights jointly helps, but less than training them afterwards. No block reverted under the tuned configuration; its failure is steady mild under-performance, not the divergence the worst search trials reach. Table 16 gives the exact held-out KL: the joint model is 23.3% worse than BLADE$^{\text{Full}}$ on C4 and 23.5% worse on WikiText-2, and against BLADE$^{\text{Mask}}$ it is a wash, 4.4% better on C4 and 1.1% worse on WikiText-2. The search run's own checkpoint, trained on 6% fewer windows under the same configuration, differs from the deliverable by 7.1% in C4 KL, against a two-stage run-to-run floor of 0.54%. The memorization ratio of the joint run, 1.63×, is identical to BLADE$^{\text{Full}}$'s, so its deficit is under-fitting rather than memorization.

Table 16: Exact KL divergence to the dense model (↓, nats/token) for the joint run and the two-stage controls on Qwen2.5-0.5B at 2:4, on two held-out corpora and each model's own calibration windows, with the memorization ratio (C4 over calibration). The search-run variant differs from the deliverable only in training on 3840 rather than 4096 windows.

Model	C4	WikiText-2	Calibration set	C4 / calib.
BLADE$^{\text{Mask}}$	0.3783	0.4540	0.2818	1.34×
Joint (deliverable)	0.3616	0.4590	0.2217	1.63×
Joint (search-run variant)	0.3375	0.4216	0.2076	1.63×
BLADE$^{\text{Full}}$	**0.2932**	**0.3716**	0.1800	1.63×

G Compression-Cost Details

This section details the measurement behind Section 3.1 and reads the regularities out of Table 17. The memory model is as follows: at every moment of training exactly one block is resident on the GPU, together with its calibration activations, which are staged on the device when they fit in spare memory and streamed from the host otherwise; the rest of the model is parked in host memory, and the calibration activations live in a host-side store that can be memory-mapped onto local NVMe for calibration sets that outgrow RAM. Total compression cost is a product of known quantities, blocks times steps per stage times step time.

Costs are computed from per-block rates taken between consecutive per-block metric timestamps, which excludes model load, calibration capture, and evaluation by construction; the runs used for timing disable checkpoint upload and held-out evaluation, so nothing but compression is timed. Setup adds 26 to 50 minutes per run, under 4% of any full run, and is reported separately rather than folded into the hours. Peak GPU memory is the allocator watermark (`torch.cuda.max_memory_allocated`) rather than a sampled system metric, since sampling misses the short-lived spikes that decide whether a run fits; reserved memory, what the card must actually hold, runs 2% above allocated on the small models and 8 to 12% above at 4:8, where many large short-lived mask tensors fragment the allocator pool. Host RSS is the peak resident set of the compression process, and disk is the size of the memory-mapped activation store described at the top of this section.

The entire additional cost of 4:8 over 2:4 sits in the mask stage. Across all three models the weight stage's per-block rate differs by at most 0.9% between patterns, since fixed-step weight fine-tuning

Table 17: Compression cost per model and sparsity pattern on a single RTX PRO 6000: wall-clock hours (↓, one-time setup excluded) for BLADE$^{\text{Mask}}$ (stage one) and BLADE$^{\text{Full}}$ (both stages), peak allocated GPU memory, peak host RSS, and the on-disk activation store. Reserved GPU memory runs 2% above allocated on the small models and 8 to 12% above at 4:8.

Model	Pattern	BLADE$^{\text{Mask}}$ (h)	BLADE$^{\text{Full}}$ (h)	GPU (GiB)	Host RSS (GiB)	Disk (GiB)
Qwen2.5-0.5B	2:4	0.84	2.0	30.5	75.3	122.8
	4:8	1.1	2.1	31.7	75.4	138.6
LLaMA-2-7B	2:4	11.8	22.8	13.4	239.2	320.5
	4:8	15.9	26.8	33.2	211.2	256.5
LLaMA-3.1-8B	2:4	12.7	23.0	14.7	239.1	386.9
	4:8	17.3	27.5	41.4	239.1	322.9

Table 18: Block-granularity ablation on Qwen2.5-0.5B at 2:4, mask stage only. Accuracy, perplexity, and held-out KL divergence aggregate as in Table 1; the per-unit reconstruction errors are in Figure 4.

Layers per unit	Acc. (%, ↑)	PPL (↓)	KLD (↓)
1 (default)	37.51	52.8	0.403
2	37.95	51.5	0.383
4	**37.96**	**50.6**	**0.371**

never sees the candidate set; the mask stage pays for $K{=}70$ candidates against six, at $1.34\times$ and $1.36\times$ the 2:4 per-block rate on the LLaMA models. The extra logits are a fixed cost per masked weight, so what varies is how large that cost is against the block's own matrix multiplies, and masked parameters per block, not total parameter count, predict where a model lands: LLaMA-2-7B and LLaMA-3.1-8B cost nearly the same per block despite the parameter gap, because LLaMA-3.1's $1.30\times$ larger MLP is offset by its grouped-query attention cutting attention parameters to $0.625\times$; masked parameters per block differ by 7.8%, against a measured 7.7% difference in per-block rate.

Each resource in Table 17 has a different owner. Peak GPU memory is block-local (Section 3.1) and, at 4:8, is dominated by the mask logits rather than the activations: LLaMA-3.1-8B rises from 14.7 to 41.4 GiB between patterns on identical weights, which is why the restricted search of Section 2.3 exists at all. Host RSS is owned by the calibration activations: on the 7B to 8B models they no longer fit on the device and stay in host memory, driving RSS to about 239 GiB against the node's 256, the binding resource at that scale, whereas Qwen2.5-0.5B stages them on-device, which is why its GPU figure exceeds LLaMA-2-7B's. Disk holds the activation store, 123 to 387 GiB at this calibration budget; it can be memory-mapped from local NVMe, so it never has to fit in RAM.

H Block-Granularity Study

BLADE's unit of compression is a span of consecutive transformer layers, and in the limit of a single unit spanning every layer the mask stage reduces, in principle, to MaskLLM's end-to-end mask learning at MaskLLM's cost, so granularity interpolates between block-local BLADE and MaskLLM. Compressing Qwen2.5-0.5B at 2:4 with the mask stage at one (the default), two, and four layers per unit, coarser units buy a modest, monotone distributional gain: four-layer units cut held-out KL divergence by 7.2% and 8.5% on the two corpora and perplexity falls in step, while the eight-task average moves within sampling noise (Table 18). This appendix backs that finding with the protocol and searched configurations (Table 19), the like-for-like per-block comparison (Figure 4), the end-to-end evaluation (Table 20), and the caveats that bound the conclusion.

At granularity N, BLADE compresses N consecutive transformer layers as one unit: the span's mask logits are trained jointly against the span's dense output under the otherwise unchanged error-compensated schedule of Section 2.2, so reconstruction error made early in a span can be absorbed by its later layers, and granularity one is byte-identical to the default path. The study compresses Qwen2.5-0.5B (24 blocks) at 2:4 with the mask stage only, from the SparseGPT prior at the standard

4096×2048 calibration budget, on two RTX PRO 6000 GPUs. Granularity barely moves peak GPU memory at this scale: 30.5 GiB at one layer per unit, 30.9 at two, and 33.4 at four.

Every granularity runs hyperparameters tuned for it. For two and four layers per unit we search ten mask-stage knobs with Optuna's multivariate TPE sampler for 100 trials each, selecting on held-out reconstruction MSE over 256 of the 4096 calibration sequences; the search runs on the first unit only and its winner is frozen for every later unit, the token budget is held at 16×4800 per unit with steps rescaled when a trial changes the batch size, and the optimizer is pinned to Lion. The one-layer control is not re-searched: it uses the incumbent 2:4 recipe of Appendix D, itself the winner of a 343-trial campaign, so the budgets are not matched (343 against 100 trials, at 2.54 minutes per trial for two-layer units and 5.04 for four-layer, 4.2 and 8.4 search hours). An earlier three-hour round that also searched the optimizer picked AdamW at both granularities, consistent with Section 3.1's finding that a first-unit search can prefer AdamW; it was superseded by the Lion-pinned 100-trial round, and its four-layer winner trails Lion's (1.9798×10^{-2} against 1.2917×10^{-2} held-out MSE) on a third of the trials, a budget confound rather than an optimizer result.

Table 19 lists the winners. Both choose soft Gumbel sampling, the smallest batch, and a starting temperature near the top of the range, echoing the incumbent's choices; where they differ most is the learning rate, 1.28×10^{-2} at two layers per unit against 1.10×10^{-3} at four, bracketing the incumbent's 7.63×10^{-3}.

Table 19: Winning mask-stage configurations for the two searched granularities (Optuna TPE, 100 trials each, Lion pinned, selection on held-out reconstruction MSE of the first unit). The one-layer control uses the incumbent recipe of Appendix D.

	2 layers/unit	4 layers/unit
learning rate	0.0128195	0.00109786
temperature start	7.54058	7.80422
temperature end fraction	0.00657527	0.0532861
logit scale start	0.59622	2.33155
logit scale ramp	3.9858	3.44996
prior strength	5.10318	2.85862
batch size	16	16
weight decay	10^{-4}	10^{-3}
regularization weight	0	0
hard (straight-through) sampling	False	False
held-out first-unit MSE	3.1183×10^{-3}	1.2917×10^{-2}
minutes per trial	2.54	5.04
search wall-clock (h)	4.2	8.4

Mask learning improves every unit at every granularity (24, 12, and 6 units). A unit's error is measured at the output of the last block it spans, so the three series align only there, and a per-series mean is not a valid comparison: it averages different blocks, the finest granularity samples shallow low-error blocks the coarse ones never measure, and it points the wrong way (0.0898, 0.1014, and 0.1202 for one, two, and four layers per unit). The comparison therefore stands only at the six blocks where all three series have a point (Figure 4): two-layer units fit worse than the control at the four shallower shared blocks (error ratios 1.630 down to 1.156 at blocks 3 to 15) and cross below it only at block 19, a geometric-mean ratio of 1.194, while four-layer units fit better everywhere except block 3 (ratios 0.856 to 0.931, and 1.058 at block 3), a geometric mean of 0.912. The first unit is an outlier by construction: at four layers the prior's span error is 8.2749×10^{-1} against 1.2917×10^{-2} after learning, a 98.4% drop, because the one-shot prior selects masks layer by layer and its errors compound uncorrected across the span; the first unit is also the only tuned one, so it improves far more than the frozen rest.

End to end, quality improves monotonically with span length (Table 20; the evaluation protocol and KL estimator of Appendix K, with the one-layer row being exactly the BLADE$^{\text{Mask}}$ row of Table 1). KL divergence to the dense model falls on every corpus and perplexity falls in step, while the eight-task average moves by at most 0.45% points, within the sampling noise of roughly 0.4% points per average (Section 3.1). The gain is largest on the calibration stream, a 20.4% drop against 7.2 to 8.5% held out, so the ratio of held-out to calibration KL, the memorization measure of Section 3.1, rises

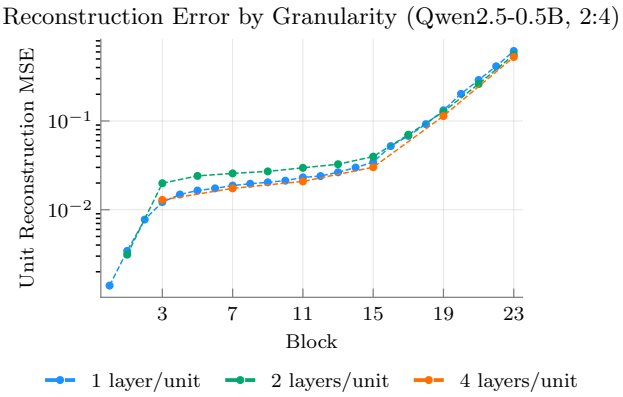


Figure 4: Per-unit reconstruction MSE by granularity on Qwen2.5-0.5B at 2:4, mask stage only (↓, log scale). Each point is one unit, plotted at the last block it spans, where its error is measured: every block at one layer per unit, every other at two, every fourth at four. Series are comparable only at blocks where they coincide.

from $1.35\times$ through $1.46\times$ to $1.56\times$: longer spans convert part of their extra freedom into a tighter fit of the calibration distribution.

Table 20: End-to-end evaluation by granularity on Qwen2.5-0.5B at 2:4, mask stage only: eight-task zero-shot average (%, ↑), perplexity, and KL divergence to the dense model (↓; nats/token, top-512 lower bound; protocol of Appendix K). The one-layer row is the BLADE$^{\text{Mask}}$ entry of Table 1.

		PPL (↓)		KLD (↓)		
Layers/unit	Acc. (↑)	WikiText-2	C4	WikiText-2	C4	Calib.
Dense	46.06	19.65	46.15	–	–	–
1 (default)	37.51	30.99	74.53	0.4400	0.3654	0.2702
2	37.95	30.34	72.56	0.4201	0.3449	0.2360
4	**37.96**	**29.97**	**71.28**	**0.4085**	**0.3344**	**0.2150**

Four caveats bound the conclusion. The study is mask-stage only, and the weight stage may exploit a span-level target differently. It covers one model and one pattern, Qwen2.5-0.5B at 2:4, whose activations fit on-device and whose four-layer spans are still a small problem; at 7B and beyond the memory and optimization pictures may both change. The search budgets are unmatched, 343 incumbent trials against 100 per searched granularity. And the downstream signal is flat: Qwen2.5-0.5B is the least capable model in our suite, its compressed accuracy sits close to chance, and at this scale the task suite is too noisy to confirm or deny an 8% KL movement, which is why the gain shows in KL divergence and perplexity instead; larger models may respond differently. The finer direction, solving the attention projections and the MLP as separate units, was not run and remains open. Given all this, the paper's default stays at one layer per unit, with coarser spans recorded as a cheap distributional win at equal memory on the one model tested.

I MaskPro Reproduction

The MaskPro rows of Table 1 are our own reproduction of Sun et al. (2025) under BLADE's conditions: the authors' official implementation, run on the three models at 2:4 and 4:8, calibrated on the same corpus as every method we train with the same sampling policy (Section 3), with the masked layer set verified to match our compression recipes module for module, and the resulting checkpoints evaluated through the identical pipeline. Following the paper's own prescription, each run learns mask logits over frozen dense weights, initialized from the mask of the corresponding SparseGPT run, at the paper's headline budget of 320,000 sequences of 128 tokens (batch 32, 10,000 steps) with

logit magnitude $C = 10$ and smoothing $\alpha = 0.99$; the repository's 512-sequence default is one point of the paper's ablation, not its headline configuration.

The released code is hard-wired to LLaMA-2-7B at 2:4, so running this suite required generalization, and we kept the arithmetic verifiably intact. The released policy-gradient step implements the closed form of the $N{=}2$ case only, so no 4:8 masks can be produced from it; we implemented the paper's general-N estimator as a subset dynamic program and verified it against brute-force enumeration over orderings to 4×10^{-15} relative error, and against the released closed form at 2:4 to 1.2×10^{-16}, so the 2:4 runs compute bit for bit what the release computes. The model, data-type, and layer-targeting generalizations carry their own checks, and every run starts by confirming that the freshly installed prior mask reproduces its recorded baseline loss exactly.

Hyperparameter transfer is where the reproduction becomes expensive, and it is the central finding of this appendix. For LLaMA-2-7B we use the exact configuration available in the authors' code base, and it trains as intended. For LLaMA-3.1-8B we began from the configuration the authors provide for LLaMA-3-8B to test whether it transfers, and it does not: at its learning rate of 50, a full 10,000-step run moved one group in 1.74 billion, a null result; among the rates we then searched (50, 100, and 200 at 2:4; 50 and 100 at 4:8), 100 trains at both patterns and is what we use, while 200 destroys the run, with logit magnitudes growing to 4.8×10^4 and 92% of groups flipped. On Qwen2.5-0.5B the published grid, 25 to 200, fails at the other end: its floor of 25 diverges outright, and among the searched rates (1, 3, 10, and 25) only 3 trains well at both patterns, with 1 near-null and 10 already running away. The usable learning rate therefore spans 3 to 100 across three models, with roughly a factor of two separating a null run from a destroyed one.

A broader search is not affordable, which bounds what the retuning can claim. One training run costs nine to eleven hours at the 7B to 8B scale (0.6 hours on the 0.5B), so bracketing even the single learning-rate axis costs days per model; the logit magnitude C and smoothing α therefore keep the paper's values, and we add two divergence guards that abort an already-destroyed run without altering the update. **A wider per-model search might improve MaskPro's rows and is untested here.** This asymmetry is exactly what the optimizer study of Section 3.1 is about: **a mask-learning recipe whose hyperparameters do not transfer across models must be re-searched at full training cost on every new model, whereas BLADE tunes once on a single block of the smallest model and transfers the winner unchanged**.

Scale is bounded by a global logit state. MaskPro holds one `fp32` logit per masked weight for the whole model at once, so its footprint is global rather than block-local: measured peak allocated GPU memory is 8.8 GiB on Qwen2.5-0.5B, 55.5 on LLaMA-2-7B, and 61.2 on LLaMA-3.1-8B, already about two thirds of a 96 GiB device at the 8B scale. For comparison, BLADE's block-local peak on LLaMA-2-7B at 2:4 is 13.4 GiB (Table 17).

J HyperPrune Reproduction

The HyperPrune rows of Table 1 reproduce Sun & Sakuma (2026) from the authors' official implementation under the same conditions as every other method. As shipped, the method refines a SparseGPT prior in three stages: the one-shot masks are computed and propagated layer by layer with OBS weight compensation, a small shared hypernetwork (135,686 parameters) is pretrained to imitate the prior's per-group choices, and the hypernetwork is then fine-tuned layer by layer against cascaded per-projection reconstruction error. All three stages run with the shipped recipe at its own 128-sequence calibration budget. We deviate in exactly two places: the calibration corpus is ours, the same one every other method uses, and every decoder layer is pruned, since the shipped default keeps the first two layers dense and lands at 46.9% sparsity rather than a true 50%, which would not be comparable to the other rows.

Hyperparameter transfer is trivial for this method, by construction. For LLaMA-2-7B we run the exact configuration available in the authors' code base; for LLaMA-3.1-8B we started from their LLaMA-3-8B configuration to test whether its hyperparameters transfer, and found the question vacuous: the two shipped configurations are byte identical in every hypernetwork and training knob, so there is nothing model-specific to transfer, and the same shared recipe (with the two deviations above) also produced the Qwen2.5-0.5B checkpoint. The method is genuinely cheap at this scale, 5 to 25 minutes per model on one GPU within 2.6 to 10.9 GB of peak allocated memory. Running

it beyond the LLaMA family required small fixes, a bias data-type cast in the reconstruction path that fires on any model whose linear layers carry biases, and explicit architecture adapters, none of which changes the shipped method's arithmetic.

The release supports 2:4 only, so the 4:8 runs required an extension. The pipeline hard-codes the four-wide group and its $\binom{4}{2} = 6$-pattern space throughout, in the mask views, the prior-bias one-hot, and the supervised cross-entropy; we generalized these to $N{:}M$, where 4:8 raises the pattern space to $\binom{8}{4} = 70$ classes. The resulting 4:8 models appear in Table 1 and Appendix K. One cell is anomalous and reported as measured: the Qwen2.5-0.5B 4:8 model scores below its own 2:4 counterpart (eight-task average 30.64 against 32.23, WikiText-2 perplexity 282.93 against 149.94), although 4:8 is the weaker constraint and every other method improves from 2:4 to 4:8. The checkpoint verifies as a genuine 4:8 pattern, which points at that training run rather than at the evaluation.

K Per-Task and Per-Corpus Results

The task suite comprises MMLU (Hendrycks et al., 2021), PIQA (Bisk et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2020), OpenBookQA (Mihaylov et al., 2018), RACE (Lai et al., 2017), and HellaSwag (Zellers et al., 2019). This appendix expands the aggregated columns of Table 1: the evaluation protocol, per-task zero-shot accuracies (Tables 21–23), per-corpus perplexity and KL divergence including the calibration stream (Tables 24–26), and the agreement of our harness with PATCH's published numbers (Table 27). The per-task table of the prior-mask ablation closes the appendix (Table 28).

Every number comes from one evaluation job per checkpoint on a single NVIDIA RTX PRO 6000 GPU in `bfloat16` (Kalamkar et al., 2019), with `lm-evaluation-harness v0.4.10` on `torch v2.13.0+cu130` and `transformers v5.15.1`. Downstream tasks run zero-shot with each task's `lm-evaluation-harness` defaults, full task sets, and no chat template; scoring is deterministic log-likelihood, so the standard errors logged alongside each accuracy (0.35 to 2.1% points per task) are sampling error over task examples rather than seed variance; the per-task tables report them alongside each accuracy. Perplexity is word perplexity over 2048-token windows for every model regardless of its native context length, on the full WikiText-2 test split and the first 2000 documents of the C4 English validation split.

KL divergence is measured as KL(dense ‖ compressed) in nats per token against a per-family capture of the dense model's top-512 next-token log-probabilities plus one tail bucket for the remaining mass; coarsening is monotone under KL, so every reported value is a lower bound. The captured mass tracks vocabulary size, from roughly 0.96 on Qwen2.5 (152k vocabulary) to 0.99 on LLaMA-2 (32k), with the per-corpus values in each table's caption; where coverage falls below 0.99, sub-percent gaps between adjacent methods lie within the bound's error budget, while the ordering of methods within a family is unaffected, since they all score the same capture. The corpora are the full WikiText-2 test split, the first 4096 documents of the C4 validation split, and the exact SlimPajama calibration stream of the compression runs (4096 sequences of 2048 tokens, seed 0). The calibration stream favors the BLADE models by construction, since it is the distribution they calibrate on while MaskLLM's released masks were trained on other data, which is why Table 1 averages only the two held-out corpora. The ablation studies of Appendices E and F use the exact full-vocabulary estimator instead; the two agree closely but are not interchangeable.

Three reading notes. MaskLLM rows exist only at 2:4, because no original 4:8 masks were released. HyperPrune 4:8 rows come from our $N{:}M$ extension of the 2:4-only release, and the Qwen2.5-0.5B 4:8 cell is anomalous: it scores below its own 2:4 row although 4:8 is the weaker constraint; the checkpoint verifies as genuine 4:8 and the row is reported as measured (Appendix J). And LLaMA-2-7B at 4:8 is the one cell where BLADE$^{\text{Full}}$ does not dominate BLADE$^{\text{Mask}}$ on every metric: WikiText-2 perplexity (14.79 against 12.82) and WikiText-2 KL divergence (0.3017 against 0.2540) regress, while C4 and calibration KL divergence and the eight-task average improve.

Table 27 validates the harness against PATCH on every configuration the two evaluations share. Our MaskLLM rows score the released masks themselves, PATCH's own mask for Qwen2.5-0.5B and the MaskLLM authors' C4-trained masks for the two LLaMA models, applied to the official dense weights, so the Qwen row reproduces the very mask behind PATCH's published number. The dense

Table 21: Per-task zero-shot accuracy on Qwen2.5-0.5B (%, ↑), with the evaluation harness's sampling standard error.

	Method	ARC-C	ARC-E	HellaSwag	MMLU	OBQA	PIQA	RACE	WinoGrande	Avg.
	Dense	29.35 ±1.33	64.69 ±0.98	40.59 ±0.49	47.23 ±0.41	24.40 ±1.92	70.29 ±1.07	35.22 ±1.48	56.75 ±1.39	46.06
2:4	Wanda	17.83 ±1.12	42.93 ±1.02	28.78 ±0.45	23.03 ±0.36	11.80 ±1.44	58.76 ±1.15	26.32 ±1.36	52.33 ±1.40	32.72
	SparseGPT	18.26 ±1.13	42.09 ±1.01	30.40 ±0.46	22.95 ±0.35	12.20 ±1.47	59.30 ±1.15	30.05 ±1.42	52.57 ±1.40	33.48
	ADMM	19.54 ±1.16	46.00 ±1.02	31.08 ±0.46	22.90 ±0.35	16.40 ±1.66	59.41 ±1.15	30.72 ±1.43	54.06 ±1.40	35.01
	HyperPrune	17.92 ±1.12	39.69 ±1.00	28.72 ±0.45	22.90 ±0.35	13.00 ±1.51	58.38 ±1.15	26.79 ±1.37	50.43 ±1.41	32.23
	MaskPro	18.60 ±1.14	41.08 ±1.01	29.04 ±0.45	23.06 ±0.36	12.40 ±1.48	58.27 ±1.15	25.55 ±1.35	53.35 ±1.40	32.67
	MaskLLM	24.06 ±1.25	56.44 ±1.02	35.41 ±0.48	24.98 ±0.36	20.20 ±1.80	66.59 ±1.10	32.92 ±1.45	52.41 ±1.40	39.13
	BLADE$^{\text{Mask}}$	21.93 ±1.21	53.20 ±1.02	33.26 ±0.47	22.94 ±0.35	19.60 ±1.78	64.15 ±1.12	33.01 ±1.46	52.01 ±1.40	37.51
	BLADE$^{\text{Full}}$	24.23 ±1.25	55.93 ±1.02	34.44 ±0.47	23.16 ±0.36	21.40 ±1.84	64.69 ±1.12	33.40 ±1.46	54.38 ±1.40	38.95
4:8	Wanda	20.14 ±1.17	47.14 ±1.02	30.94 ±0.46	26.11 ±0.37	14.60 ±1.58	60.83 ±1.14	28.13 ±1.39	53.12 ±1.40	35.13
	SparseGPT	20.14 ±1.17	48.23 ±1.03	32.08 ±0.47	24.01 ±0.36	16.00 ±1.64	61.32 ±1.14	29.86 ±1.42	54.70 ±1.40	35.79
	ADMM	21.93 ±1.21	51.26 ±1.03	32.77 ±0.47	24.30 ±0.36	18.20 ±1.73	61.92 ±1.13	31.48 ±1.44	52.96 ±1.40	36.85
	HyperPrune	18.34 ±1.13	32.53 ±0.96	27.38 ±0.45	22.93 ±0.35	12.40 ±1.48	56.26 ±1.16	24.88 ±1.34	50.36 ±1.41	30.64
	MaskPro	20.31 ±1.18	46.34 ±1.02	30.92 ±0.46	25.62 ±0.37	15.60 ±1.62	60.99 ±1.14	27.66 ±1.38	52.33 ±1.40	34.97
	BLADE$^{\text{Mask}}$	23.63 ±1.24	56.44 ±1.02	34.58 ±0.47	23.96 ±0.36	20.60 ±1.81	64.42 ±1.12	33.01 ±1.46	55.41 ±1.40	39.01
	BLADE$^{\text{Full}}$	25.77 ±1.28	58.12 ±1.01	35.83 ±0.48	23.14 ±0.36	20.60 ±1.81	65.45 ±1.11	34.26 ±1.47	53.83 ±1.40	39.63

Table 22: Per-task zero-shot accuracy on LLaMA-2-7B (%, ↑), with the evaluation harness's sampling standard error.

	Method	ARC-C	ARC-E	HellaSwag	MMLU	OBQA	PIQA	RACE	WinoGrande	Avg.
	Dense	42.75 ±1.45	75.55 ±0.88	57.07 ±0.49	40.98 ±0.41	33.40 ±2.11	77.91 ±0.97	39.52 ±1.51	69.46 ±1.29	54.58
2:4	Wanda	29.61 ±1.33	61.49 ±1.00	41.15 ±0.49	25.72 ±0.37	23.00 ±1.88	69.91 ±1.07	35.60 ±1.48	61.25 ±1.37	43.47
	SparseGPT	30.55 ±1.35	63.80 ±0.99	42.95 ±0.49	25.46 ±0.37	23.40 ±1.90	69.59 ±1.07	37.22 ±1.50	63.46 ±1.35	44.55
	ADMM	30.89 ±1.35	64.35 ±0.98	43.74 ±0.50	28.79 ±0.38	24.40 ±1.92	71.11 ±1.06	36.94 ±1.49	64.64 ±1.34	45.61
	HyperPrune	28.75 ±1.32	63.89 ±0.99	41.98 ±0.49	26.57 ±0.37	24.20 ±1.92	70.02 ±1.07	37.51 ±1.50	65.59 ±1.34	44.81
	MaskPro	31.40 ±1.36	63.51 ±0.99	44.89 ±0.50	24.89 ±0.36	23.80 ±1.91	72.74 ±1.04	37.70 ±1.50	64.64 ±1.34	45.45
	MaskLLM	35.32 ±1.40	69.02 ±0.95	51.25 ±0.50	27.81 ±0.38	28.20 ±2.01	75.08 ±1.01	38.47 ±1.51	64.48 ±1.35	48.71
	BLADE$^{\text{Mask}}$	34.56 ±1.39	69.95 ±0.94	49.43 ±0.50	29.77 ±0.38	28.60 ±2.02	73.39 ±1.03	39.43 ±1.51	65.35 ±1.34	48.81
	BLADE$^{\text{Full}}$	36.52 ±1.41	69.61 ±0.94	50.63 ±0.50	30.81 ±0.39	29.20 ±2.04	73.72 ±1.03	39.14 ±1.51	66.93 ±1.32	49.57
4:8	Wanda	34.64 ±1.39	66.79 ±0.97	47.02 ±0.50	25.61 ±0.37	24.60 ±1.93	73.56 ±1.03	38.37 ±1.51	65.59 ±1.34	47.02
	SparseGPT	33.79 ±1.38	67.63 ±0.96	47.39 ±0.50	28.56 ±0.38	27.40 ±2.00	72.91 ±1.04	38.85 ±1.51	65.51 ±1.34	47.76
	ADMM	34.90 ±1.39	68.81 ±0.95	48.22 ±0.50	28.26 ±0.38	29.40 ±2.04	73.50 ±1.03	39.04 ±1.51	67.48 ±1.32	48.70
	HyperPrune	35.92 ±1.40	67.85 ±0.96	47.45 ±0.50	27.11 ±0.37	30.20 ±2.06	73.45 ±1.03	39.04 ±1.51	66.77 ±1.32	48.47
	MaskPro	33.79 ±1.38	66.12 ±0.97	49.41 ±0.50	27.36 ±0.38	27.80 ±2.01	74.27 ±1.02	39.81 ±1.51	64.40 ±1.35	47.87
	BLADE$^{\text{Mask}}$	36.77 ±1.41	71.04 ±0.93	51.33 ±0.50	29.98 ±0.38	30.20 ±2.06	75.03 ±1.01	39.71 ±1.51	65.59 ±1.34	49.96
	BLADE$^{\text{Full}}$	37.71 ±1.42	71.51 ±0.93	52.12 ±0.50	33.90 ±0.40	30.80 ±2.07	75.24 ±1.01	40.19 ±1.52	66.93 ±1.32	51.05

references agree within 0.3% points and every compressed method within 1.4; the remaining spread is consistent with evaluation-stack differences rather than checkpoint differences.

Table 28 expands the prior-mask ablation of Section 3.1 with per-task zero-shot accuracies for the eight-task suite.

L Extended Limitations

BLADE's design choices carry corresponding limits, summarized in Section 4 and elaborated here. The block solves are sequential and greedy: each block is optimized once, conditioned on its predecessors, and never revisited, so no downstream signal can improve an earlier block's mask, the price of never holding more than one block in memory. The enumeration underlying stage one is practical through 4:8 but not beyond (Section 2.3); patterns such as 8:16 require linear-space parameterizations that we do not evaluate. Within 4:8, the top-32 restricted search is validated only at scales where the full search still fits, with agreement declining from 99.5% on Qwen2.5-0.5B to 94.7% on LLaMA-3.1-8B, so at the sizes where the restriction is actually required its fidelity is an extrapolation, checked only by the matched held-out block error of Section 3.1. All masks and weight updates are calibrated on a fixed data distribution, and behavior under distribution shift is not guaranteed. The tune-once recipe assumes blocks are homogeneous enough for first-block hyperparameters to transfer; Appendix D supports this on the evaluated families under Lion, and Section 3.1 shows that the assumption does not hold under other mask optimizers, so heterogeneous architectures or a different optimizer may need re-tuning. Compression wall-clock, while requiring only single-block memory, remains a day at the 7B scale (Section 3.1) rather than the minutes of

Table 23: Per-task zero-shot accuracy on LLaMA-3.1-8B (%, ↑), with the evaluation harness's sampling standard error.

	Method	ARC-C	ARC-E	HellaSwag	MMLU	OBQA	PIQA	RACE	WinoGrande	Avg.
	Dense	52.05 ±1.46	82.24 ±0.78	60.71 ±0.49	63.97 ±0.38	34.00 ±2.12	79.05 ±0.95	39.14 ±1.51	73.56 ±1.24	60.59
2:4	Wanda	26.79 ±1.29	60.02 ±1.01	37.97 ±0.48	30.34 ±0.39	20.20 ±1.80	69.70 ±1.07	33.01 ±1.46	60.69 ±1.37	42.34
	SparseGPT	30.12 ±1.34	62.84 ±0.99	41.97 ±0.49	31.11 ±0.39	23.20 ±1.89	69.10 ±1.08	35.98 ±1.49	64.72 ±1.34	44.88
	ADMM	30.20 ±1.34	64.65 ±0.98	42.42 ±0.49	35.57 ±0.40	22.80 ±1.88	69.04 ±1.08	36.08 ±1.49	64.88 ±1.34	45.71
	HyperPrune	31.83 ±1.36	64.94 ±0.98	42.45 ±0.49	29.90 ±0.38	24.40 ±1.92	69.42 ±1.07	37.42 ±1.50	65.35 ±1.34	45.71
	MaskPro	29.69 ±1.34	61.83 ±1.00	41.80 ±0.49	30.27 ±0.39	22.20 ±1.86	69.86 ±1.07	33.40 ±1.46	62.12 ±1.36	43.90
	MaskLLM	40.02 ±1.43	73.70 ±0.90	54.34 ±0.50	43.99 ±0.41	28.20 ±2.01	76.93 ±0.98	38.76 ±1.51	67.88 ±1.31	52.98
	BLADE$^{\text{Mask}}$	38.91 ±1.42	72.22 ±0.92	51.06 ±0.50	41.57 ±0.41	26.60 ±1.98	74.76 ±1.01	40.48 ±1.52	68.11 ±1.31	51.71
	BLADE$^{\text{Full}}$	40.70 ±1.44	73.61 ±0.90	52.27 ±0.50	41.35 ±0.40	28.20 ±2.01	75.14 ±1.01	40.77 ±1.52	70.01 ±1.29	52.75
4:8	Wanda	34.04 ±1.38	65.70 ±0.97	44.64 ±0.50	38.81 ±0.41	23.20 ±1.89	72.47 ±1.04	36.36 ±1.49	67.09 ±1.32	47.79
	SparseGPT	36.69 ±1.41	69.15 ±0.95	46.65 ±0.50	41.25 ±0.41	26.20 ±1.97	71.33 ±1.06	37.03 ±1.49	67.80 ±1.31	49.51
	ADMM	37.37 ±1.41	69.15 ±0.95	47.51 ±0.50	45.93 ±0.41	26.00 ±1.96	72.20 ±1.05	37.51 ±1.50	69.61 ±1.29	50.66
	HyperPrune	36.43 ±1.41	68.86 ±0.95	47.26 ±0.50	44.14 ±0.41	28.00 ±2.01	73.07 ±1.04	37.80 ±1.50	70.40 ±1.28	50.74
	MaskPro	36.69 ±1.41	67.68 ±0.96	46.36 ±0.50	41.75 ±0.41	26.40 ±1.97	72.36 ±1.04	37.80 ±1.50	67.80 ±1.31	49.60
	BLADE$^{\text{Mask}}$	41.72 ±1.44	74.24 ±0.90	53.14 ±0.50	47.80 ±0.41	29.40 ±2.04	76.44 ±0.99	39.04 ±1.51	68.19 ±1.31	53.75
	BLADE$^{\text{Full}}$	43.17 ±1.45	74.58 ±0.89	53.94 ±0.50	50.31 ±0.40	29.20 ±2.04	76.88 ±0.98	39.71 ±1.51	69.85 ±1.29	54.71

Table 24: Per-corpus perplexity and KL divergence to the dense model on Qwen2.5-0.5B (↓; KLD in nats/token, top-512 lower bound). Coverage of the top-512 capture: 0.9657 (WikiText-2), 0.9565 (C4), 0.9646 (calibration stream). Best method per column and pattern in **bold**, the second best underlined.

		PPL (↓)		KLD (↓)		
	Method	WikiText-2	C4	WikiText-2	C4	Calib.
	Dense	19.65	46.15	–	–	–
2:4	Wanda	155.51	414.99	1.7263	1.8873	1.6212
	SparseGPT	54.86	140.66	0.9865	0.8735	0.8371
	ADMM	48.58	121.88	0.8200	0.7882	0.7305
	HyperPrune	149.94	317.76	1.8042	1.8113	1.6670
	MaskPro	129.98	308.64	1.6331	1.6461	1.4785
	MaskLLM	**24.34**	**55.26**	**0.2940**	**0.2716**	0.2305
	BLADE$^{\text{Mask}}$	30.99	74.53	0.4400	0.3654	0.2702
	BLADE$^{\text{Full}}$	27.83	68.06	0.3516	0.2836	**0.1728**
4:8	Wanda	77.08	197.60	1.1593	1.2673	1.0901
	SparseGPT	38.74	97.13	0.6299	0.5793	0.5577
	ADMM	36.80	91.97	0.5619	0.5479	0.5158
	HyperPrune	282.93	698.66	2.2399	2.1878	2.0174
	MaskPro	67.38	161.62	1.1442	1.0493	0.9788
	BLADE$^{\text{Mask}}$	27.61	65.82	0.3349	0.2747	0.1997
	BLADE$^{\text{Full}}$	**25.70**	**63.94**	**0.2846**	**0.2197**	**0.1321**

one-shot solvers, and our largest evaluated models are at the 8B scale: nothing in the cost model grows super-linearly with depth, but we have not verified quality beyond the scales of Section 3. Finally, quality at a pattern and speedup from a pattern are separate facts: realized acceleration at 4:8 depends on the deployment target's kernel and hardware support, which remains uneven outside 2:4 tensor cores.

Table 25: Per-corpus perplexity and KL divergence to the dense model on LLaMA-2-7B (↓; KLD in nats/token, top-512 lower bound). Coverage of the top-512 capture: 0.9908 (WikiText-2), 0.9873 (C4), 0.9906 (calibration stream). Best method per column and pattern in **bold**, the second best underlined.

		PPL (↓)		KLD (↓)		
	Method	WikiText-2	C4	WikiText-2	C4	Calib.
	Dense	9.19	18.73	–	–	–
2:4	Wanda	23.91	55.78	0.7613	0.7540	0.7057
	SparseGPT	18.53	41.78	0.5687	0.5447	0.5118
	ADMM	19.14	43.36	0.5728	0.5758	0.5275
	HyperPrune	21.65	49.35	0.6528	0.6611	0.5982
	MaskPro	18.14	42.49	0.5343	0.5385	0.4941
	MaskLLM	**13.10**	**26.42**	0.3010	**0.2449**	0.2468
	BLADE$^{\text{Mask}}$	14.32	32.76	0.3951	0.3411	0.1459
	BLADE$^{\text{Full}}$	13.60	31.55	**0.2892**	0.3030	**0.0832**
4:8	Wanda	15.47	34.64	0.4105	0.4280	0.3898
	SparseGPT	14.52	32.14	0.3684	0.3723	0.3436
	ADMM	14.65	32.59	0.3625	0.3737	0.3412
	HyperPrune	15.41	34.18	0.3930	0.4026	0.3632
	MaskPro	14.33	33.37	0.3539	0.3705	0.3298
	BLADE$^{\text{Mask}}$	**12.82**	**29.09**	**0.2540**	0.2637	0.1039
	BLADE$^{\text{Full}}$	14.79	30.92	0.3017	**0.2395**	**0.0614**

Table 26: Per-corpus perplexity and KL divergence to the dense model on LLaMA-3.1-8B (↓; KLD in nats/token, top-512 lower bound). Coverage of the top-512 capture: 0.9884 (WikiText-2), 0.9809 (C4), 0.9850 (calibration stream). Best method per column and pattern in **bold**, the second best underlined.

		PPL (↓)		KLD (↓)		
	Method	WikiText-2	C4	WikiText-2	C4	Calib.
	Dense	8.07	17.51	–	–	–
2:4	Wanda	35.65	96.49	1.2919	1.3136	1.1994
	SparseGPT	21.12	51.35	0.8940	0.8299	0.7842
	ADMM	21.22	52.19	0.8669	0.8590	0.7924
	HyperPrune	22.27	53.96	0.9056	0.8950	0.8211
	MaskPro	24.41	61.13	0.9795	0.9342	0.8927
	MaskLLM	**12.07**	**24.04**	**0.3710**	**0.3115**	0.3552
	BLADE$^{\text{Mask}}$	14.15	33.71	0.5220	0.5120	0.2328
	BLADE$^{\text{Full}}$	13.66	31.77	0.4638	0.4574	**0.1303**
4:8	Wanda	19.90	48.95	0.7829	0.7916	0.7278
	SparseGPT	15.51	37.18	0.6273	0.5817	0.5470
	ADMM	15.87	37.32	0.6104	0.5908	0.5410
	HyperPrune	16.39	38.57	0.6358	0.6289	0.5624
	MaskPro	16.75	40.84	0.6612	0.6360	0.5933
	BLADE$^{\text{Mask}}$	12.53	30.30	0.4091	0.4225	0.1755
	BLADE$^{\text{Full}}$	**12.16**	**28.93**	**0.3705**	**0.3805**	**0.0999**

Table 27: Harness validation at 2:4: eight-task zero-shot averages (%) from our unified evaluation against the values published by PATCH (its Tables 9 to 11), on every configuration the two evaluations share.

Model	Method	Ours	PATCH	Δ
Qwen2.5-0.5B	Dense	46.06	46.00	+0.06
	Wanda	32.72	32.97	−0.25
	SparseGPT	33.48	34.81	−1.34
	MaskLLM	39.13	39.33	−0.21
LLaMA-2-7B	Dense	54.58	54.61	−0.03
	Wanda	43.47	44.30	−0.84
	SparseGPT	44.55	45.09	−0.53
	MaskLLM	48.71	48.62	+0.08
LLaMA-3.1-8B	Dense	60.59	60.31	+0.28
	Wanda	42.34	41.77	+0.57
	SparseGPT	44.88	45.53	−0.65
	MaskLLM	52.98	52.80	+0.18

Table 28: Per-task zero-shot accuracy (%, ↑) for the prior-mask ablation on Qwen2.5-0.5B.

	Prior	ARC-C	ARC-E	HellaSwag	MMLU	OBQA	PIQA	RACE	WinoGrande	Avg.
2:4	SparseGPT	23.38	53.79	33.91	22.87	18.20	64.31	31.48	54.46	37.80
	Magnitude	22.10	53.66	33.69	22.92	19.60	64.04	32.82	54.78	37.95
	Wanda	22.18	54.38	33.77	22.92	19.40	64.36	31.96	55.33	38.04
	ADMM	23.98	53.37	33.83	22.91	18.40	63.82	32.82	54.62	37.97
4:8	SparseGPT	25.43	57.95	34.91	22.94	20.60	65.40	33.11	54.14	39.31
	Magnitude	25.26	58.92	34.87	23.83	19.80	65.72	33.88	54.54	39.60
	Wanda	25.26	55.77	35.02	22.92	20.40	64.96	32.34	53.67	38.79
	ADMM	25.26	56.44	34.83	22.96	21.40	64.74	34.16	54.22	39.25