SMALL LLMS: PRUNING VS. TRAINING FROM SCRATCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning promises a shortcut to strong small models. However, Liu et al. (2019) found that in CNNs, this shortcut often loses to training from scratch. LLMs are substantially different, with larger model and data scales and different architectures, so the same conclusions may not hold. We examine this by pruning Llama-3.1-8B at pruning ratios of 0.5–0.8 with six methods spanning depth, width, and sparse granularities, under two token-matched settings. **(1)** With the same training token budget, pruned initialization consistently outperforms random initialization. This shows that the parent model provides a strong starting point, although the advantage narrows as the training token budget grows and, for depth pruning, as the pruning ratio rises, nearly vanishing at the highest ratio we study. **(2)** When training from scratch is instead given the full token budget consumed by the whole pipeline, pruning at finer granularities still retains a small advantage, while coarser structured pruning can be matched or surpassed. This suggests that the parent model may transfer knowledge that additional tokens alone cannot fully recover, but only at fine granularity. This forms a clear recommendation: with a strong parent and a limited token budget, pruning is more effective; when tokens are plentiful, training from scratch is competitive at coarse granularity, so a parent is not always necessary.

1 INTRODUCTION

Remove more than half of a trained language model's parameters, add a lightweight retraining pass, and the resulting small model still matches most of the original's performance (Ma et al., 2023; Xia et al., 2024; Sun et al., 2024; Sreenivas et al., 2024). This is the promise of network pruning (Le Cun et al., 1989; Hassibi et al., 1993): a shortcut to strong small language models that bypasses the rising cost of pretraining Large Language Models (LLMs) from scratch on trillions of tokens (Grattafiori et al., 2024; Qwen Team, 2025; DeepSeek-AI, 2024; Kaplan et al., 2020; Hoffmann et al., 2022; Henighan et al., 2020; Touvron et al., 2023a;b).

Multi-billion-parameter open-weight LLMs are now widely available, with releases such as Llama (Grattafiori et al., 2024; Meta AI, 2024), Qwen (Qwen Team, 2025), Gemma (Google, 2025b), and OLMo (OLMo Team, 2025) putting pretrained checkpoints in public hands. At the same time, demand for smaller models is rising, driven by deployment constraints, inference efficiency, and accessibility (Hägele et al., 2024; Hu et al., 2024; Nag et al., 2025). This is exactly the gap pruning promises to fill. This raises a practical question: to obtain a capable small model, should we train it from scratch, or carve it out of an existing large one via pruning? Pruning is appealing because it may transfer knowledge from the parent, giving a stronger starting point than random initialization. But the inheritance is not free: a large parent must be pretrained before it can be pruned, and whether pruning's advantage survives a token-matched comparison remains unclear.

While pruning is typically viewed as a compression technique, we instead treat it as an *initialization strategy*: the pruned weights are a starting point for training the target small model. This reframes the natural baseline: rather than comparing the pruned model to its parent, we compare it to a model of the same size trained from a random initialization on the same data stream. The question becomes: under an equal training token budget, does *initializing by pruning a larger model* beat *random initializing*?

Liu et al. (2019) first examined this question for convolutional networks on image-classification benchmarks. For *structured* pruning, they found that a target architecture trained from *scratch* matches or surpasses the same architecture pruned and fine-tuned from a larger model, implying

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.

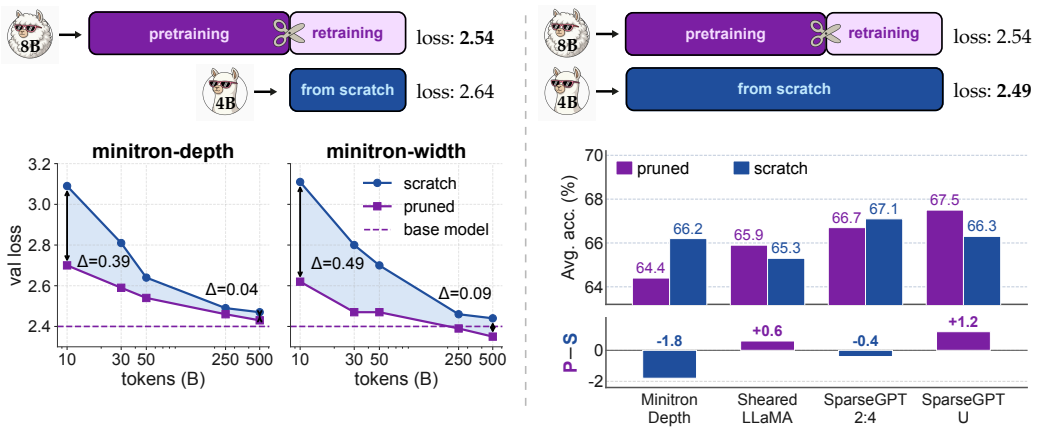


Figure 1. Initialization by pruning provides a strong advantage over random initialization, but this advantage diminishes as training continues. *Left:* under the same training token budget, pruning initialization beats random initialization, although the advantage decreases with longer training. *Right:* when the random initialization baseline is trained with the full token budget used by the entire pruning pipeline, it becomes competitive with the pruning pipeline. Overall, pruning is a powerful shortcut to strong small models, but its advantage diminishes as training scales.

that pruning's value is the *architecture* it discovers rather than the weights it inherits, and recasting structured pruning as a form of neural architecture search. For *unstructured* pruning, however, the inherited weights were harder to replace, with training from scratch falling short of the pruned-and-fine-tuned model at larger scale. This suggests that pruning's benefit shifts with granularity, from the architecture alone at the coarse end toward the specific inherited weights at the fine end. Whether this carries over to LLMs is unclear, given a different regime: larger data and model scales, different architectures, and higher demands for compact deployable models.

We therefore ask a sharper question: are pruning's gains merely a head start that more training can buy, or knowledge that extra data cannot recover? To find out, we compare pruned against random initialization under two token-matched settings. **(i) Equal training token budget**: both target models are trained under the same training token budget, isolating the effect of initialization. **(ii) Equal total token budget**: training from scratch is instead given the *entire* token budget of the pruning pipeline (the parent's pretraining tokens plus the retraining tokens), testing whether more tokens alone can close the gap. Varying pruning ratio and granularity across both, we draw two key observations, both illustrated in Figure 1:

(1) Under an equal training token budget, pruned initialization consistently outperforms random initialization, though for depth pruning this advantage shrinks at higher pruning ratios.

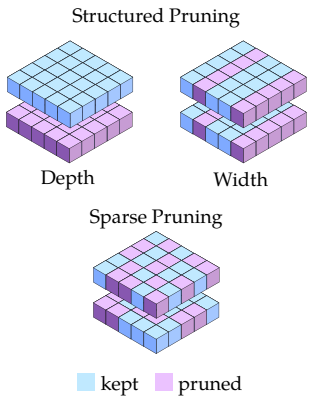
(2) Even when training from scratch uses the full token budget (number of training iterations) of the pruning pipeline, sparse pruning still holds a small edge; for structured pruning, however, training from scratch can close the gap.

Together, these results recast pruning as a token-efficient initialization strategy: with a strong parent model in hand and a limited training token budget, pruning beats training from scratch, especially at fine granularity, where it appears to transfer knowledge that additional tokens alone cannot recover.

2 Background

Network pruning (Le Cun et al., 1989) identifies and removes unimportant weights or modules in neural networks to improve efficiency (Frankle & Carbin, 2019). Methods fall into two families (Figure 2 (a)): *sparse pruning*, which zeroes out individual weights while preserving model shape, and *structured pruning*, which removes entire architectural components such as layers, attention heads, or feedforward channels.

(a) Pruning granularity



(b) Methods evaluated

method	target modules	criterion
Minitron depth	layers	influence on validation loss
Minitron width	hidden channels, mlp channels	block output activation norm
FLAP	hidden channels, mlp channels	input feature variance × weight column norm
Sheared LLaMA	layers, hidden channels, mlp channels, attn heads	learned pruning masks (Lagrangian optim.)
Wanda	individual weights	weight magnitude × input feature norm
SparseGPT	individual weights	layerwise reconstruction error

Figure 2. Pruning granularity and method overview. (a) The pruning granularities we study: depth, width, and sparse pruning. (b) The six pruning methods we evaluate, with their target modules and criteria.

Sparse pruning. Sparse methods mask individual weights to zero. *Semi-structured* (n:m) pruning (Zhou et al., 2021) keeps exactly n of every m weights and can exploit hardware support (Mishra et al., 2021; Hubara et al., 2021); *unstructured* pruning places no constraint on which weights are removed. SparseGPT (Frantar & Alistarh, 2023) frames weight selection as a layerwise reconstruction problem inspired by Optimal Brain Surgeon (Hassibi et al., 1993). Wanda (Sun et al., 2024) approximates the same objective more cheaply by scoring each weight as its magnitude times the corresponding input feature norm, with follow-up work (Yang et al., 2025) further refining the activation-based scoring. Both methods support n:m and unstructured modes and need no retraining: SparseGPT updates the remaining weights, while Wanda does not.

Structured pruning. Structured methods, with roots in classical CNN-era channel and filter pruning (Li et al., 2017; Liu et al., 2017; Luo et al., 2017; Huang & Wang, 2018), remove whole modules, yielding smaller dense architectures. *Depth pruning* removes entire layers: ShortGPT (Men et al., 2025), Shortened LLaMA (Kim et al., 2024), and SLEB (Song et al., 2024) estimate layer importance via activation similarity, Taylor approximations, and block-skip perplexity respectively; Minitron-depth (Sreenivas et al., 2024) searches for the consecutive layer group whose removal least hurts validation loss. *Width pruning* removes hidden or feedforward channels: Minitron-width (Sreenivas et al., 2024) ranks channels by output activation norm, while FLAP (An et al., 2024) uses input feature variance weighted by column norm. Sheared LLaMA (Xia et al., 2024) jointly prunes both depth and width by learning pruning masks via Lagrangian optimization subject to a fixed target model size constraint specified in advance.

Pruning ratio and granularity. The *pruning ratio* is the fraction of parameters removed. *Granularity* refers to the smallest unit treated as a single pruning decision, ranging from whole layers (coarsest) through channels and attention heads down to individual weights (finest). Under the same ratio, finer granularity preserves more of the base model's performance but yields less hardware speedup and reduced memory savings at inference time (An et al., 2024; Sreenivas et al., 2024).

3 Methodology

We study pruning under constrained token budgets through controlled comparisons that isolate (i) the value of pruning as an initialization strategy and (ii) whether additional training from scratch can close the gap to the pruning pipeline. Section 3.1 describes the experimental design, base model, data pipeline, optimizer, and evaluation protocol. Section 3.2 formalizes the six pruning methods.

3.1 Training Setup

Experimental design. Throughout, we use three notations: SN denotes training the target architecture from Scratch with random initialization for NB tokens; P200-RN denotes the pruning pipeline, which Pretrains a larger model for 200B tokens, prunes it at a target ratio, then Retrains the pruned model for NB tokens; and *Meta*-RN denotes the same pipeline but starting from Meta's released Llama-3.1-8B checkpoint instead of our 200B-pretrained one, retrained for NB tokens (used as a reference point in Figure 3). Both token-matched experiments compare the pruning pipeline against training from scratch, differing only in how much data we allow the scratch baseline to see:

(1) **Equal training token budget** (S50 vs. P200-R50): the scratch baseline trains on the same 50B-token data stream used to retrain the pruned model, under the same target architecture and optimizer schedule, isolating the effect of initialization alone. We further sweep the pruning ratio to track how this advantage evolves with more aggressive compression.

(2) **Equal total token budget** (S250 vs. P200-R50): the scratch baseline trains on all 250B tokens consumed by the pipeline (pretraining plus retraining), testing whether additional tokens alone can close the gap. This matches tokens, not FLOPs: S250 receives about 62% of the pipeline's training FLOPs (counting non-zero parameters, Table 1), though for sparse models, which run no faster than dense ones on our TPUs, realized costs are near equal.

Model and dataset. We adopt Llama-3.1-8B (Grattafiori et al., 2024) as the base model for our experiments, as Llama-family models are the most common choice in prior LLM pruning work (Frantar & Alistarh, 2023; Sun et al., 2024; Sreenivas et al., 2024). For training data, we adopt DCLM-Baseline-1.0 (Li et al., 2024), a large-scale curated training corpus comparable to other contemporary web-scale corpora such as RefinedWeb (Penedo et al., 2023), Dolma (Soldaini et al., 2024), and RedPajama (Weber et al., 2024). The dataset is tokenized with the Llama-3.1 tokenizer into chunks of 8192 tokens.

Data pipeline. The 200B pretraining tokens and the 50B retraining tokens are strictly non-overlapping subsets of the DCLM corpus. For the S250 comparison, the 250B tokens are exactly the union of these two disjoint sets, ensuring that S250 and P200-R50 see exactly the same data in total. For the S50 comparison, S50 uses the same 50B tokens as the retraining phase of P200-R50. This guarantees that all comparisons are token-fair and that differences in performance reflect initialization strategy, not data composition.

Training recipe. We follow the default recipe in Lingua (Videau et al., 2024), using AdamW (Loshchilov & Hutter, 2019) with cosine learning rate decay and a 5% warmup. For each experiment we sweep the max learning rate over `{1e-5,3e-5,1e-4,3e-4,1e-3}` and select the one yielding lowest training loss in a short prefix run. Both arms use plain next-token cross-entropy: we study pruning as an initialization, not each method's strongest recipe (e.g., with distillation; Appendix A). This is important for retraining after pruning, where the learning rate must be chosen carefully to prevent catastrophic forgetting (Parmar et al., 2024; Gupta et al., 2023). Full training hyperparameters are in Appendix D.1.

Evaluation. We evaluate on two dimensions: (i) linguistic perplexity across four general-domain corpora (C4, WikiText-103, WikiText-2, CNN Dailymail) and (ii) zero-shot downstream accuracy across eight benchmarks spanning commonsense QA (WinoGrande, HellaSwag, PIQA), scientific QA (ARC-Challenge, ARC-Easy, SciQ, OpenBookQA), and logical reasoning (BoolQ). Full protocols, citations, and random-chance baselines are in Appendix E.

3.2 Pruning Methods

We evaluate six representative LLM pruning methods spanning two families, with their target modules and scoring criteria summarized in Figure 2 (b) (and described in Section 2). *Structured* methods remove entire model components while keeping the weight matrix dense (**Minitron-depth** (Muralidharan et al., 2024; Sreenivas et al., 2024), **Minitron-width** (Muralidharan et al., 2024; Sreenivas et al., 2024), **FLAP** (An et al., 2024), and **Sheared LLaMA** (Xia et al., 2024)); *sparse* methods retain the full macro-architecture but zero-mask individual weights (**Wanda** (Sun et al., 2024) and

Table 1. Architectures of the pruned models obtained with different pruning methods and pruning ratios. The base model is Llama-3.1-8B (8.0B parameters). For sparse methods, we count only non-zero weights.

method	ratio	num layers	attention heads	hidden size	FFN size	sparsity	parameters
base	0.0%	32	32	4096	14336	0.0%	8.0B
Minitron-D	50.0%	16	32	4096	14336	0.0%	4.5B
Minitron-D	62.5%	8	32	4096	14336	0.0%	2.8B
Minitron-D	75.0%	4	32	4096	14336	0.0%	1.9B
Minitron-D	81.3%	2	32	4096	14336	0.0%	1.5B
Minitron-W	50.0%	32	32	3072	9216	0.0%	4.5B
Minitron-W	62.5%	32	32	2432	6144	0.0%	3.1B
Minitron-W	75.0%	32	32	1792	6016	0.0%	2.0B
Minitron-W	81.3%	32	32	1536	4736	0.0%	1.5B
FLAP	50.0%	32	20	4096	6656	0.0%	4.5B
Sheared LLaMA	50.0%	32	32	3456	7552	0.0%	4.6B
Wanda	50.0%	32	32	4096	14336	50.0%	4.5B
SparseGPT	50.0%	32	32	4096	14336	50.0%	4.5B

SparseGPT (Frantar & Alistarh, 2023)). For brevity, we refer to Minitron-depth as **Minitron-D**, Minitron-width as **Minitron-W**, and the unstructured variants of Wanda and SparseGPT as **Wanda-U** and **SparseGPT-U**, respectively. Formal descriptions of all six methods are in Appendix C.2.

Implementation. For Minitron-depth and Minitron-width, we developed a simplified re-implementation and verified its correctness against the original results. FLAP and Sheared LLaMA were originally designed for models with multi-head attention (MHA); we adapted both to the grouped-query attention (GQA) (Ainslie et al., 2023) of Llama-3.1 and verified that they achieve reasonable pruning performance on it. For Wanda and SparseGPT, we used the authors' original code unmodified. When a target architecture was not specified in the original paper, we performed a lightweight architecture search over candidate dimension configurations; this search, calibration sets, and hyperparameters are in Appendix D.2.

Pruning configurations. The six methods span five granularities: depth, width, depth-and-width, 2:4 sparse, and unstructured sparse. We center our comparison at a 50% pruning ratio, the canonical setting at which Minitron (Sreenivas et al., 2024), SparseGPT (Frantar & Alistarh, 2023), and Wanda (Sun et al., 2024) report their primary results, and search for the best configuration within each method's design space. Concretely, we evaluate Minitron-depth at $\{50\%, 62.5\%, 75\%, 81.3\%\}$ pruning ratios, Minitron-width at $\{50\%, 62.5\%, 75\%, 81.3\%\}$, FLAP and Sheared LLaMA at 50%; SparseGPT and Wanda use 2:4 and 50% unstructured sparsity (architectures in Table 1).

4 Pruning vs. Training from Scratch

We compare P200-R50 against S50 (equal training token budget) and S250 (equal total token budget), and, for Minitron-D and Minitron-W, vary the pruning ratio beyond 50%. Table 2 summarizes all six methods at 50% pruning ratio (full breakdown in Table 16). Among the six methods, **Wanda-U** yields the most accurate pruned model (68.1% average) and **FLAP** the lowest perplexity. The accuracy ranking broadly tracks pruning granularity, though the most accurate (sparse) methods bring little inference speedup on commodity hardware—a trade-off we examine in Section 4.3.

4.1 Equal Training Token Budget

Finding 1. Under an equal training token budget, pruning initialization consistently beats random initialization. For depth pruning, the advantage vanishes near an 81% pruning ratio; width pruning keeps a clear advantage at every ratio.

Table 2. Pruning–retraining vs. training from scratch at 50% pruning ratio (summary). WikiText-2 perplexity (↓) and average downstream accuracy (↑, eight benchmarks) for P200-R50 (pretrain 200B, prune, retrain 50B), S50 (scratch for 50B; equal training token budget), and S250 (scratch for 250B; equal total token budget). **Bold**: best P200-R50 result per metric. *Pruning beats equal-training-token-budget scratch (S50) for every method; under an equal total token budget (S250), only the finer (sparse) granularities keep a small edge, while coarser structured pruning is matched.* Per-benchmark results are in Table 16.

	WikiText-2 ppl ↓			Avg accuracy (%) ↑		
method	S50	P200-R50	S250	S50	P200-R50	S250
Minitron-D	10.77	9.41	9.01	60.7	64.4	66.2
Minitron-W	11.17	9.01	8.69	59.9	66.3	67.5
FLAP	11.34	**7.97**	8.82	57.7	66.5	65.1
Sheared LLaMA	11.28	8.63	8.85	58.4	65.9	65.3
Wanda-2:4	10.55	9.15	8.60	61.5	66.7	66.2
Wanda-U	10.66	8.37	8.57	61.6	**68.1**	67.2
SparseGPT-2:4	10.54	8.90	8.65	62.1	66.7	67.1
SparseGPT-U	10.47	8.44	8.59	61.9	67.5	66.3

Pruning initialization is consistently better than random initialization under 50% pruning ratio. Table 2 summarizes the 50% pruning results; Figure 3 plots DCLM loss and average accuracy across retraining token budgets for all four structured methods (full results in Tables 12–15), and its $N{=}50$ slice matches the comparison discussed here. On Minitron-D, P200-R50 outperforms S50 on all 8 benchmarks, with 4.5% and 6.1% gains on ARC Challenge and HellaSwag (Figure 3a). On Minitron-W, the gains are larger, 8.2% on ARC Challenge and 10.2% on HellaSwag (Figure 3b), which is consistent with the observation that Minitron-W preserves more of the base model's capabilities than Minitron-D (Sreenivas et al., 2024). FLAP and Sheared LLaMA exhibit the same qualitative trend across token budgets (Figures 3c and 3d).

Sparse pruning follows the same pattern, with the finer unstructured granularity gaining more than 2:4: for Wanda, P200-R50 improves WikiText-2 perplexity over S50 by 1.40 (2:4) versus 2.30 (unstructured), and for SparseGPT the ARC-Challenge gain grows from 2.0% (2:4) to 8.1% (unstructured). This unstructured-over-2:4 ordering holds across nearly all benchmarks (Table 2).

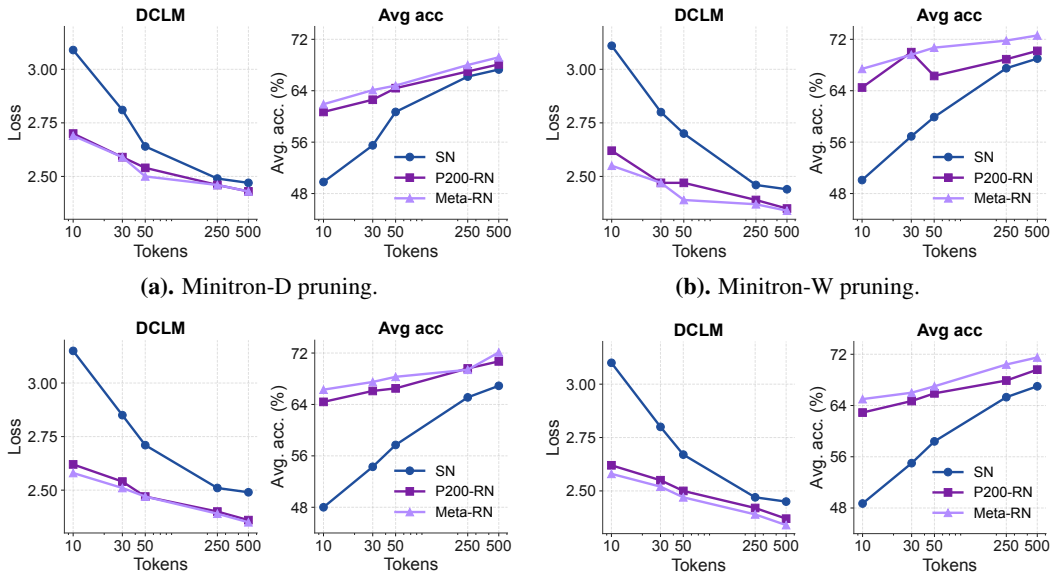
For depth pruning, the advantage diminishes as pruning ratio increases. Table 11 (Appendix H.2) reports both Minitron variants across pruning ratios. For Minitron-D at 50% pruning ratio, P200-R50 shows a clear advantage over S50, with 1.31 lower perplexity on WikiText-103. The gap narrows substantially at 62.5% and 75%. At 81.3% pruning ratio, P200-R50 performs on par with or even slightly worse than S50 on some benchmarks, suggesting that depth-pruned initialization loses its benefit under extreme compression.

On perplexity, the Minitron-D gap closes as the pruning ratio increases to 81.3%, with the curves eventually intersecting; accuracy follows a similar pattern, with exceptions such as Winogrande. Minitron-W behaves differently: its advantage persists at every ratio, with $+5.1$ average accuracy even at 81.3%. Figure 4 visualizes both trends across pruned model sizes: the advantage shrinks as the target gets smaller for depth pruning (Figure 4a) but not for width pruning (Figure 4b).

4.2 Can Extended Training Catch Up?

Finding 2. Given the pipeline's full token budget, training from scratch catches up to pruning at coarser granularities, while pruning at finer granularities keeps a small edge.

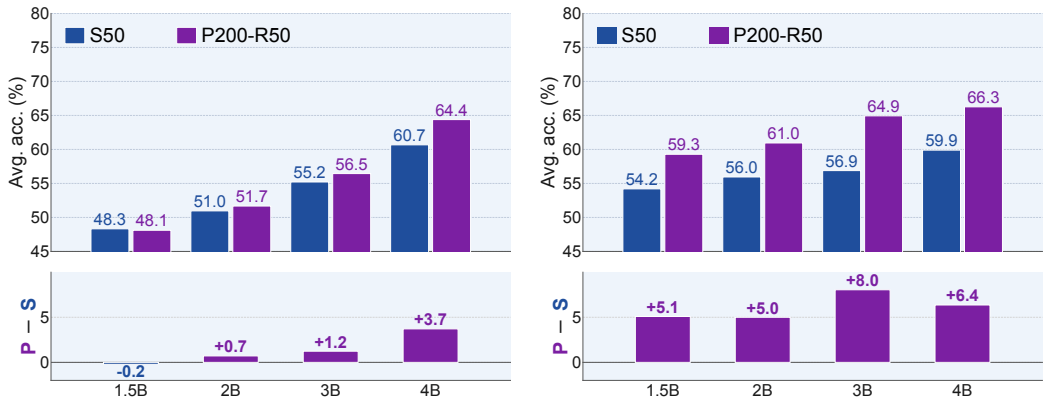
Longer scratch training matches pruning at coarser granularities; at finer ones, pruning keeps a small edge. Figure 5 compares P200-R50 against S250 (equal total token budget). On Minitron-D, S250 surpasses P200-R50 on every benchmark (0.40 lower WikiText-2 perplexity; 0.6–4.0% higher accuracy). The margin is smaller for the finer Minitron-W (e.g., the OBQA gain shrinks from 4.0% to 2.2%), showing that longer scratch training recovers the advantage of coarse-granularity pruning.



(a). Minitron-D pruning. **(b).** Minitron-W pruning.

(c). FLAP pruning. **(d).** Sheared LLaMA pruning.

Figure 3. Four structured pruning methods across retraining token budgets (Llama-3.1-8B → 4B, 50% pruning). Each panel shows DCLM validation loss (↓) and average downstream accuracy (↑; averaged over WinoGrande, ARC-C, ARC-E, HellaSwag, PIQA, SciQ, BoolQ, OBQA) as a function of retraining tokens $N \in \{10, 30, 50, 250, 500\}$B under three initialization strategies: *SN* (train from scratch for NB tokens), *P200-RN* (pretrain 200B tokens, prune, retrain for NB tokens), and *Meta-RN* (prune from Meta's released Llama-3.1-8B, retrain for NB tokens). *Pruned initialization (P200-RN) starts well ahead of scratch (SN) and keeps its lead as retraining grows, though the gap narrows with more tokens.* Full results are in Appendix H.3.



(a). Depth pruning: S50 vs. P200-R50 by model size. **(b).** Width pruning: S50 vs. P200-R50 by model size.

Figure 4. S50 vs. P200-R50 across model sizes for depth and width pruning. Top panels show average accuracy (WinoGrande, ARC-C, ARC-E, HellaSwag, PIQA, SciQ, BoolQ, OBQA); bottom panels show the gain of P200-R50 over S50 per model size. Dark blue bars denote scratch-trained models (S50); purple bars denote pruned-then-retrained models (P200-R50) at the same target parameter count. *For depth pruning, the advantage shrinks as the target model gets smaller, nearly closing at the smallest size; width pruning keeps a clear advantage at every size.*

On sparse pruning methods, however, prolonged training from scratch only performs comparably with or worse than the pruning pipeline. For Wanda-2:4, S250 has lower perplexity than P200-R50, yet underperforms P200-R50 on 6 of 8 accuracy benchmarks, including 1.8% and 1.2% degradation on Winogrande and ARC Challenge respectively. On Wanda-U, S250 underperforms P200-R50 on 6 of 8 benchmarks, except BoolQ and OpenBookQA (improvements of 0.4% and 0.8%). This suggests that at 2:4 semi-structured sparsity training from scratch reaches parity with pruning, while at unstructured (the finest) sparsity pruning initialization keeps a small edge even with matched total

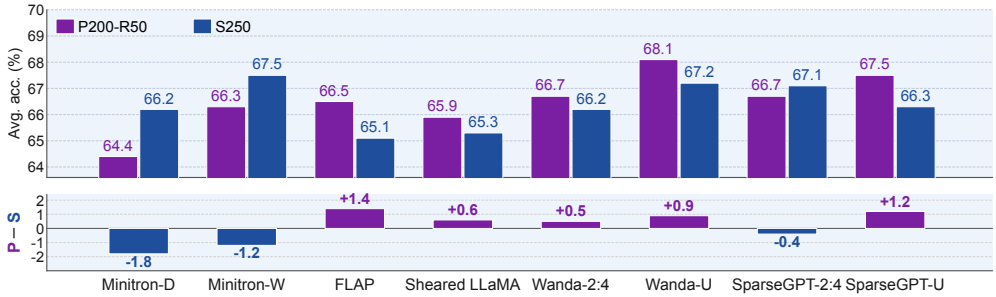


Figure 5. P200-R50 versus S250 across pruning methods. We plot the average accuracy (WinoGrande, ARC-C, ARC-E, HellaSwag, PIQA, SciQ, BoolQ, OBQA) from Table 16. Blue bars denote S250 scratch training, and purple bars denote P200-R50 pruning followed by retraining, matching the color convention in Figure 1. The bottom panel shows P200-R50 minus S250 (P−S); positive values indicate that pruning remains better after matching the total token budget.

data. These margins (0.5–1.2 points in average accuracy) fall within evaluation noise on several benchmarks, and we do not measure run-to-run variance, so we view this edge as suggestive.

4.3 Finer Granularity Unlocks Greater Benefits

Finding 3. At the same pruning ratio, finer pruning granularity yields a stronger initialization and a larger advantage over training from scratch, at the cost of hardware efficiency.

Under 50% pruning ratio, finer pruning granularity gives the pruning pipeline more advantage. From Table 16, we can observe that for structured pruning methods, Minitron-W in general gives P200-R50 over S50 more advantage compared to Minitron-D; for sparse pruning methods, unstructured sparsity provides a stronger pruned model than 2:4 sparsity. For P200-R50 and S250, longer training from scratch surpasses pruning-retraining on structured methods, is roughly on par at 2:4 sparsity, and remains slightly behind unstructured sparse pruning. Taken together, these findings suggest that under the same pruning ratio, finer granularity yields a stronger pruned model for retraining, one that outperforms the same architecture trained from scratch. These observations correspond to previous observations that finer-grained pruning methods are more capable of preserving the base model's performance, no matter whether retraining is performed or not (An et al., 2024; Sun et al., 2024; Sreenivas et al., 2024). We provide a mechanistic discussion of why sparse pruning remains uniquely hard to match by training from scratch in Appendix G.

The superior performance of sparse pruning is at the price of efficiency. Table 9 (Appendix F) compares the training efficiency of the methods we explored (perplexities are in Table 16). All step times are profiled on Google TPU v4-256, which lacks sparse tensor cores. On this hardware, both unstructured 50% sparse and 2:4 sparse models run at virtually the same speed as the dense model (1.0×).[1] Minitron-D achieves the highest training speedup under the same pruning ratio but at the cost of higher perplexity, while Minitron-W offers a middle ground. These observations imply that for practitioners who want compact, general-purpose models that are efficient on commodity hardware, structured pruning is the practical choice, yet this is where training from scratch can match the pruning pipeline under the full token budget, motivating more token-efficient training recipes.

5 Related Work

Our work draws on three lines of research; we summarize them here and extend in Appendix B.

[1] On NVIDIA GPUs with sparse tensor cores (e.g., A100, H100), 2:4 sparsity yields meaningful inference speedups via cuSPARSELt acceleration (Mishra et al., 2021), whereas unstructured sparsity does not. This hardware dependence does not affect the training-time comparison reported here, but practitioners targeting GPU inference should account for it.

Scaling under data constraints. As models and datasets grow to hundreds of billions of parameters and trillions of tokens (DeepSeek-AI, 2024; Qwen Team, 2025; Penedo et al., 2024; Li et al., 2024), data availability rather than compute increasingly bottlenecks pretraining, motivating more token-efficient recipes (Kaplan et al., 2020; Kim et al., 2025; Muennighoff et al., 2023). This is exactly the regime our study targets.

Continual pretraining. Our retraining stage is effectively a form of continual pretraining (CPT) (Sun et al., 2020), where *catastrophic forgetting* (McCloskey & Cohen, 1989) and careful learning-rate re-warming are central concerns (Gupta et al., 2023; Grattafiori et al., 2024; Ibrahim et al., 2024). These works inform the training recipe used in our experiments.

Understanding network pruning. A line of work studies whether pruned subnetworks can be matched by training from scratch, from the Lottery Ticket Hypothesis (Frankle & Carbin, 2019; Evci et al., 2020) to pruning-scaling laws (Xu et al., 2024; Chen et al., 2025). Most directly, Liu et al. (2019) found that for structured pruning, training the target architecture from scratch can match pruning and fine-tuning, framing pruning as a form of neural architecture search. We carry this question into the LLM regime which has far larger scale, costlier parents, and a sharper need for small deployable models. We find the answer hinges on pruning granularity. Extended discussion of these findings and the limitations of our study are deferred to Appendix A.

6 Conclusion

Treating pruning as an initialization strategy rather than a compression tool, we ran a token-matched comparison against training from scratch. Under an equal training token budget, pruning initialization wins, with the margin shrinking at higher pruning ratios for depth pruning. Under an equal total token budget, the outcome splits along granularity: a longer scratch run overtakes structured pruning, while sparse pruning keeps a small lead that FLOP-matched, multi-seed comparisons should confirm.

Reproducibility Statement

We describe the experimental design, base model, data pipeline, and training recipe in Section 3.1, with the full set of training hyperparameters in Appendix D.1. Formal descriptions of all six pruning methods are in Appendix C.2, and our implementation choices (including the GQA adaptations of FLAP and Sheared LLaMA) are in Section 3.2. Calibration sets, per-method configurations, and architecture-search candidates are given in Appendices D.2 and D.3, and the resulting architectures are listed in Table 1. The evaluation protocol, benchmarks, and standard-deviation estimates are given in Appendix E. The data (DCLM-Baseline-1.0), base model (Llama-3.1-8B), training framework (Lingua), and evaluation harness (lm-evaluation-harness) are all public. We will release our code, including the pruning implementations and training configurations, upon publication.

Ethics Statement

This work studies initialization strategies for training small language models and does not involve human subjects, personally identifiable data, or new data collection. All experiments use publicly released models and datasets under their respective licenses. Models trained on web-scale corpora such as DCLM may inherit biases and harmful content from their training data; our pruned and scratch-trained models are research artifacts for comparing training strategies, not for deployment.

AI Use Statement

In this work, we used Claude Code to assist in implementing methods, specifically the modifications that adapt Sheared LLaMA to grouped-query attention. We did not use generative AI tools to design the research methodology or experiments: all experimental settings, including the token-matched comparisons, data splits, and baselines, were designed manually by the authors to ensure a rigorous comparison. We also did not use them to formulate hypotheses or interpret results, and all other method implementations were written by the authors; the remaining required-disclosure tasks are not applicable to this work. Additionally, we used Claude Code to create and edit the software for orchestrating our TPU training jobs, to modify scientific figures, and to polish the writing of the

paper. We have reviewed all AI-assisted work: all AI-assisted code was reviewed and tested by the authors, and the adapted Sheared LLaMA implementation was verified to achieve reasonable pruning performance on Llama-3.1 (Section 3.2). All AI-polished text and AI-modified figures were checked by the authors for accuracy against our experimental results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

AI-Hypercomputer. Maxtext: A simple, performant and scalable jax llm library, 2025. URL https://github.com/AI-Hypercomputer/maxtext.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *EMNLP*, 2023.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *AAAI*, 2024.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *ICLR*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan LeBras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *ACL*, 2016.

Xiaodong Chen, Yuxuan Hu, Xiaokang Zhang, Yanling Wang, Cuiping Li, Hong Chen, and Jing Zhang. P^2 law: Scaling law for post-training after model pruning. In *ACL*, 2025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

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.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

EleutherAI. A framework for few-shot language model evaluation. https://github.com/EleutherAI/lm-evaluation-harness, 2021.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *ICML*, 2023.

Google. A new era of intelligence with gemini 3. 2025a. URL https://blog.google/products-and-platforms/products/gemini/gemini-3/.

Google. Gemma 3 technical report, 2025b.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. Continual pre-training of large language models: How to (re)warm your model? *arXiv preprint arXiv:2308.04014*, 2023.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *NeurIPS*, 2024.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal brain surgeon: extensions and performance comparisons. In *NeurIPS*, 1993.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *NeurIPS*, 2022.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, dahai li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *COLM*, 2024.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n:m transposable masks. In *NeurIPS*, 2021.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *TMLR*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Bo-Kyeong Kim, Geon min Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. Shortened llama: A simple depth pruning for large language models. In *ICLR ME-FoMo Workshop*, 2024.

Konwoo Kim, Suhas Kotha, Percy Liang, and Tatsunori Hashimoto. Pre-training under infinite compute. *arXiv preprint arXiv:2509.14786*, 2025.

Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *NeurIPS*, 1989.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Jeffrey Li, Alex Fang, Georgios Smyrnis, et al. Datacomp-LM: In search of the next generation of training sets for language models. In *NeurIPS*, 2024.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE/ACM TASLP*, 2025.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *NeurIPS*, 2023.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*. 1989.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of ACL*, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

Meta AI. Introducing meta llama 3.1, 2024. URL https://ai.meta.com/blog/meta-llama-3-1.

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 *EMNLP*, 2018.

Asit K. 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.

Niklas Muennighoff, Alexander M Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. In *NeurIPS*, 2023.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. In *NeurIPS*, 2024.

Arijit Nag, Soumen Chakrabarti, Animesh Mukherjee, and Niloy Ganguly. Efficient continual pre-training of llms for low-resource languages. In *NAACL Industry Track*, 2025.

OLMo Team. Olmo 3, 2025.

Jupinder Parmar, Sanjeev Satheesh, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Reuse, don't retrain: A recipe for continued pretraining of language models. *arXiv preprint arXiv:2407.07263*, 2024.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data, and web data only. In *NeurIPS Datasets and Benchmarks*, 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *NeurIPS*, 2024.

Qwen Team. Qwen3 technical report, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*, 2020.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, et al. Dolma: an open corpus of three trillion tokens for language model pretraining research. In *ACL*, 2024.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and jae-joon kim. SLEB: Streamlining LLMs through redundancy verification and elimination of transformer blocks. In *ICML*, 2024.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Llm pruning and distillation in practice: The minitron approach. *arXiv preprint arXiv:2408.11796*, 2024.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *ICLR*, 2024.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. Ernie 2.0: A continual pre-training framework for language understanding. In *AAAI*, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. In *NeurIPS*, 2024.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *W-NUT*, 2017.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In *ICLR*, 2024.

Zhiqiu Xu, Yanjie Chen, Kirill Vishniakov, Yida Yin, Zhiqiang Shen, Trevor Darrell, Lingjie Liu, and Zhuang Liu. Initializing models with larger ones. In *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 ACL*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *ACL*, 2019.

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 *ICLR*, 2021.

Appendix

This appendix provides additional methodology details, configurations, and per-benchmark results supporting the main paper:

- § A extends the conclusion with a discussion of what transfers from the larger model and enumerates the axes of variation our study leaves unexplored.
- § B gives the full related-work discussion condensed in the main paper.
- § C catalogues the pruning methods we evaluate, with formal scoring and reconstruction objectives for each.
- § D documents the training recipe, calibration sets, pruning ratios, and the architecture-search procedure used to pick a single configuration per method.
- § E specifies the perplexity corpora, downstream benchmarks, and the binomial standard error we use to mark within-noise differences.
- § F reports training-speed comparisons of the pruning methods we evaluate.
- § G discusses why sparse pruning remains hard to match by training from scratch, drawing on prior work on lottery tickets and weight importance.
- § H gives the full per-benchmark breakdown, including pruning-only baselines, pruning ratio ablations, retraining-token budgets, and the equal-total-token-budget setting.

A Extended Discussion and Limitations **15**

B Extended Related Work **15**

C Pruning Methods **16**

C.1 Overview 16

C.2 Method details 16

D Implementation Details **18**

D.1 Training recipe 18

D.2 Pruning configurations 18

D.3 Architecture search 19

E Evaluation protocol **19**

F Efficiency comparison **20**

G Why Does Sparse Pruning Maintain Its Advantage? **21**

H Detailed Per-Benchmark Results **22**

H.1 Pruning-only baselines 22

H.2 Pruning ratio ablations 22

H.3 Token scaling across pruning methods 22

H.4 Pruning vs. scratch under equal total token budget 23

A Extended Discussion and Limitations

Discussion. The difference between sparse and structured pruning suggests that what transfers from the larger model may be *weight-level* information rather than architectural shape: sparse pruning keeps the original parameter values, whereas a smaller dense architecture trained for longer can match a structurally pruned counterpart. For structured pruning, the benefit is therefore largest when the training token budget is the binding constraint; with more tokens, a longer scratch run on a smaller dense model is competitive. Sparse pruning keeps a small edge at the full pipeline budget, but it also has the weakest hardware support: unstructured sparsity gives no speedup on the TPUs we used and only a partial speedup on GPUs with sparse tensor cores. Closing this gap will likely require better token-efficient recipes for structured small models or broader hardware and kernel support for unstructured sparsity. In practice, the right pipeline depends on which resource is scarce. When the token budget for the target model is the bottleneck, pruning beats a scratch run of equal length. When the full pipeline budget is counted, the case for pruning is weaker, and results reported under a fixed retraining budget may overstate its benefit once the cost of pretraining the parent is included.

Limitations. Our study leaves five axes of variation unexplored. (1) *Model family.* All experiments use Llama-3.1-8B as the base; we do not test whether the same pruning-vs-scratch ordering holds for other architectures (e.g. Qwen (Qwen Team, 2025), Gemma (Google, 2025b), OLMo (OLMo Team, 2025)) or for substantially smaller and larger base models. (2) *Pretraining corpus.* Both the P200 checkpoint and all retraining runs use DCLM. We do not vary the corpus, so the gap between sparse and structured pruning may shift under different data distributions such as FineWeb-Edu (Penedo et al., 2024) or Dolma (Soldaini et al., 2024), and the Meta-RN comparisons control for architecture but not for the original Meta pretraining mixture. (3) *Knowledge-distillation baselines.* We compare pruning to plain language-modeling retraining; we do not include the post-pruning knowledge-distillation pipelines used by recent structured methods (e.g. Minitron, Sheared LLaMA), which could narrow or close the gap for structured pruning under matched total data. (4) *Data order.* Each setting is trained with a single shuffle of the retraining corpus; we do not measure run-to-run variance or the sensitivity of the pruning advantage to data ordering, which can be non-trivial in low-token regimes. (5) *Random-initialization strategies.* Our scratch baselines use a single standard initialization; alternative schemes such as μP, depth-aware variance scaling, or warm-starting from a smaller pretrained checkpoint may shift the scratch curve and could change which pruning methods retain an advantage over training from scratch.

B Extended Related Work

This section gives the full treatment of the three lines of research summarized in Section 5.

Model scaling and constraints. The traditional language model scaling paradigm assumes that simple scaling of model size and training data results in stronger performance (Kaplan et al., 2020), however, as recent language models scale to hundreds of billions of parameters (DeepSeek-AI, 2024; Qwen Team, 2025; Google, 2025a) and training data scales to trillions of tokens (Penedo et al., 2024; Li et al., 2024), data availability has become a practical challenge in scaling language model pretraining. Recent works on pretraining under limited data and unlimited compute have explored the effect of traditional scaling method in this scenario. Kim et al. (2025) show that existing approaches such as increasing epoch and parameter count eventually overfit and improved training recipes are needed for better scaling performance. Muennighoff et al. (2023) observed similar plateau of scaling training epochs and proposed a compute optimality scaling law. These results suggest that data, rather than compute, is becoming the main bottleneck in language model pretraining, which motivates more token-efficient training methods.

Continual pretraining. As the domain-specific capabilities of LLMs become increasingly strong, *continual pretraining* (CPT) (Sun et al., 2020) has emerged as an approach to adapt general-purpose language models for domain specific task. The primary challenge in CPT is *catastrophic forgetting* (McCloskey & Cohen, 1989; Luo et al., 2025), where the model loses its prior knowledge and capabilities during the continual learning stage, and the learning rate schedule must be carefully designed to mitigate this phenomenon. Gupta et al. (2023) observe that starting CPT with a high learning rate causes a sharp drop in model's performance, and rewarming the LR to 10–20% of the

max learning rate significantly mitigates this problem. Grattafiori et al. (2024) also suggested that learning rate schedule with re-warmup has been adopted to improve stability when continual pretraining on a new domain. Ibrahim et al. (2024) provide a systematic study of data replay and learning rate re-warming strategies in CPT, showing that mixing a small fraction of original pretraining data prevents catastrophic forgetting. These works provide valuable reference for the design of training recipes in our experiments, since network pruning damages the internal structure of the base model and the retraining stage is effectively a form of CPT.

Understanding network pruning. Since network pruning enables efficient acquisition of compact and capable models, its properties and impacts on the pruned models' behaviors have been a concerning topic in the network pruning community. The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) conjectures that every randomly initialized network contains a sparse subnetwork that can be trained to match the full network's performance, provided that it is trained from its *original* initialization, since random reinitialization substantially degrades performance. Evci et al. (2020) further show that static sparse training from scratch can get stuck in poor local minima, whereas dynamic sparse topologies help escape them. Together, these results suggest that a sparse mask derived from a pretrained model is difficult to match via random reinitialization. Xu et al. (2024) observe that initializing small models from large pretrained models with rule-based weight selection yields better training performance. Liu et al. (2019) show that small models with the same architecture trained from scratch have on-par or stronger performance than their pruned-and-retrained counterparts for structured methods, and interpret network pruning as a form of Neural Architecture Search (NAS). Chen et al. (2025) propose a pruning-scaling law connecting training data quality and post-pruning budget to final performance. These prior works have greatly inspired the experimental design of this work.

C Pruning Methods

C.1 Overview

Table 3 summarizes representative LLM pruning methods organized by pruning granularity. We group methods into four categories: *depth* pruning removes entire transformer layers; *width* pruning reduces the hidden dimension, FFN intermediate size, or number of attention heads; *depth+width* pruning jointly compresses along both axes; and *sparse* pruning zero-masks individual weights within linear modules while keeping the macro-architecture intact. For each method, we list the modules it targets and the scoring metric it uses to decide which units to prune. This taxonomy motivates the six representative methods evaluated in our experiments (Minitron-depth, Minitron-width, FLAP, Sheared LLaMA, Wanda, and SparseGPT), which together span all four granularity regimes.

C.2 Method details

Preliminaries. Let $\theta \in \mathbb{R}^d$ denote the parameters of a pretrained LLM and $\mathcal{D}_{\text{cal}}$ a small calibration corpus. Pruning at ratio r produces a smaller model by selecting a binary mask $m \in \{0,1\}^d$ that retains a fraction $1-r$ of the prunable components (whole layers, channels, or individual weights). Existing LLM pruning methods fall into two families based on how m is chosen.

(i) Loss / reconstruction minimization. These methods select m to minimize an objective on $\mathcal{D}_{\text{cal}}$,

$$m^\star \;=\; \arg\min_{m \in \mathcal{M}_r} \; \mathcal{L}\big(\theta \odot m;\, \mathcal{D}_{\text{cal}}\big), \tag{1}$$

where $\mathcal{M}_r$ is the feasible mask set at ratio r and $\mathcal{L}$ is either the next-token validation loss (Minitron-depth, Sheared LLaMA) or a layerwise output-reconstruction error (SparseGPT).

(ii) Activation-based importance scoring. These methods assign each prunable component i a scalar score s_i summarizing its contribution to the output activations on $\mathcal{D}_{\text{cal}}$, and retain the top-k highest-scoring components with $k = \lceil (1-r)\, d \rceil$:

$$m_i^\star \;=\; \mathbb{1}\big[\, i \in \mathrm{TopK}\big(\{s_j\}_{j=1}^{d},\, k\big)\,\big]\,. \tag{2}$$

The unit indexed by i may be a channel (Minitron-width, FLAP) or an individual weight (Wanda).

Table 3. A taxonomy of different LLM pruning approaches categorized by pruning granularity. Depth pruning removes layers, width pruning prunes attention heads / hidden / FFN dimensions, and sparse pruning zero-masks individual weights without changing the model shape.

Pruning Method	Target Modules	Pruning Metric
→ ***Depth***		
ShortGPT (Men et al., 2025)	layers	$1 - \mathbb{E}_{h,t} \frac{h_{i,t}^T h_{i+1,t}}{\|h_{i,t}\|_2 \|h_{i+1,t}\|_2}$
Minitron-depth (Sreenivas et al., 2024)	layers	$\mathcal{L}(\theta, z) - \mathcal{L}(\theta)$
Shortened LLaMA (Kim et al., 2024)	layers	$\left\| \frac{\partial \mathcal{L}(D)}{\partial W_{i,j}^{k,n}} W_{i,j}^{k,n} \right\|$
→ ***Width***		
LLM-Pruner (Ma et al., 2023)	hidden size, FFN size, attention heads	$\left\| \frac{\partial \mathcal{L}(\mathcal{D})}{\partial W_i^k} W_i^k - \frac{1}{2} \sum_{i=1}^{N} \left(\frac{\partial \mathcal{L}(\mathcal{D})}{\partial W_i^k} W_i^k \right)^2 + \mathcal{O}\left(\|W_i^k\|^3 \right) \right\|$
Minitron-width (Sreenivas et al., 2024)	hidden size, FFN size	$\|X_{:,j}\|^2, \quad 1 \le j \le d$
Wanda-sp (An et al., 2024)	hidden size, FFN size	$\sum_i \|W_{ij}\| \cdot \|X_{:,j}\|_2$
FLAP (An et al., 2024)	hidden size, FFN size	$\frac{1}{N-1} \sum_{n=1}^{N} (X_{n,j}^l - \bar{X}_{:,j}^l)^2 \cdot \|W_{:,j}^l\|_2^2$
SliceGPT (Ashkboos et al., 2024)	hidden size	$\lambda_j(\mathbf{X}^{l\top} \mathbf{X}^l), \quad 1 \le j \le d$
→ ***Depth + Width***		
Sheared LLaMA (Xia et al., 2024)	layers, hidden size, FFN size, attention heads	$\mathcal{L}(\theta, z) + \sum_{j=1}^{L_S} \tilde{\mathcal{L}}_j^{\text{head}} + \sum_{j=1}^{L_S} \tilde{\mathcal{L}}_j^{\text{int}} + \tilde{\mathcal{L}}^{\text{layer}} + \tilde{\mathcal{L}}^{\text{hidden}}$
→ ***Sparse***		
Magnitude (Han et al., 2015)	individual weights	$\|W_{ij}\|$
SparseGPT (Frantar & Alistarh, 2023)	individual weights	$\frac{\|W\|^2}{\text{diag}[(X^T X + \lambda I)^{-1}]}$
Wanda (Sun et al., 2024)	individual weights	$\|W_{ij}\| \cdot \|X_j\|_2$

Minitron-depth (Sreenivas et al., 2024) prunes a contiguous block of k transformer layers, with k determined by the pruning ratio. For each candidate starting index i, the validation loss is recomputed with layers $[i,\, i+k)$ replaced by identity mappings, and the block causing the smallest loss increase is removed:

$$i^\star \;=\; \arg\min_i \;\; \mathcal{L}\big(\theta_{[i:i+k]\to \mathrm{id}};\, \mathcal{D}_{\mathrm{cal}}\big) \;-\; \mathcal{L}(\theta;\, \mathcal{D}_{\mathrm{cal}}). \tag{3}$$

Minitron-width (Sreenivas et al., 2024) scores each hidden or FFN channel j by the L2 norm of its block-output activations on $\mathcal{D}_{\mathrm{cal}}$, and keeps the top-scoring channels up to the target dimension:

$$s_j \;=\; \big\|X_{:,j}\big\|_2, \qquad 1 \le j \le d. \tag{4}$$

FLAP (An et al., 2024) scores each hidden / FFN channel by the variance of its input feature, weighted by the squared norm of the corresponding output-weight column:

$$s_j \;=\; \frac{1}{N-1} \sum_{n=1}^{N} \big(X_{n,j} - \bar{X}_{:,j}\big)^2 \cdot \big\|W_{:,j}\big\|_2^2. \tag{5}$$

Channels with the largest s_j are retained.

Sheared LLaMA (Xia et al., 2024) introduces continuous mask variables z over layers, attention heads, hidden channels, and FFN channels, and learns them jointly with the model parameters via a constrained optimization. The target-architecture constraints are relaxed into an augmented Lagrangian penalty:

$$\min_{\theta,\, z} \;\; \mathcal{L}\big(\theta \odot z;\, \mathcal{D}\big) \;+\; \sum_{c \in \mathcal{C}} \Big[\lambda_c \big(\hat{s}_c(z) - s_c^\star \big) \;+\; \phi_c \big(\hat{s}_c(z) - s_c^\star \big)^2 \Big], \tag{6}$$

where c ranges over architectural axes, $\hat{s}_c(z)$ is the current size along axis c, $s_c^\star$ is the target, and (λ_c, ϕ_c) are dual variables updated jointly with z.

Wanda (Sun et al., 2024) scores each individual weight W_{ij} by its magnitude times the L2 norm of the corresponding input feature on $\mathcal{D}_{\mathrm{cal}}$, and keeps the top-scoring weights within each output row:

$$s_{ij} \;=\; |W_{ij}| \cdot \big\|X_{:,j}\big\|_2. \tag{7}$$

For unstructured sparsity, weights are compared within each output row; for 2:4 sparsity, they are compared within each group of four consecutive weights in a row, keeping the top two. Neither pattern requires weight updates after pruning.

SparseGPT (Frantar & Alistarh, 2023) formulates pruning as a layerwise reconstruction problem. For each linear layer with weight W and calibration input X, it jointly searches for a binary mask m and updated weights $\hat{W}$ that minimize the output mismatch:

$$\min_{m,\,\hat{W}} \left\| XW^\top - X(\hat{W} \odot m)^\top \right\|_F^2 \quad \text{s.t.} \quad \|m\|_0 \le (1-r)\,|W|. \tag{8}$$

The combinatorial mask is approximated by an OBS-style score $|W_{ij}|^2/[(X^\top X + \lambda I)^{-1}]_{jj}$, and surviving weights are updated through the inverse Hessian to compensate for those removed.

D Implementation Details

D.1 Training recipe

All pretraining and retraining experiments use the default Lingua (Videau et al., 2024) recipe. Key hyperparameters are summarised in Table 4.

Table 4. Training recipes of our pretraining and retraining experiments.

config	value
sequence length	8192
batch size	512
weight decay	0.01
warmup ratio	5%
learning rate schedule	cosine decay
min learning rate ratio	0.1
optimizer	AdamW
optimizer momentum	$\beta_1 = 0.9$, $\beta_2 = 0.95$
optimizer eps	1e-8

Learning rate schedule. A cosine decay schedule is used for all runs. The warmup phase occupies 5% of total steps, after which the learning rate decays to 10% of its peak value. For each experiment we sweep the peak learning rate over {`1e-5, 3e-5, 1e-4, 3e-4, 1e-3`} using a short trial prefix and select the value that yields the lowest training loss.

D.2 Pruning configurations

We describe the calibration sets and configurations for each pruning method.

Minitron-depth. Calibration set: 1024 samples of 8192 tokens from WikiText-103, used to compute validation loss. Pruning ratios $\{50\%, 62.5\%, 75\%, 81.3\%\}$ correspond to 16, 8, 4, and 2 hidden layers in the pruned model.

Minitron-width. Same calibration set as Minitron-depth. L2-norm of block output activations is used to score each hidden channel and MLP channel. Pruning ratios $\{50\%, 62.5\%, 75\%, 81.3\%\}$. Since Minitron-width prunes hidden-size and MLP-size dimensions independently, multiple target architectures are possible for a given parameter budget. To select the final configuration at each pruning ratio, we enumerate several candidate dimension pairs (hidden size, MLP size) that yield roughly the same total parameter count, following the original Minitron paper in keeping attention heads fixed at 32 and constraining $2h \le \text{MLP} \le 4h$. We prune the pretrained Llama-3.1-8B checkpoint to each candidate and evaluate validation loss on the calibration set without retraining. The candidate with the **lowest validation loss** is used for all subsequent retraining. The selected configurations are listed in Table 1.

FLAP. Calibration set: 128 samples of 8192 tokens from C4. Channels are scored by input feature variance weighted by the corresponding weight column norm, and pruned at a 50% ratio.

Table 5. Candidate architectures for **Minitron-W**. Attention heads are fixed at 32; we vary hidden size with $2h \leq \text{MLP} \leq 4h$. The selected configuration at each pruning ratio is marked with ✓.

Ratio	Hidden	Attn heads	MLP	Params	Sel.
62.5%	2176	32	7168	3.1B	
	2304	32	6656	3.1B	
	2432	32	6144	3.1B	✓
	2560	32	6400	3.1B	
	2688	32	5888	3.1B	
75%	1664	32	6656	2.0B	
	1792	32	6016	2.0B	✓
	1920	32	5248	2.0B	
	2048	32	4480	2.0B	
81.3%	1408	32	5632	1.5B	
	1536	32	4736	1.5B	✓
	1664	32	3840	1.5B	

Sheared LLaMA. We specify a target architecture matching a 50% pruning ratio and jointly learn pruning masks over layers, attention heads, hidden dimensions, and feedforward dimensions on the training data. Following the original paper, this mask-learning stage uses 0.4B tokens, under 0.2% of the 250B-token pipeline budget.

SparseGPT and Wanda. Calibration set: 128 samples of 8192 tokens from C4. Both 2:4 semi-structured and 50% unstructured sparsity patterns are applied.

D.3 Architecture search

For **Minitron-W**, **FLAP**, and **Sheared LLaMA**, the pruning method does not uniquely determine the output architecture for a given parameter budget: multiple dimension configurations can yield the same total parameter count. For each method we enumerate a set of candidate architectures that match the target budget, prune the pretrained Llama-3.1-8B checkpoint to each candidate without any retraining, and evaluate validation loss on the calibration set. The candidate with the **lowest validation loss** is selected and used for all subsequent retraining experiments in this paper.

Minitron-W. Following the original Minitron paper, attention heads are kept unchanged (32 heads). We vary the hidden size and constrain MLP size such that $2h \leq \text{MLP} \leq 4h$. The full candidate set and selected configuration are listed in Table 5.

FLAP. Following the UL-MM configuration in the FLAP paper, hidden size is kept unchanged at 4096. We vary the number of query heads and MLP size to reach the target of ∼4.5B parameters, consistent with **Minitron**, **Wanda**, and **SparseGPT**. Candidates and the selected configuration are listed in Table 6.

Sheared LLaMA. We vary attention heads, hidden size, and MLP size to reach ∼4.5B parameters. Because varying all three dimensions simultaneously yields too many combinations, we fix hidden size to two representative values (3072 and 3456) and vary attention heads and MLP size within each. The full candidate set and selected configuration are listed in Table 7.

E Evaluation protocol

Linguistic perplexity. We evaluate on the general-domain corpora C4 (Raffel et al., 2020), WikiText-103 (Merity et al., 2017), and WikiText-2 (Merity et al., 2017), along with the news-and-summaries corpus CNN Dailymail (Chen et al., 2016). For each corpus, we collect 256 sequences of length 8192 (our training sequence length) as the evaluation set.

Table 6. Candidate architectures for **FLAP**. Following the UL-MM config in the FLAP paper, hidden size is fixed at 4096; query heads and MLP size are varied. The selected configuration is marked with ✓.

Ratio	Hidden	Query heads	MLP	Params	Sel.
50%	4096	16	7168	4.5B	
	4096	20	6656	4.5B	✓
	4096	24	6272	4.5B	
	4096	28	5888	4.5B	
	4096	32	5376	4.5B	

Table 7. Candidate architectures for **Sheared LLaMA**. Hidden size is fixed to 3072 or 3456; attention heads and MLP size are varied within each. The selected configuration is marked with ✓.

Ratio	Hidden	Attn heads	MLP	Params	Sel.
50%	3072	16	11008	4.5B	
	3072	20	10496	4.5B	
	3072	24	10112	4.5B	
	3072	28	9728	4.5B	
	3072	32	9216	4.5B	
	3456	16	9216	4.5B	
	3456	20	8832	4.5B	
	3456	24	8448	4.5B	
	3456	28	7936	4.5B	
	3456	32	7552	4.5B	✓

Table 8. Per-benchmark binomial standard errors of accuracy.

Benchmark	**Std. err.**
WinoG	±1.3%
ARC-C	±1.4%
ARC-E	±1.0%
HSwag	±0.4%
PIQA	±1.0%
SciQ	±0.8%
BoolQ	±0.8%
OBQA	±2.2%

Downstream accuracy. We evaluate on eight zero-shot benchmarks spanning three categories: commonsense QA (WinoGrande (Sakaguchi et al., 2020), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020)), scientific QA (ARC-Challenge (Clark et al., 2018), ARC-Easy (Clark et al., 2018), SciQ (Welbl et al., 2017), OpenBookQA (Mihaylov et al., 2018)), and logical reasoning (BoolQ (Clark et al., 2019)). Zero-shot evaluation measures foundational knowledge and capability, independent of in-context learning biases. The random-chance baselines are 50% for WinoGrande (binary), 25% for ARC-C and ARC-E (four-way), and 50% for PIQA and BoolQ (binary); results near these baselines indicate that the model has essentially lost the measured capability, and we interpret them accordingly.

Evaluation framework and standard errors. All downstream evaluations are run with lm-evaluation-harness (EleutherAI, 2021). Grey accuracy entries in Tables 11 and 16 denote differences smaller than the benchmark's binomial standard error, $\sqrt{acc(1-acc)/N}$, where N is the number of evaluation examples (Table 8). This is the sampling error of a single accuracy estimate; it is not training-seed variance, nor a formal test for a paired difference between two models.

F Efficiency comparison

Table 9 compares training speed among the pruning methods we experiment with; WikiText-103 (Merity et al., 2017) perplexities are in Table 16. Minitron-D (Sreenivas et al., 2024), the coarsest gran-

Table 9. Efficiency comparison between pruning methods. Models are obtained by pretraining Llama-3.1-8B for 200B tokens, pruning at the listed ratio, and retraining for 50B tokens. For Wanda and SparseGPT, 50% denotes unstructured sparsity. FLOPs are computed for a single forward pass with sequence length 2048; for sparse models, theoretical FLOPs assume 50% of weight multiplications are skipped. **All step times are profiled on Google TPU v4-256 using MaxText (AI-Hypercomputer, 2025), which has no sparse tensor cores.** On this hardware, sparse methods yield no training speedup despite halved theoretical FLOPs. On NVIDIA GPUs with sparse tensor cores (A100/H100), 2:4 sparse models obtain meaningful speedups via cuSPARSELt; unstructured sparse models do not.

method	sparsity	FLOPs (TF)	Speedup
base	0.0%	32.9	1.0×
Minitron-D	50.0%	17.5	1.9×
	62.5%	9.8	4.2×
	75.0%	6.0	7.2×
	81.3%	4.1	11.2×
Minitron-W	50.0%	19.1	1.4×
	62.5%	12.6	2.0×
	75.0%	9.8	2.4×
	81.3%	7.9	2.9×
FLAP	50.0%	17.7	1.5×
Sheared LLaMA	50.0%	18.9	1.4×
Wanda	50.0%	16.5	1.0×
	2:4	16.5	1.0×
SparseGPT	50%	16.5	1.0×
	2:4	16.5	1.0×

ularity, gives the largest speedup but the highest WikiText-103 perplexity among the 50%-pruned models. The sparse methods, SparseGPT (Frantar & Alistarh, 2023) and Wanda (Sun et al., 2024), reach lower perplexity but give no training speedup on our hardware. Minitron-W (Sreenivas et al., 2024) lies in between, with a moderate speedup and intermediate perplexity.

G Why Does Sparse Pruning Maintain Its Advantage?

Sparse pruning keeps a small advantage over training from scratch even when scratch training uses the full 250B-token budget (Section 4.3). Prior work suggests two possible explanations.

Optimizing a sparse model from scratch is hard. The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) posits that a pruned subnetwork trained from its *original* initialization converges faster and to higher accuracy than the same sparse structure trained from a *random* reinitialization. Evci et al. (2020) further demonstrate that static sparse training can get stuck in isolated local minima, and that allowing the sparse topology to evolve during training helps escape them. Our S250 sparse baselines use a fixed mask with randomly reinitialized weights, which is the setting these works identify as hard to optimize.

Sparse pruning identifies and preserves non-redundant weights. Liu et al. (2019) observe that both fine-tuned and scratch-trained pruned models have far fewer near-zero weights than the unpruned base model, suggesting that pruning actively identifies structurally important weights. For unstructured sparse pruning in particular, the fine-tuned model diverges more strongly from the scratch-trained counterpart than structured pruning does, suggesting that the inherited weight *values*, and not only the sparsity pattern, contribute to the advantage.

H Detailed Per-Benchmark Results

H.1 Pruning-only baselines

To contextualize the benefit of retraining, we report zero-shot performance immediately after pruning, before any retraining is applied. Table 10 covers both structured pruning (Minitron-D, Minitron-W, FLAP, Sheared LLaMA) and sparse pruning (Wanda and SparseGPT at 2:4 and unstructured 50% sparsity). Some structured configurations have very high perplexity immediately after pruning; for example, P200 with Minitron-W reaches 24686.5 on C4 and 36761.5 on WikiText-2. After 50B tokens of retraining, however, the same configuration reaches accuracy and perplexity comparable to the other 50%-pruned methods (Table 16), so post-pruning perplexity alone is a poor predictor of the retrained model's quality.

Table 10. Zero-shot performance immediately after pruning, without any retraining. All pruning methods use 50% sparsity. *Minitron-D* and *Minitron-W* denote the depth and width variants of Minitron pruning, respectively. *Wanda-U* and *SparseGPT-U* denote the unstructured sparsity variants of Wanda and SparseGPT (as opposed to the 2:4 semi-structured variants). C4 and WT-2: perplexity (lower is better); others: 0-shot accuracy (%). Avg is the mean over all 8 zero-shot benchmarks (WinoG, ARC-C, ARC-E, HSwag, PIQA, SciQ, BoolQ, OBQA). Purple-shaded rows mark the unpruned base model in each block (Meta's released Llama-3.1-8B and our 200B-token pretrained checkpoint); **purple bold values** mark the best result among the pruned methods within each block.

Base model	Method	perplexity ↓		accuracy(%) ↑								
		C4	WT-2	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
Meta	Base	9.9	5.8	74.3	53.4	81.1	78.9	81.3	96.3	82.0	45.0	74.0
Meta	Minitron-D	532.4	599.9	48.5	24.0	30.6	29.1	54.7	31.1	37.9	27.2	35.4
Meta	Minitron-W	411.0	182.0	52.4	22.6	30.3	28.6	53.0	51.2	37.8	27.4	37.9
Meta	FLAP	85.4	34.2	53.4	25.5	36.8	34.6	57.7	61.9	38.8	30.0	42.3
Meta	Sheared LLaMA	19.1	13.3	61.6	35.0	59.3	57.0	69.9	88.9	65.7	36.2	59.2
Meta	Wanda 2:4	38.0	21.6	59.4	30.5	52.0	49.2	69.0	83.8	67.9	30.8	55.3
Meta	Wanda-U	15.5	9.3	70.4	42.2	**68.3**	68.5	76.4	**90.8**	79.1	40.2	67.0
Meta	SparseGPT-2:4	24.2	15.0	63.5	34.8	59.3	56.5	70.5	87.3	68.2	35.0	59.4
Meta	SparseGPT-U	**14.5**	**8.8**	**72.5**	**43.0**	67.4	**71.4**	**76.5**	87.9	**79.2**	**42.4**	**67.5**
P200	Base	15.8	9.0	68.4	46.3	75.1	74.5	78.8	95.3	69.3	41.0	68.6
P200	Minitron-D	504.0	364.2	53.6	29.9	34.7	36.9	58.8	35.4	53.8	28.6	41.5
P200	Minitron-W	24686.5	36761.5	49.3	26.3	25.8	26.2	51.2	22.8	38.9	27.6	33.5
P200	FLAP	128.0	57.0	51.5	24.8	47.8	38.1	63.9	69.4	62.1	31.2	48.6
P200	Sheared LLaMA	**20.4**	15.2	61.1	41.8	69.4	64.1	74.9	**91.5**	65.8	37.6	63.3
P200	Wanda-2:4	142.8	67.1	57.5	30.5	53.9	52.7	67.6	89.5	63.9	32.0	56.0
P200	Wanda-U	30.9	15.0	63.4	42.4	67.8	**69.1**	76.4	90.3	**66.9**	38.2	64.3
P200	SparseGPT-2:4	42.7	20.3	62.6	35.8	63.5	59.4	73.5	89.1	65.6	37.4	60.9
P200	SparseGPT-U	22.4	**11.9**	**66.5**	**43.9**	**70.2**	68.9	**77.9**	90.4	65.4	**40.2**	**65.4**

H.2 Pruning ratio ablations

Table 11 reports per-benchmark results for Minitron-D and Minitron-W at pruning ratios $\{50.0\%, 62.5\%, 75.0\%, 81.3\%\}$ (equal training token budget). The ΔP200-R50 rows give the advantage of pruning initialization over scratch training; it diminishes with the pruning ratio for depth pruning but not for width pruning.

Per-benchmark observations. BoolQ is the most volatile benchmark across pruning ratios under both granularities: under depth pruning it decays monotonically from $+6.2$ at 50% to -4.7 at 81.3%, while under width pruning it swings non-monotonically between $+6.6$ and $+16.8$. On ARC-Challenge, Minitron-D at 62.5% trails S50 by 1.6 points, although pruning is ahead at both 50% and 75%. Minitron-W keeps a positive average advantage at every ratio we tested (including 81.3%, ΔAvg $= +5.1$), unlike depth pruning, where the average advantage vanishes at 81.3% (-0.2) and only WinoG, ARC-E, PIQA, and SciQ remain positive.

H.3 Token scaling across pruning methods

Tables 12, 13, 14, and 15 report per-benchmark results across retraining token budgets $\{10\text{B}, 30\text{B}, 50\text{B}, 250\text{B}, 500\text{B}\}$ for Minitron-D, Minitron-W, FLAP, and Sheared LLaMA, respectively, under the

Table 11. Effect of pruning ratio on the initialization advantage (equal training token budget). *P200-R50:* pretrain 200B tokens, prune, retrain 50B tokens. *S50:* train from scratch for 50B tokens. ΔP200-R50: advantage of P200-R50 over S50, signed so that positive means P200-R50 is better (lower loss or perplexity, higher accuracy). Blue: P200-R50 better; red: P200-R50 worse; grey: accuracy difference smaller than one binomial standard error (Appendix E), or zero. For depth pruning, the advantage diminishes as the pruning ratio increases and vanishes near 81.3%; width pruning retains a clear advantage at every ratio.

ratio	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
→ **Minitron-*depth***															
50.0%	S50	2.64	16.90	10.61	10.77	10.35	56.8	36.2	70.1	61.9	74.4	90.4	57.8	37.8	60.7
	P200-R50	2.54	15.34	9.30	9.41	9.46	64.4	40.7	71.5	68.0	76.8	91.7	64.0	38.0	64.4
	ΔP200-R50	+0.10	+1.56	+1.31	+1.36	+0.89	+7.6	+4.5	+1.4	+6.1	+2.4	+1.3	+6.2	+0.2	+3.7
62.5%	S50	2.82	20.23	13.43	13.70	12.57	54.0	**32.3**	62.3	51.1	71.4	85.2	51.6	33.8	55.2
	P200-R50	2.77	19.35	12.67	12.89	12.03	55.3	30.7	62.5	53.6	72.3	**87.3**	55.7	34.2	56.4
	ΔP200-R50	+0.05	+0.88	+0.76	+0.81	+0.54	+1.3	-1.6	+0.2	+2.5	+0.9	+2.1	+4.1	+0.4	+1.2
75.0%	S50	3.02	24.57	17.29	17.69	15.55	51.4	27.6	56.2	40.9	68.1	79.5	55.0	29.0	51.0
	P200-R50	3.01	24.41	17.11	17.48	15.49	51.1	29.3	55.8	41.4	68.6	81.9	54.9	30.4	51.7
	ΔP200-R50	+0.01	+0.16	+0.18	+0.21	+0.06	-0.3	+1.7	-0.4	+0.5	+0.5	+2.4	-0.1	+1.4	+0.7
81.3%	S50	3.25	30.79	24.57	25.16	19.97	49.4	25.9	52.2	35.7	65.1	74.4	55.1	28.8	48.3
	P200-R50	3.25	30.79	24.92	25.48	20.33	51.2	24.6	53.5	35.4	66.4	76.0	50.4	27.4	48.1
	ΔP200-R50	0.00	0.00	-0.35	-0.32	-0.36	+1.8	-1.3	+1.3	-0.3	+1.3	+1.6	-4.7	-1.4	-0.2
→ **Minitron-*width***															
50.0%	S50	2.70	17.47	10.99	11.17	10.65	61.3	33.6	65.1	60.3	74.7	89.6	60.8	33.8	59.9
	P200-R50	2.47	14.6	8.93	9.01	8.92	65.4	41.8	73.4	70.5	77.4	93.8	67.4	40.4	66.3
	ΔP200-R50	+0.23	+2.87	+2.06	+2.16	+1.73	+4.1	+8.2	+8.3	+10.2	+2.7	+4.2	+6.6	+6.6	+6.4
62.5%	S50	2.75	18.67	12.00	12.23	11.45	56.1	30.1	65.2	56.3	72.9	86.2	55.7	32.6	56.9
	P200-R50	2.50	14.51	8.49	8.61	9.01	61.5	41.7	71.0	69.5	75.4	92.6	72.4	35.4	64.9
	ΔP200-R50	+0.25	+4.16	+3.51	+3.62	+2.44	+5.4	+11.6	+5.8	+13.2	+2.5	+6.4	+16.7	+2.8	+8.0
75.0%	S50	2.82	20.00	12.99	13.26	12.27	56.3	29.8	61.7	52.6	71.9	87.4	55.7	32.2	56.0
	P200-R50	2.60	15.98	9.61	9.76	9.91	57.2	37.2	68.0	63.5	73.4	89.8	66.1	32.4	61.0
	ΔP200-R50	+0.22	+4.02	+3.38	+3.50	+2.36	+0.9	+7.4	+6.3	+10.9	+1.5	+2.4	+10.4	+0.2	+5.0
81.3%	S50	2.86	20.93	13.77	14.08	12.88	54.1	27.9	60.7	49.5	70.7	86.0	53.2	31.5	54.2
	P200-R50	2.63	16.16	9.53	9.71	10.10	54.0	35.8	65.8	61.2	71.7	88.3	67.7	29.8	59.3
	ΔP200-R50	+0.23	+4.77	+4.24	+4.37	+2.78	-0.1	+7.9	+5.1	+11.7	+1.0	+2.3	+14.5	-1.7	+5.1

three initialization strategies (train from scratch, prune from our 200B-pretrained checkpoint, and prune from Meta's released Llama-3.1-8B). Figure 3 in the main text plots these four tables.

Per-benchmark observations. As the retraining budget grows, individual benchmarks begin to favor scratch training: at 500B tokens, P200-R500 trails S500 on WinoGrande for Minitron-D (-2.0) and on OBQA for Minitron-W (-4.2) and FLAP (-4.6), while BoolQ still favors pruning ($+0.4$, $+7.0$, $+13.5$). The average advantage of P200-RN over SN nonetheless stays positive for all four structured methods at every budget we test ($\Delta\text{Avg} \geq +0.8$). Meta-R$N$ tends to outperform P200-RN at low budgets (e.g. Minitron-D $\Delta_{S10} = +12.1$ vs. $+10.9$), and the gap closes as N grows. One possible reason is that Meta's parent was pretrained on far more, and different, data than our 200B-token DCLM parent.

H.4 Pruning vs. scratch under equal total token budget

Table 16 reports the complete per-benchmark breakdown underlying Figure 5, comparing P200-R50 against S50 (equal training token budget) and S250 (equal total token budget) across all six pruning methods (with both 2:4 and unstructured variants for Wanda and SparseGPT) at the 50% pruning ratio. Under an equal total token budget (P200-R50 vs. S250), S250 surpasses Minitron-D and Minitron-W in average accuracy, FLAP and Sheared LLaMA keep small leads, and all sparse variants except SparseGPT-2:4 keep small leads.

Per-benchmark observations. FLAP exhibits the strongest BoolQ initialization signal in the table: $\Delta\text{BoolQ} = +16.6$ vs. S50, and it is the only benchmark/method combination where the residual advantage at S250 exceeds 8 points ($+8.7$). FLAP also holds the best-cell perplexity numbers overall (C4 13.64, WT 7.90, WT-2 7.97), ahead of every sparse method even though it is a structured method.

Table 12. Minitron depth pruning across retraining token budgets (Llama-3.1-8B → 4B, 50% pruning). *SN:* train from scratch for NB tokens. *P200-RN:* pretrain 200B tokens on DCLM, prune, retrain NB tokens. *Meta-RN:* prune from Meta's released Llama-3.1-8B, retrain NB tokens. DCLM loss is validation cross-entropy on DCLM (lower is better). Avg is the mean of WinoG, ARC-C, ARC-E, HSwag, PIQA, SciQ, BoolQ, and OBQA. Each ΔSN row reports the difference between the pruned-model row directly above (P200-RN or Meta-RN) and the SN baseline.

method	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
Minitron depth	S10	3.09	25.92	18.18	18.71	16.07	49.0	26.7	49.3	38.6	67.2	76.7	60.0	31.2	49.8
	P200-R10	2.70	16.83	10.29	10.44	10.79	57.7	34.6	68.5	60.3	75.8	89.8	60.5	38.2	60.7
	ΔS10	+0.39	+9.09	+7.89	+8.27	+5.28	+8.7	+7.9	+19.2	+21.7	+8.6	+13.1	+0.5	+7.0	+10.9
	Meta-R10	2.69	16.29	9.86	9.98	10.58	58.5	36.7	70.1	61.9	75.6	92.0	61.2	39.4	61.9
	ΔS10	+0.41	+9.63	+8.32	+8.73	+5.49	+9.5	+10.0	+20.8	+23.3	+8.4	+15.3	+1.2	+8.2	+12.1
	S30	2.81	19.40	12.50	12.75	11.91	54.3	30.4	63.7	54.2	72.3	86.1	48.8	33.8	55.5
	P200-R30	2.59	15.40	9.19	9.27	9.68	59.0	38.3	71.2	65.6	76.5	91.2	61.5	37.8	62.6
	ΔS30	+0.21	+4.00	+3.31	+3.48	+2.23	+4.7	+7.9	+7.5	+11.4	+4.2	+5.1	+12.7	+4.0	+7.1
	Meta-R30	2.59	15.07	8.96	9.04	9.64	61.7	39.8	72.8	66.8	76.7	93.0	61.2	40.6	64.1
	ΔS30	+0.22	+4.33	+3.54	+3.71	+2.27	+7.4	+9.4	+9.1	+12.6	+4.4	+6.9	+12.4	+6.8	+8.6
	S50	2.64	16.90	10.61	10.77	10.35	56.8	36.2	70.1	61.9	74.4	90.4	57.8	37.8	60.7
	P200-R50	2.54	15.34	9.30	9.41	9.46	64.4	40.7	71.5	68.0	76.8	91.7	64.0	38.0	64.4
	ΔS50	+0.10	+1.56	+1.31	+1.36	+0.89	+7.6	+4.5	+1.4	+6.1	+2.4	+1.3	+6.2	+0.2	+3.7
	Meta-R50	2.50	14.42	8.55	8.61	9.21	63.3	40.0	72.7	69.0	77.6	93.2	62.2	40.6	64.8
	ΔS50	+0.14	+2.48	+2.06	+2.16	+1.14	+6.5	+3.8	+2.6	+7.1	+3.2	+2.8	+4.4	+2.8	+4.1
	S250	2.49	14.79	8.94	9.01	9.03	65.0	41.6	73.7	70.6	77.9	93.1	65.4	42.0	66.2
	P200-R250	2.46	13.87	8.12	8.16	8.57	67.3	43.3	74.2	72.5	78.6	94.4	62.7	42.8	67.0
	ΔS250	+0.03	+0.92	+0.82	+0.85	+0.46	+2.3	+1.7	+0.5	+1.9	+0.7	+1.3	-2.7	+0.8	+0.8
	Meta-R250	2.46	13.72	8.00	8.03	8.56	66.6	44.5	75.0	72.9	78.8	94.4	68.6	43.0	68.0
	ΔS250	+0.03	+1.07	+0.94	+0.98	+0.47	+1.6	+2.9	+1.3	+2.3	+0.9	+1.3	+3.2	+1.0	+1.8
	S500	2.47	14.15	8.42	8.46	8.65	68.4	42.7	74.6	72.5	77.9	93.4	68.1	40.8	67.3
	P200-R500	2.43	13.59	7.93	7.96	8.36	66.4	45.0	74.6	74.1	79.0	94.6	68.5	42.4	68.1
	ΔS500	+0.04	+0.56	+0.49	+0.50	+0.29	-2.0	+2.3	0.0	+1.6	+1.1	+1.2	+0.4	+1.6	+0.8
	Meta-R500	2.43	13.46	7.81	7.83	8.34	68.7	45.8	75.7	74.5	80.0	95.0	69.4	44.4	69.2
	ΔS500	+0.04	+0.69	+0.61	+0.63	+0.31	+0.3	+3.1	+1.1	+2.0	+2.1	+1.6	+1.3	+3.6	+1.9

Table 13. Minitron width pruning across retraining token budgets (Llama-3.1-8B → 4B, 50% pruning). *SN:* train from scratch for NB tokens. *P200-RN:* pretrain 200B tokens on DCLM, prune, retrain NB tokens. *Meta-RN:* prune from Meta's released Llama-3.1-8B, retrain NB tokens. DCLM loss is validation cross-entropy on DCLM (lower is better). Avg is the mean of WinoG, ARC-C, ARC-E, HSwag, PIQA, SciQ, BoolQ, and OBQA. Each ΔSN row reports the difference between the pruned-model row directly above (P200-RN or Meta-RN) and the SN baseline.

method	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
Minitron width	S10	3.11	26.68	18.74	19.26	16.55	51.9	25.3	51.5	38.2	67.1	78.6	59.7	28.6	50.1
	P200-R10	2.62	15.57	9.44	9.58	9.83	65.0	37.2	69.6	68.3	76.2	93.9	66.0	39.8	64.5
	ΔS10	+0.49	+11.11	+9.30	+9.68	+6.72	+13.1	+11.9	+18.1	+30.1	+9.1	+15.3	+6.3	+11.2	+14.4
	Meta-R10	2.55	14.40	8.35	8.46	9.07	66.8	42.2	73.3	70.9	78.3	95.1	73.6	38.8	67.4
	ΔS10	+0.56	+12.28	+10.39	+10.80	+7.48	+14.9	+16.9	+21.8	+32.7	+11.2	+16.5	+13.9	+10.2	+17.3
	S30	2.80	19.19	12.32	12.55	11.79	56.4	30.3	63.2	55.1	72.5	86.4	57.3	33.6	56.9
	P200-R30	2.47	13.66	7.94	8.03	8.55	68.9	48.0	76.3	73.8	79.9	95.6	76.0	41.6	70.0
	ΔS30	+0.32	+5.53	+4.38	+4.52	+3.24	+12.5	+17.7	+13.1	+18.7	+7.4	+9.2	+18.7	+8.0	+13.1
	Meta-R30	2.47	13.46	7.73	7.80	8.45	68.7	45.2	76.3	74.2	78.7	95.3	75.8	42.2	69.6
	ΔS30	+0.33	+5.73	+4.59	+4.75	+3.34	+12.3	+14.9	+13.1	+19.1	+6.2	+8.9	+18.5	+8.6	+12.7
	S50	2.70	17.47	10.99	11.17	10.65	61.3	33.6	65.1	60.3	74.7	89.6	60.8	33.8	59.9
	P200-R50	2.47	14.60	8.93	9.01	8.92	65.4	41.8	73.4	70.5	77.4	93.8	67.4	40.4	66.3
	ΔS50	+0.23	+2.87	+2.06	+2.16	+1.73	+4.1	+8.2	+8.3	+10.2	+2.7	+4.2	+6.6	+6.6	+6.4
	Meta-R50	2.39	12.95	7.41	7.47	8.13	69.7	48.5	77.6	75.7	79.0	95.8	75.3	43.6	70.7
	ΔS50	+0.31	+4.52	+3.58	+3.70	+2.52	+8.4	+14.9	+12.5	+15.4	+4.3	+6.2	+14.5	+9.8	+10.8
	S250	2.46	14.37	8.62	8.69	8.75	65.8	46.2	75.6	72.2	77.9	91.2	68.4	42.6	67.5
	P200-R250	2.39	13.02	7.56	7.60	8.02	66.3	45.2	76.0	75.2	78.8	95.5	75.6	38.3	68.9
	ΔS250	+0.07	+1.35	+1.06	+1.09	+0.73	+0.5	-1.0	+0.4	+3.0	+0.9	+4.3	+7.2	-4.3	+1.4
	Meta-R250	2.37	12.51	7.13	7.16	7.78	70.6	48.8	78.7	77.3	79.8	96.0	78.3	45.2	71.8
	ΔS250	+0.09	+1.86	+1.49	+1.53	+0.97	+4.8	+2.6	+3.1	+5.1	+1.9	+4.8	+9.9	+2.6	+4.3
	S500	2.44	13.68	8.11	8.15	8.36	66.9	45.8	75.9	73.9	79.9	94.8	71.6	42.8	69.0
	P200-R500	2.35	12.52	7.12	7.14	7.73	66.4	48.2	77.2	77.2	79.2	95.9	78.6	38.6	70.2
	ΔS500	+0.09	+1.16	+0.99	+1.01	+0.63	-0.5	+2.4	+1.3	+3.3	-0.7	+1.1	+7.0	-4.2	+1.2
	Meta-R500	2.34	12.35	6.99	7.01	7.65	72.1	50.6	79.8	78.1	80.1	95.7	79.5	44.8	72.6
	ΔS500	+0.09	+1.33	+1.12	+1.14	+0.71	+5.2	+4.8	+3.9	+4.2	+0.2	+0.9	+7.9	+2.0	+3.6

Table 14. FLAP across retraining token budgets (Llama-3.1-8B → 4B, 50% pruning). *SN:* train from scratch for NB tokens. *P200-RN:* pretrain 200B tokens on DCLM, prune, retrain NB tokens. *Meta-RN:* prune from Meta's released Llama-3.1-8B, retrain NB tokens. DCLM loss is validation cross-entropy on DCLM (lower is better). Avg is the mean of WinoG, ARC-C, ARC-E, HSwag, PIQA, SciQ, BoolQ, and OBQA.

method	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
FLAP	S10	3.15	26.38	18.55	19.08	16.43	49.0	26.5	51.3	38.5	66.8	72.0	59.5	20.5	48.0
	P200-R10	2.62	14.75	8.40	8.53	9.51	63.7	40.8	72.8	68.0	76.7	93.0	73.0	27.2	64.4
	Δ_{S10}	+0.52	+11.63	+10.15	+10.56	+6.92	+14.7	+14.3	+21.5	+29.5	+9.8	+21.0	+13.5	+6.7	+16.4
	Meta-R10	2.58	14.40	8.15	8.28	9.25	66.0	43.0	74.5	70.5	77.8	93.8	74.5	30.0	66.3
	Δ_{S10}	+0.57	+11.98	+10.40	+10.80	+7.18	+17.0	+16.5	+23.2	+32.0	+11.0	+21.8	+15.0	+9.5	+18.3
	S30	2.85	19.36	12.47	12.72	11.94	53.9	30.9	64.3	54.7	72.1	80.7	52.7	24.9	54.3
	P200-R30	2.54	14.10	8.10	8.22	8.95	65.5	43.5	75.5	71.0	77.5	93.5	74.5	27.4	66.1
	Δ_{S30}	+0.31	+5.26	+4.37	+4.50	+2.99	+11.6	+12.6	+11.2	+16.3	+5.4	+12.8	+21.8	+2.5	+11.8
	Meta-R30	2.51	13.85	7.95	8.02	8.85	67.8	45.8	75.8	72.5	78.3	93.8	74.9	31.0	67.5
	Δ_{S30}	+0.34	+5.51	+4.52	+4.70	+3.09	+13.9	+14.9	+11.5	+17.8	+6.2	+13.1	+22.2	+6.1	+13.2
	S50	2.71	17.57	11.16	11.34	10.77	58.3	34.9	67.2	59.8	73.8	83.3	58.9	25.4	57.7
	P200-R50	2.47	13.64	7.90	7.97	8.55	65.9	43.5	75.8	72.8	76.9	93.9	75.5	27.6	66.5
	Δ_{S50}	+0.24	+3.93	+3.26	+3.37	+2.22	+7.6	+8.6	+8.6	+13.0	+3.1	+10.6	+16.6	+2.2	+8.8
	Meta-R50	2.47	13.43	7.69	7.75	8.39	69.2	46.3	76.5	74.6	78.8	93.8	75.3	31.6	68.3
	Δ_{S50}	+0.24	+4.14	+3.47	+3.59	+2.38	+10.9	+11.4	+9.3	+14.8	+5.0	+10.5	+16.4	+6.2	+10.6
	S250	2.51	14.57	8.76	8.82	8.92	66.1	43.2	74.8	71.7	77.4	91.1	66.8	29.8	65.1
	P200-R250	2.40	12.85	7.35	7.38	8.00	68.8	48.3	76.9	76.1	79.3	94.8	79.8	32.8	69.6
	Δ_{S250}	+0.11	+1.72	+1.41	+1.44	+0.92	+2.7	+5.1	+2.1	+4.4	+1.9	+3.7	+13.0	+3.0	+4.5
	Meta-R250	2.39	12.80	7.30	7.33	7.93	69.9	48.5	77.2	76.4	79.8	93.9	75.4	34.2	69.4
	Δ_{S250}	+0.12	+1.77	+1.46	+1.49	+0.99	+3.8	+5.3	+2.4	+4.7	+2.4	+2.8	+8.6	+4.4	+4.3
	S500	2.49	13.91	8.25	8.28	8.53	69.2	43.7	75.5	73.4	78.9	93.0	69.8	31.4	66.9
	P200-R500	2.36	11.89	6.40	6.43	7.42	68.8	51.5	79.7	79.3	79.8	96.4	83.3	26.8	70.7
	Δ_{S500}	+0.13	+2.02	+1.85	+1.85	+1.11	-0.5	+7.9	+4.2	+5.9	+0.9	+3.4	+13.5	-4.6	+3.8
	Meta-R500	2.35	11.75	6.28	6.31	7.35	72.3	52.0	80.3	80.0	80.4	95.8	79.5	36.5	72.1
	Δ_{S500}	+0.14	+2.16	+1.97	+1.97	+1.18	+3.1	+8.3	+4.8	+6.6	+1.5	+2.8	+9.7	+5.1	+5.2

Table 15. Sheared LLaMA across retraining token budgets (Llama-3.1-8B → 4B, 50% pruning). *SN:* train from scratch for NB tokens. *P200-RN:* pretrain 200B tokens on DCLM, prune, retrain NB tokens. *Meta-RN:* prune from Meta's released Llama-3.1-8B, retrain NB tokens. DCLM loss is validation cross-entropy on DCLM (lower is better). Avg is the mean of WinoG, ARC-C, ARC-E, HSwag, PIQA, SciQ, BoolQ, and OBQA.

method	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
Sheared LLaMA	S10	3.10	26.30	18.46	18.98	16.31	50.7	25.9	50.6	38.4	67.1	77.8	59.8	19.0	48.7
	P200-R10	2.62	15.45	9.42	9.55	9.62	63.4	39.8	71.5	67.8	75.8	91.8	64.0	29.0	62.9
	Δ_{S10}	+0.48	+10.85	+9.04	+9.43	+6.69	+12.7	+13.9	+20.9	+29.4	+8.7	+14.0	+4.2	+10.0	+14.2
	Meta-R10	2.58	14.95	8.95	9.05	9.30	65.5	42.5	73.6	70.2	77.0	93.2	67.0	31.0	65.0
	Δ_{S10}	+0.52	+11.35	+9.51	+9.93	+7.01	+14.8	+16.6	+23.0	+31.8	+9.9	+15.4	+7.2	+12.0	+16.3
	S30	2.80	19.29	12.41	12.65	11.85	55.6	30.3	63.4	54.7	72.4	86.3	53.9	23.2	55.0
	P200-R30	2.55	14.70	8.85	8.96	9.10	64.8	42.0	73.6	70.0	76.4	92.8	66.2	32.0	64.7
	Δ_{S30}	+0.25	+4.59	+3.56	+3.69	+2.75	+9.2	+11.7	+10.2	+15.3	+4.0	+6.5	+12.3	+8.8	+9.7
	Meta-R30	2.52	14.25	8.55	8.65	8.90	66.0	43.5	74.6	71.3	77.2	94.0	68.0	33.5	66.0
	Δ_{S30}	+0.28	+5.04	+3.86	+4.00	+2.95	+10.4	+13.2	+11.2	+16.6	+4.8	+7.7	+14.1	+10.3	+11.0
	S50	2.67	17.51	11.10	11.28	10.69	60.0	34.2	66.1	59.9	74.6	89.0	59.6	23.8	58.4
	P200-R50	2.50	14.35	8.55	8.63	8.90	65.8	42.8	74.7	71.2	76.7	93.6	68.0	34.2	65.9
	Δ_{S50}	+0.17	+3.16	+2.55	+2.65	+1.79	+5.8	+8.6	+8.6	+11.3	+2.1	+4.6	+8.4	+10.4	+7.5
	Meta-R50	2.47	13.95	8.25	8.32	8.70	67.0	44.6	75.5	72.5	77.8	94.3	69.0	35.0	67.0
	Δ_{S50}	+0.20	+3.56	+2.85	+2.96	+1.99	+7.0	+10.4	+9.4	+12.6	+3.2	+5.3	+9.4	+11.2	+8.6
	S250	2.47	14.57	8.77	8.85	8.89	64.8	44.2	74.7	71.6	77.9	91.9	67.1	30.4	65.3
	P200-R250	2.42	13.35	7.78	7.85	8.35	67.5	45.8	76.4	74.2	79.4	94.6	70.0	35.5	67.9
	Δ_{S250}	+0.05	+1.22	+0.99	+1.00	+0.54	+2.7	+1.6	+1.7	+2.6	+1.5	+2.7	+2.9	+5.1	+2.6
	Meta-R250	2.39	13.00	7.45	7.52	8.15	70.8	49.2	79.0	77.2	80.5	95.8	74.0	37.0	70.4
	Δ_{S250}	+0.08	+1.57	+1.32	+1.33	+0.74	+6.0	+5.0	+4.3	+5.6	+2.6	+3.9	+6.9	+6.6	+5.1
	S500	2.45	13.91	8.26	8.30	8.50	67.5	44.6	75.4	73.3	79.1	94.2	70.2	31.9	67.0
	P200-R500	2.37	12.75	7.25	7.32	8.00	69.5	47.8	78.0	75.8	79.6	95.8	73.0	37.0	69.6
	Δ_{S500}	+0.08	+1.16	+1.01	+0.98	+0.50	+2.0	+3.2	+2.6	+2.5	+0.5	+1.6	+2.8	+5.1	+2.6
	Meta-R500	2.34	12.35	6.95	7.02	7.75	72.0	51.0	80.0	78.2	79.8	96.2	76.0	38.5	71.5
	Δ_{S500}	+0.11	+1.56	+1.31	+1.28	+0.75	+4.5	+6.4	+4.6	+4.9	+0.7	+2.0	+5.8	+6.6	+4.5

Table 16. Pruning-retraining versus training from scratch at 50% pruning ratio. *P200-R50:* pretrain 200B tokens, prune, retrain 50B tokens. *S50:* train from scratch for 50B tokens (equal training token budget). *S250:* train from scratch for 250B tokens (equal total token budget). ΔP200-R50: advantage of P200-R50 over the scratch row directly above, signed so that positive means P200-R50 is better (lower loss or perplexity, higher accuracy), as in Table 11. Blue: P200-R50 better; red: P200-R50 worse; grey: accuracy difference smaller than one binomial standard error (Appendix E), or zero.

method	tokens	loss ↓	perplexity ↓				accuracy(%) ↑								
		DCLM	C4	WT	WT-2	CNN	WinoG	ARC-C	ARC-E	HSwag	PIQA	SciQ	BoolQ	OBQA	**Avg**
Minitron depth	P200-R50	2.54	15.34	9.30	9.41	9.46	64.4	40.7	71.5	68.0	76.8	91.7	64.0	38.0	64.4
	S50	2.64	16.90	10.61	10.77	10.35	56.8	36.2	70.1	61.9	74.4	90.4	57.8	37.8	60.7
	ΔP200-R50	+0.10	+1.56	+1.31	+1.36	+0.89	+7.6	+4.5	+1.4	+6.1	+2.4	+1.3	+6.2	+0.2	+3.7
	S250	2.49	14.79	8.94	9.01	9.03	65.0	41.6	73.7	70.6	77.9	93.1	65.4	42.0	66.2
	ΔP200-R50	-0.05	-0.55	-0.36	-0.40	-0.43	-0.6	-0.9	-2.2	-2.6	-1.1	-1.4	-1.4	-4.0	-1.8
Minitron width	P200-R50	2.47	14.60	8.93	9.01	8.92	65.4	41.8	73.4	70.5	77.4	93.8	67.4	40.4	66.3
	S50	2.70	17.47	10.99	11.17	10.65	61.3	33.6	65.1	60.3	74.7	89.6	60.8	33.8	59.9
	ΔP200-R50	+0.23	+2.87	+2.06	+2.16	+1.73	+4.1	+8.2	+8.3	+10.2	+2.7	+4.2	+6.6	+6.6	+6.4
	S250	2.46	14.37	8.62	8.69	8.75	65.8	46.2	75.6	72.2	77.9	91.2	68.4	42.6	67.5
	ΔP200-R50	-0.01	-0.23	-0.31	-0.32	-0.17	-0.4	-4.4	-2.2	-1.7	-0.5	+2.6	-1.0	-2.2	-1.2
FLAP	P200-R50	2.47	**13.64**	**7.90**	**7.97**	8.55	65.9	43.5	**75.8**	72.8	76.9	93.9	**75.5**	27.6	66.5
	S50	2.71	17.57	11.16	11.34	10.77	58.3	34.9	67.2	59.8	73.8	83.3	58.9	25.4	57.7
	ΔP200-R50	+0.24	+3.93	+3.26	+3.37	+2.22	+7.6	+8.6	+8.6	+13.0	+3.1	+10.6	+16.6	+2.2	+8.8
	S250	2.51	14.57	8.76	8.82	8.92	66.1	43.2	74.8	71.7	77.4	91.1	66.8	29.8	65.1
	ΔP200-R50	+0.04	+0.93	+0.86	+0.85	+0.37	-0.2	+0.3	+1.0	+1.1	-0.5	+2.8	+8.7	-2.2	+1.4
Sheared LLaMA	P200-R50	2.50	14.35	8.55	8.63	8.90	65.8	42.8	74.7	71.2	76.7	93.6	68.0	34.2	65.9
	S50	2.67	17.51	11.10	11.28	10.69	60.0	34.2	66.1	59.9	74.6	89.0	59.6	23.8	58.4
	ΔP200-R50	+0.17	+3.16	+2.55	+2.65	+1.79	+5.8	+8.6	+8.6	+11.3	+2.1	+4.6	+8.4	+10.4	+7.5
	S250	2.47	14.57	8.77	8.85	8.89	64.8	44.2	74.7	71.6	77.9	91.9	67.1	30.4	65.3
	ΔP200-R50	-0.03	+0.22	+0.22	+0.22	-0.01	+1.0	-1.4	0.0	-0.4	-1.2	+1.7	+0.9	+3.8	+0.6
Wanda 2:4	P200-R50	2.49	14.71	9.07	9.15	8.96	66.1	43.6	74.5	70.5	78.8	93.5	66.7	40.2	66.7
	S50	2.63	16.68	10.40	10.55	10.23	59.2	36.8	70.3	62.8	75.4	90.4	60.4	37.0	61.5
	ΔP200-R50	+0.14	+1.97	+1.33	+1.40	+1.27	+6.9	+6.8	+4.2	+7.7	+3.4	+3.1	+6.3	+3.2	+5.2
	S250	2.56	14.33	8.54	8.60	8.74	64.2	42.4	74.4	70.3	77.1	92.3	68.5	40.4	66.2
	ΔP200-R50	+0.07	-0.38	-0.53	-0.55	-0.22	+1.9	+1.2	+0.1	+0.2	+1.7	+1.2	-1.8	-0.2	+0.5
Wanda-U	P200-R50	**2.43**	13.94	8.31	8.37	**8.47**	67.7	**46.7**	**75.8**	**73.1**	**79.0**	**94.9**	66.7	40.8	**68.1**
	S50	2.63	16.72	10.51	10.66	10.26	60.1	36.3	69.5	62.9	75.9	90.5	60.0	37.2	61.6
	ΔP200-R50	+0.20	+2.78	+2.20	+2.29	+1.79	+7.6	+10.4	+6.3	+10.2	+3.1	+4.4	+6.7	+3.6	+6.5
	S250	2.45	14.29	8.51	8.57	8.73	64.7	43.6	75.5	72.1	78.8	94.2	67.1	41.6	67.2
	ΔP200-R50	+0.02	+0.35	+0.20	+0.20	+0.26	+3.0	+3.1	+0.3	+1.0	+0.2	+0.7	-0.4	-0.8	+0.9
SparseGPT 2:4	P200-R50	2.48	14.60	8.83	8.90	8.90	66.5	42.3	73.2	70.6	78.1	93.7	67.7	**41.6**	66.7
	S50	2.63	16.72	10.40	10.54	10.25	62.2	40.4	70.2	62.7	75.4	90.6	58.6	36.6	62.1
	ΔP200-R50	+0.15	+2.12	+1.57	+1.64	+1.35	+4.3	+1.9	+3.0	+7.9	+2.7	+3.1	+9.1	+5.0	+4.6
	S250	2.53	14.32	8.59	8.65	8.72	67.1	42.0	73.5	71.1	78.9	95.0	68.8	40.0	67.1
	ΔP200-R50	+0.05	-0.28	-0.24	-0.25	-0.18	-0.6	+0.3	-0.3	-0.5	-0.8	-1.3	-1.1	+1.6	-0.4
SparseGPT-U	P200-R50	2.44	14.06	8.39	8.44	8.56	**68.3**	45.0	75.3	72.3	78.9	94.1	65.9	40.0	67.5
	S50	2.63	16.65	10.31	10.47	10.22	60.4	36.9	69.3	62.5	75.0	90.6	62.5	38.2	61.9
	ΔP200-R50	+0.19	+2.59	+1.92	+2.03	+1.66	+7.9	+8.1	+6.0	+9.8	+3.9	+3.5	+3.4	+1.8	+5.6
	S250	2.48	14.31	8.53	8.59	8.74	67.0	43.2	73.7	71.0	78.2	93.1	62.4	41.6	66.3
	ΔP200-R50	+0.04	+0.25	+0.14	+0.15	+0.18	+1.3	+1.8	+1.6	+1.3	+0.7	+1.0	+3.5	-1.6	+1.2

OBQA splits the sparse and structured methods at S250: Sheared LLaMA keeps a $+3.8$ advantage, the largest among structured methods, while Wanda-U and SparseGPT-U trail S250. Finally, the sparse-method advantage at full data is small and benchmark-dependent rather than uniform: Wanda-2:4, Wanda-U, and SparseGPT-U post average deltas of only $+0.5$ to $+1.2$ at S250, SparseGPT-2:4 trails on average (-0.4), and individual benchmarks can favor scratch (e.g., Wanda-2:4 on BoolQ and WT-2 perplexity). The sparse advantage at S250 therefore varies by benchmark.