MODEL CARVING: OPTIMIZING THE TOPOLOGY OF STANDARD AND LOOPED TRANSFORMERS ✂

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer architectures are largely fixed: the same block repeats across layers, and the connections between them, for both standard and looped transformers, are prescribed in advance rather than learned. We ask: can a Transformer's topology, learned jointly with its weights, beat a hand-designed one on the quality-efficiency frontier? In this paper, we introduce CARVING, a method to learn a model's topology during training. CARVING places a gate on each edge of a Transformer's computation graph, trains the gates jointly with the weights, and compiles the remaining graph into a sparser, carved model. On both standard and looped Transformers, carved models beat the $1.7\times$ larger models they were carved from, reaching lower validation loss with half the inference FLOPs, one-third and one-half the KV cache, and $3.5\times$ and $2.1\times$ faster decoding throughput at the 1.3B and 2.7B scales, respectively. Carved models also beat training-FLOP-matched shallower Transformers with the same inference cost in validation quality and throughput. The architectures learned share similar features across model sizes, scales, and seeds, removing 35% of the MLP sublayers and 53% of the attention sublayers on average. These results suggest that standard, homogeneously repeated blocks are not always the most efficient design. Carved models also outperform larger models pruned post-hoc to their same inference cost on the quality-efficiency frontier, showing that the learned topology both differs from and beats standard post-hoc compression.

1 INTRODUCTION

A classical Transformer (Vaswani et al., 2017) executes a homogeneous set of blocks, in which every block contains attention and MLP matrices in a fixed order. A looped Transformer goes further and repeats pre-determined blocks of this whole stack several times. But:

Why should every layer of a Transformer look the same? Can the optimal topology be learned?

We ask whether a Transformer can learn its own topology, and whether that learned topology can beat hand-designed ones on the quality-efficiency frontier. Recent work has shown that much of a transformer's computation can be pruned while costing little to its quality. For instance, pruning methods remove the least important sublayers of a trained model, ranked by residual similarity, first-order gradient estimates, or other importance heuristics (Men et al., 2024; Gromov et al., 2025), or by learning masks over its trained weights (Xia et al., 2022; 2024). Routing keeps every layer and instead decides at inference time which ones to execute per token (Raposo et al., 2024; Li et al., 2026; Bae et al., 2025), so the executed graph changes per-input and the model is still served in full. Neural architecture search learns the wiring of one small cell and repeats it everywhere. However, we want to learn the wiring of the whole model, so every layer can differ (Liu et al., 2019; Xie et al., 2019). None of these approaches optimize the full model topology with the weights jointly during training, and most are limited to homogeneous topologies that repeat identically across depth.

We introduce CARVING, a method that learns a Transformer's topology together with its weights, deciding which sublayers to keep and how they connect to one another. CARVING starts from an *overconnected* Transformer, in which every layer can read from every earlier layer. Then, we place hard gates (Louizos et al., 2018) on each sublayer and each connection and train the gates jointly with the weights given a compute budget. The model keeps a sublayer or connection only if it reduces the

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.

loss enough to justify its cost. After search, the active gates compile into a smaller Transformer with a nonuniform computation graph, which we call the **carved model** (Figure 2). Notably, because this topology is fixed across inputs, any carved model can run on regular Transformer kernels. We test CARVING for both standard and looped Transformers, and evaluate retraining the carved models from scratch as well as a single continuous training procedure.

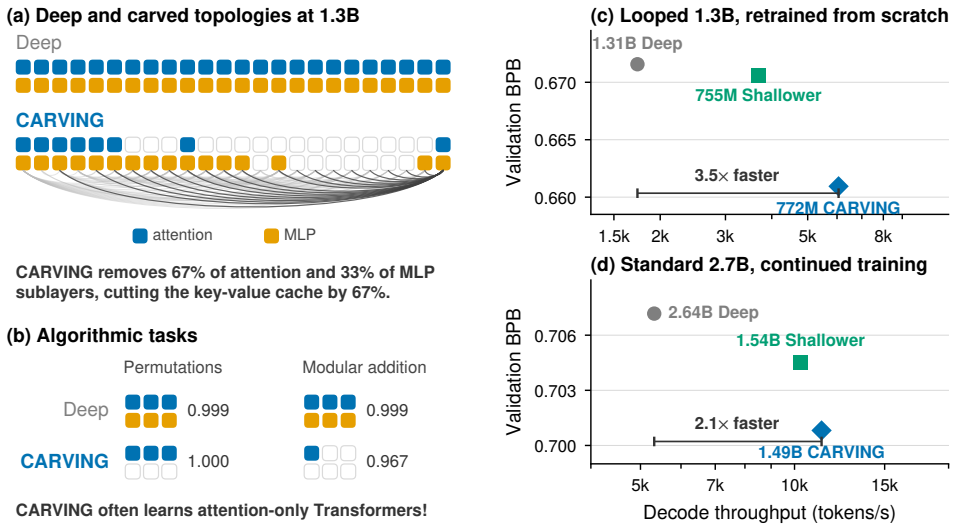


Figure 1: **A carved Transformer beats the deep model's validation performance at** 2.1 **to** $3.5\times$ **its decode throughput.** (a) The attention (blue), MLP (orange), and empty (removed) sublayers kept in each layer of a 1.3B looped Transformer before and after CARVING. The arcs are kept connections. (b) Two synthetic tasks on which the carved model learns an attention-only topology. (c, d) Validation performance against decode throughput on one H100 for the 1.3B looped and the 2.7B standard Transformers. We compare the deep model, a shallower model at the carved model's inference cost, and the carved model, each trained for the *same FLOPs*. Looped models retrain from scratch with search additional; standard models continue from search weights with search included.

CARVING produces smaller and faster Transformers that beat larger deep ones in validation quality. A 772M looped Transformer carved from a 1.3B deep model beats its validation BPB with 41% fewer parameters, half the inference FLOPs, and a third of the key-value cache per token (Figure 1, Table 1). Similarly, a 1.5B standard Transformer carved from a 2.7B deep model beats it with 44% fewer parameters and inference FLOPs, half the key-value cache, and $2.1\times$ the decode throughput. Both carved models beat a shallower Transformer of the same inference cost, and both beat the deep model pruned post-hoc to the same inference cost. Moreover, we evaluate scaling from 140M to 2.7B parameters and show that CARVING improves compute-optimal scaling as we increase fixed training compute, while requiring at most 15% of the training FLOPs to learn a settled topology (Appendix B.4). Finally, we find that the topologies themselves are similar in structure across model type and scale: every search keeps the first layer's attention, attention is kept less with depth, and the last layer receives the most incoming connections. On synthetic tasks such as modular addition, CARVING learns the cheapest sublayer subsets (such as attention-only Transformers!) that solve the task (Section 5.2).

2 CARVING

CARVING has three steps: search the gates for at most 15% of the reference training FLOPs, compile the retained topology into a Transformer, and train that Transformer.

Looped Transformers. A looped Transformer stores L layers and applies them R times, so it executes LR layer applications with the parameters of L layers (Dehghani et al., 2019). Looping unchanged layers causes small amplifications of the residual stream to compound into residual-norm

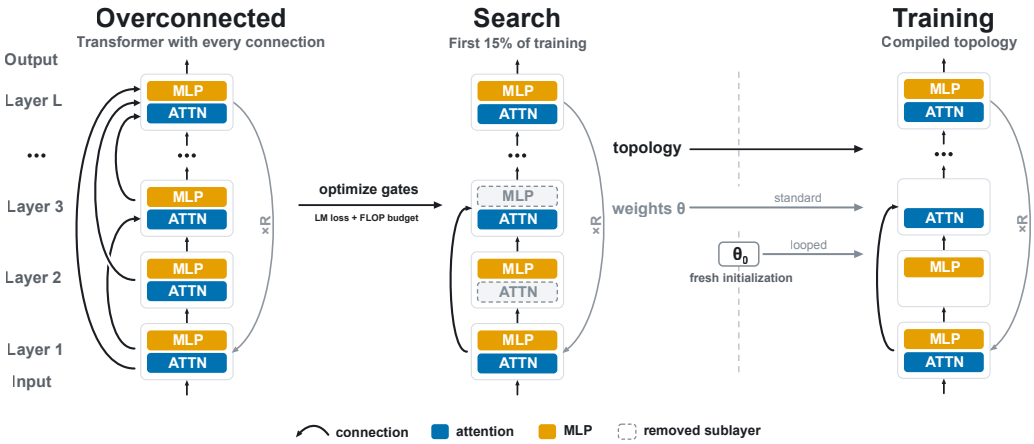


Figure 2: **Search, compile, train.** CARVING gates every attention and MLP sublayer and every connection of an overconnected Transformer (Eqs. (1) and (2)) and trains the gates with the weights under a FLOP budget (Eq. (3)). During search the gates are continuous. A closed gate removes its sublayer or connection (the dotted sublayers), and the kept ones are compiled into a smaller Transformer, which we retrain from scratch or from the search weights.

blowups, and gradients accumulated across loops often oscillate early in training (Fu et al., 2026), so recent work has focused on stabilizing the loop to release looped models at scale (Zhu et al., 2025; Geiping et al., 2025). We evaluate CARVING on the Fully Looped Transformer (FLT) of Fu et al. (2026), a more stable variant of the naive looped Transformer in which every loop restarts from the token embeddings and reads the previous loop through its attention queries (Appendix A.1).

2.1 Gated sublayers and connections

Consider a Transformer with L layers, where layer j updates the residual stream $\mathbf{h}$ with an attention sublayer and then an MLP sublayer. CARVING places a gate on each sublayer output:

$$\mathbf{h} \leftarrow \mathbf{h} + g_c F_c(\mathbf{h}), \qquad g_c \in \{0, 1\}, \tag{1}$$

where c indexes the sublayer and F_c is its attention or MLP function. Since a sublayer only adds its output to the residual stream, a zero gate removes it entirely.

CARVING also places a gate $g_{i\to j}$ on every connection from an earlier layer i to a later layer j, so the search starts from what we define as an *overconnected Transformer*. Connections that are kept let layer j read the residual stream as it was directly after layer i. Multiple connections are averaged before going into layer j (taking the mean instead of the sum allows us to keep the input of layer j at the scale of a single residual stream). We represent this as $\mathbf{h}_j^-$, that is, the input that layer j reads after its connections are applied:

$$\mathbf{h}_j^- = \frac{\max(0,\, 1 - m_j)\, \mathbf{h}_j^{\text{pre}} + \sum_{i<j} g_{i\to j}\, \mathbf{h}_i}{\max(1,\, m_j)}, \qquad m_j = \sum_{i<j} g_{i\to j}. \tag{2}$$

If no connection is kept, the layer reads $\mathbf{h}_j^{\text{pre}}$ as in a standard Transformer. Equation (2) also applies during search, when gates are continuous in $[0, 1]$. CARVING therefore learns $2L$ sublayer gates and $L(L-1)/2$ connection gates, and we call one binary setting of these gates a *topology*. In a looped Transformer, every loop shares one topology.

We also run a *sublayer-only ablation* detailed in Appendix B.2 that keeps the $2L$ sublayer gates without connection gates, so each layer reads the output of the layer below.

2.2 Learning the gates under a FLOP budget

Objective. We want the gated model to minimize the training loss subject to its expected FLOP fraction ρ meeting a budget ρ^*. We solve this with a Lagrangian, following the Lagrangian formulations

used by CoFi and Sheared LLaMA for structured pruning (Xia et al., 2022; 2024):

$$\mathcal{L}(\theta, a, \lambda) = \mathbb{E}_u\big[\ell\big(\theta, g(a,u)\big)\big] + \lambda\big(\rho(a) - \rho^*\big), \tag{3}$$

where ℓ is the next-token cross-entropy of the gated model on the training data, θ the weights, a the gate parameters, $g(a,u)$ the gates sampled with noise u (Eq. (4)), and $\rho(a)$ the expected FLOP fraction of the sampled model. Weights and gates are trained jointly from scratch, with every sublayer gate open and each layer's connection gates summing to $1/2$ (Appendix A.3).

Gates. We use the hard-concrete distribution of Louizos et al. (2018), which gives each gate a reparameterized gradient and puts a nonzero probability on exact zeros. Each gate g_c has one parameter a_c, and during search we sample

$$\tilde{g}_c = \sigma\left(\frac{a_c + \log u_c - \log(1-u_c)}{\beta}\right), \quad u_c \sim U(0,1), \qquad g_c = \mathrm{clip}_{[0,1]}\big((\zeta-\gamma)\tilde{g}_c + \gamma\big), \tag{4}$$

where σ is the logistic sigmoid, β a temperature, and $\gamma < 0 < 1 < \zeta$ stretch the sample so that clipping puts probability mass at exactly 0 and 1. The probability that a sampled gate is nonzero is

$$p_c(a_c) = \Pr(g_c > 0) = \sigma(a_c - \beta t_0), \qquad t_0 = \log(-\gamma/\zeta). \tag{5}$$

When a gate samples zero, the corresponding sublayer or connection contributes nothing to the forward pass, and likewise receives no gradient from the loss on that step. Gate parameters use their own AdamW update (Appendix A.3).

Cost. Let w_c be the training FLOPs per token of sublayer or connection c, and let B be the cost of one pass through all L layers (the sum of w_c over the attention and MLP sublayers). The expected FLOP fraction of the gated model is

$$\rho(a) = \frac{1}{B}\sum_c w_c\, p_c(a_c), \tag{6}$$

so $\rho = 1$ is one full pass through the L layers, excluding the output head. We count analytic training FLOPs following Kaplan et al. (2020) (Appendix A.6): with width d, H heads of width d_h and sequence length T, an attention sublayer costs $24d^2 + 12Hd_hT$ per token, an MLP $48d^2$, and a connection only $3d$. Since $Hd_h = d$ in our models, an MLP costs more than an attention sublayer once $d > T/2$, which holds from 370M up.

Multiplier. The multiplier λ sets how strongly the budget term in Eq. (3) pushes gates closed. Specifically, the budget term contributes $\lambda w_c/B$ to the derivative with respect to p_c, so positive λ acts as a *price* per unit of FLOP fraction. Then, the carving process keeps a sublayer or connection only while the loss reduction it gives is worth that price. After each optimizer step we update λ from the constraint error $\rho(a) - \rho^*$ with a proportional–integral rule (Sohrabi et al., 2024), so λ grows while the expected cost is above the budget and shrinks once it is below. We regulate how λ descends in two ways. The CARVING process for our looped Transformer experiments keeps $\lambda \geq 0$, so the budget acts as a ceiling and the search stops pushing once the cost is under it. However, for the standard-Transformer and synthetic carves, using $\lambda \geq 0$ caused the gates to crash (and stay closed) early. Therefore, for the standard-Transformer and synthetic tasks, we instead anneal the cost target b_t from the cost of the all-open model down to the desired budget (Appendix A.4).

2.3 Compiling and training the carved model

After search we set each gate from its deterministic value $\widehat{g}_c = \mathrm{clip}_{[0,1]}\big((\zeta-\gamma)\sigma(a_c) + \gamma\big)$ (Louizos et al., 2018), keep component c if $\widehat{g}_c > 0$ (equivalently $a_c > t_0$), and recompile the Transformer with only the retained sublayers and connections (no gates). Then, we retrain carved looped Transformers from scratch to test the topology alone. For the standard Transformers, the carved topology retrained from scratch matches the deep model at 0.76× inference FLOPs, but degrades at the more aggressive 0.56× compression used in our headline setting (see Table 7 in Appendix B.3). Therefore, retraining from scratch remains viable for carved, standard Transformers, with slightly less inference FLOPs savings compared to carved looped Transformers. For consistency, we decide to use the carved, standard Transformers *continued from their search weights* for our headline comparisons. We match training FLOPs to the deep model, counting search inside this budget for standard Transformers and separately for looped Transformers (Appendix A.5). As in any isoFLOP comparison, the carved model trains on more tokens because a carved token is cheaper.

3 EXPERIMENTAL SETUP

Language models. We train Fully Looped Transformers at 140M, 370M, 770M, and 1.3B parameters, with 16 to 24 layers looped four times, and standard Transformers with 24 layers at 1.3B and 32 layers at 2.7B (Appendix A.2). Since every token passes through the stored layers four times, the 1.3B looped Transformer costs about twice the inference FLOPs per token of the 2.7B standard one. We train all models on a deduplicated FineWeb-Edu corpus (Penedo et al., 2024) with sequence length 2,048, vocabulary size 49,152, and 524,288 tokens per optimizer step. Our headline looped models are trained for approximately 100 tokens per parameter. The deep standard Transformers are trained for approximately 40 tokens per parameter. The looped topology-search cost is treated separately as described in Section 2.3.

Evaluation. For evaluation, we use validation bits per byte (comparable across tokenizers) on one held-out validation set (Appendix C.1). We say two models *match* when their BPB differ by less than 0.003 which we determined from seed variance (Appendix B.1). We also report token perplexity (PPL) on the same set; CORE-22, the 22 CORE downstream tasks of DCLM (Li et al., 2024); the accuracy of greedy generation on DCLM's nine language-modeling tasks (Gen-9); and pass@k on the same nine tasks at equal inference FLOPs. To evaluate throughput, we measure greedy decode throughput on one 80 GB H100 with a 1,024-token context, at the largest tested batch that fits in memory (Appendix B.5). See Appendix B.1 for language-model runs and Appendix E for synthetic-task runs.

Baselines. We match training FLOPs using the search-accounting protocol in Section 2.3; Appendix B reports deviations. The *deep* model is the full-depth Transformer that the search starts from. To test whether CARVING's gains come only from training a smaller model on more tokens, we compare each carved model with a *shallower* model: the same family and width, with the fewest layers whose inference FLOPs meet or exceed the carved model's. For instance, the 772M model carved from the 1.3B deep model is compared against a 755M shallower model with 13 of the 24 layers (Table 3). To test whether the carved topology can be found after training instead, we compare output-ablation (Kim et al., 2024), Taylor (Ma et al., 2023), and random pruning orders. At the carved model's target inference cost, we retrain the candidate with the lowest pre-retraining validation BPB from the deep weights for the FLOPs of the search (Appendix B.7).

4 RESULTS

4.1 CARVED MODELS REACH LOWER VALIDATION LOSS THAN 1.7× LARGER MODELS AT HALF THE INFERENCE COST

CARVING ends with the lowest validation loss in both families. At 1.3B, the looped model carved from the deep model and retrained from scratch ends 0.011 BPB below it, with 41% fewer parameters, half the inference FLOPs and a third of the key-value cache per token (Table 1, Figure 1c). The 13-layer shallower model, trained on about as many tokens, ends 0.010 above the carved model. The same holds at 2.7B, where the carved standard Transformer ends 0.006 below the deep model and 0.0037 below the shallower model, with 44% fewer inference FLOPs and half the key-value cache (Figure 1d).

The learned topology alone matches the deep model from a fresh initialization in both families. Looped models retrain the found carved topologies from scratch, and the carved 1.3B standard Transformer at 0.76× inference FLOPs is within 0.0012 BPB of deep (Appendices B.1, B.3). At 0.56×, search-weight continuation beats the shallower model where fresh weights tie it as the carving search process provides useful weights within the standard Transformer's training budget.

At equal inference FLOPs, CARVING solves more generative problems. A carved token is cheaper to decode, so for the inference FLOPs the deep model spends on k samples, the carved model draws more. At 2.7B, this raises pass@1 on the nine generative CORE tasks by 6.2 points over the deep model and 2.9 over the shallower model, and CARVING leads both at pass@8 and pass@16. At 1.3B it leads deep by 4.1 points at pass@1 and both baselines on GSM8K at every k (Table 13).

Table 1: **At matched retraining FLOPs, CARVING reaches the lowest validation BPB at about half the deep model's inference cost, and decodes faster than the deep and shallower models.** Within each block, the deep, shallower and carved models get retrained for the same FLOPs (for the carved looped models, topology-search compute is additional). The pruned-and-retrained rows prune the deep model to the carved model's inference FLOPs and retrain it so that pruning plus retraining uses the carved model's total training FLOPs (details in Appendix B.7). For pruned-and-retrained rows, Tokens includes the deep model's pretraining tokens plus post-pruning retraining tokens. KV is the key-value cache per token, and decode throughput is greedy decoding at a 1,024-token context and the largest batch that fits on one H100. pass@k is on the nine generative CORE tasks at equal inference FLOPs. The blue row divides the carved model's value by the deep model's. Evaluations are reported as the mean of three seeds. Full evaluations are in Tables 13 and 14 (Appendix C.2).

Model	Params (M)	Tokens (B)	GFLOPs /token	KV (KiB/tok)	Largest batch	Decode tok/s	Val. BPB	CORE-22	Gen-9	pass@k k=1	pass@k k=8	pass@k k=16
1.3*B looped Transformer*												
Deep (24 layers)	1,309	119.7	11.48	768	72	1,736	0.6716	**47.24**	**42.7**	44.0	67.6	72.9
Shallower (13 layers)	755	217.7	6.31	416	128	3,686	0.6706	44.21	39.8	**48.7**	**70.2**	**74.7**
Pruned and retrained	772	132.2	6.24	320	176	4,851	0.8492	35.82	31.10	–	–	–
CARVING	772	224.9	**6.11**	**256**	**208**	**6,058**	**0.6609**	44.59	40.3	48.1	69.5	73.6
Ratio to deep	0.59×		0.53×	0.33×	2.9×	3.5×						
2.7*B standard Transformer*												
Deep (32 layers)	2,642	105.7	5.96	320	144	5,319	0.7072	**45.44**	**40.9**	43.4	65.8	71.0
Shallower (18 layers)	1,541	181.9	3.46	180	288	10,275	0.7045	42.83	38.6	46.7	66.1	70.3
Pruned and retrained	1,542	109.2	3.38	**140**	**320**	**12,741**	0.8483	34.48	30.0	–	–	–
CARVING	1,489	188.7	**3.31**	**160**	**320**	11,294	**0.7008**	42.54	40.3	**49.6**	**68.9**	**72.8**
Ratio to deep	0.56×		0.56×	0.50×	2.2×	2.1×						

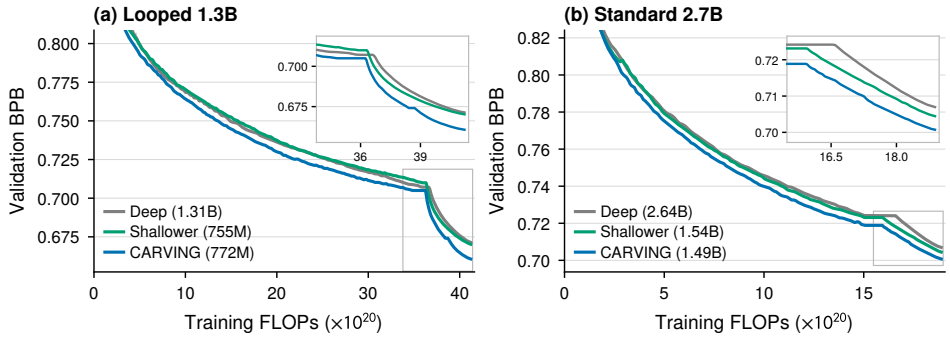


Figure 3: **The carved model's validation BPB stays below the deep model's for the whole run.** Validation bits per byte against cumulative training FLOPs for (a) the 1.3B looped Transformer, retrained from scratch, and (b) the 2.7B standard Transformer, trained from its search weights. In (a) the deep model is below the shallower model from about 2.6×10^{21} FLOPs until the cooldown, but never below CARVING.

The deep model keeps an edge on single-answer benchmarks. When each problem gets one answer, the deep model leads CARVING by 2.6 and 2.9 points on CORE-22 and by 2.4 and 0.6 points on Gen-9 at 1.3B and 2.7B. CARVING is ahead of the shallower model on both at 1.3B, and at 2.7B is within 0.3 points on CORE-22 while leading it by 1.7 points on Gen-9.

The carved model beats the deeper model for the whole run. Figure 3 plots the deep, shallower, and carved models against cumulative training FLOPs. Both cheaper models start below the deep model because at equal FLOPs they have seen more tokens. At 1.3B the shallower model falls behind the deep model from about 2.6×10^{21} FLOPs until the cooldown, and at 2.7B it ends 0.0026 BPB below the deep model. The carved model stays below the deep model throughout the entire run.

4.2 Post-hoc pruning does not recover the carved model

Pruning and retraining the deep model does worse than the carved model (Figure 9). We prune the trained deep model to the carved model's target inference cost using output-ablation, Taylor, or

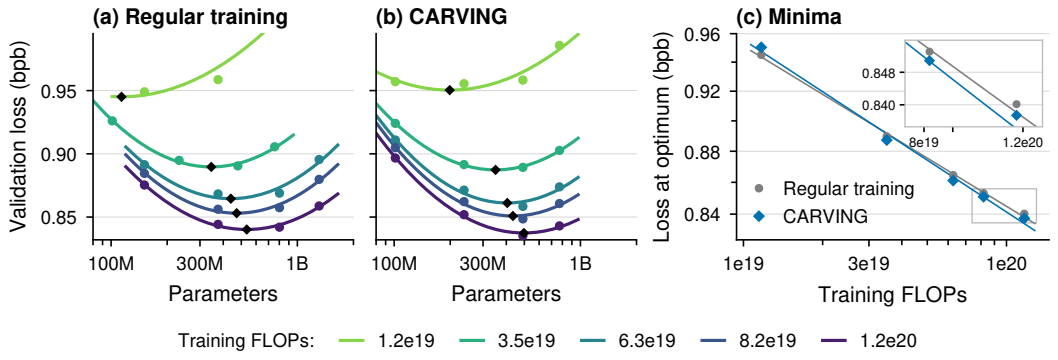


Figure 4: **From 3.5×10^{19} training FLOPs up, CARVING's compute-optimal model reaches lower loss than regular training's.** (a, b) Validation BPB of looped Transformers against parameters at five training budgets, with a parabola in log parameters fitted to each budget. We use diamonds to mark each parabola's minimum. Here, "regular training" means the training of the deeper baseline. (c) The minima against training FLOPs, with a power law fitted to each.

random ordering, and retrain the candidate with the lowest pre-retraining validation BPB. Retrained using the same total training compute, the pruned model ends 0.19 and 0.15 BPB above the carved model on the fixed validation window at 1.3B and 2.7B, and 8.1 points below it on CORE-22 at 2.7B (Table 1, Appendix B.7). The pruned model also keeps different sublayers, as the set found by pruning overlaps the search's with Jaccard 0.35 to 0.36, comparable to random masks at 0.36 to 0.41 (Appendix B.6). Therefore, pruning via importance scores after training does not find the topology that CARVING finds during training.

4.3 Scaling CARVING

From 3.5×10^{19} training FLOPs up, CARVING's compute-optimal model reaches lower loss than regular training's. Following Hoffmann et al. (2022), we train each looped recipe at four sizes per training budget, fit a parabola in log parameters to find its compute-optimal size, and define regular training as the training of the deeper baseline (Figure 4). At 3.5×10^{19} FLOPs, CARVING's optimum reaches 0.8873 BPB against 0.8896 for regular training, at 2.71 inference GFLOPs per token against 2.85, and it stays lower at every larger budget, by 0.0032 BPB at 1.2×10^{20} FLOPs. Only at the smallest budget, 1.2×10^{19} FLOPs, is regular training ahead.

Across runs up to 4.1×10^{21} FLOPs, CARVING trails deep only at the smallest budgets and wins in every looped run above about 7×10^{20} FLOPs. At our largest budgets, measured BPB improves by 1.6% for looped and 0.9% for standard Transformers. The lead grows with budget: 0.0009 to 0.011 BPB at 1.3B looped (20 to 91 tokens per parameter), and 0.0025 to 0.0064 at 2.7B standard (10 to 40; Tables 5, 6, 1).

4.4 Carved models decode fastest

Carved models decode significantly faster at every size. On one H100 at a 1,024-token context, the carved looped Transformers decode 2.7× to 4.3× faster than their dense parents and the carved standard Transformers 1.7× to 2.1×, against 1.9× to 2.2× and 1.3× to 1.9× for the shallower dense models of the same inference cost (Figure 8b, Appendix B.5). This gain persists at longer contexts as well.

CARVING complements cache quantization. At batch 64 one decode step of the dense 1.3B looped Transformer reads 52 GB of key-value cache and 2.6 GB of weights. The carved model keeps 8 of the 24 attention sublayers and therefore stores a *third of the cache per token* (half for the 2.7B standard Transformer). Thus, the freed memory holds a three times larger batch (Figure 8a), over which the weight reads are amortized. An int8 cache halves the bytes again, and it about doubles the largest batch every model fits without changing BPB. The carved 1.3B model decodes at 9,466 tokens/s, 5.5× the dense model's bf16 throughput and 3.4× its int8 throughput (Appendix B.5).

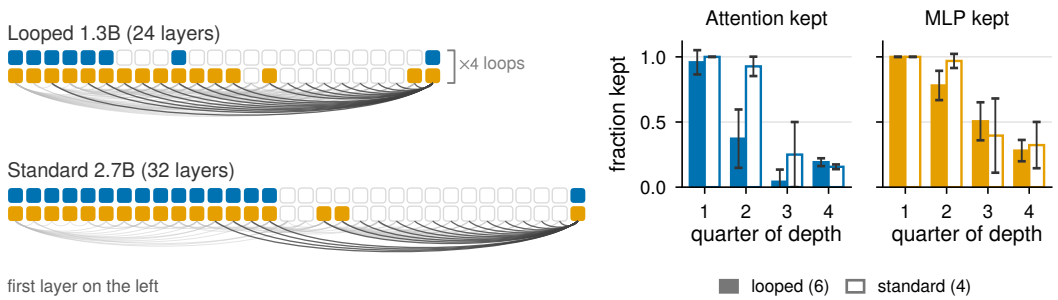


Figure 5: **The CARVING searches share similar features.** Left: attention (blue), MLP (orange) sublayers kept by the 1.3B looped search, whose 24 layers are applied four times, and by the 2.7B standard search. The arcs are the kept connections. Right: fraction of attention and MLP sublayers kept in each quarter of depth, mean and standard deviation of carved models across all scales.

5 Analysis

5.1 Learned topologies share one structure

CARVING consistently learns similar structural features. Regardless of scale, family, or seed, all carved models keep the first layer's attention, the last layer's MLP, and the last layer receives the most connections (Figure 5). Moreover, the retention of both sublayers decreases with depth where every carved model keeps nearly all of its layers in the first quarter, and on average a fifth of the attention and under a third of the MLPs in the last quarter. However, the searches do differ in how much attention they keep. Carved standard Transformers keep $58 \pm 8\%$ of their attention sublayers and keep attention through the first half of depth, while carved looped Transformers keep $39 \pm 6\%$, mostly in the first quarter and in the last layer. The spread across searches is largest in the middle half of depth. This consistency suggests that one topology might be transferrable across sizes without a new search. Appendix D gives every topology. Without the connection gates, the standard-Transformer searches save fewer inference FLOPs (19% and 38% instead of 24% and 44% at 1.3B and 2.7B) and end 0.0021 and 0.0040 BPB above the deep model (Figure 7, Appendix B.2).

Interestingly, cost does not solely explain the preference for MLPs. From 370M up an MLP "costs" more than an attention sublayer (Appendix A.6), yet CARVING consistently removes more attention. This therefore suggests that the preference comes from the sublayers' contribution to reducing loss. This preference is also what gives carved models their smaller key-value cache.

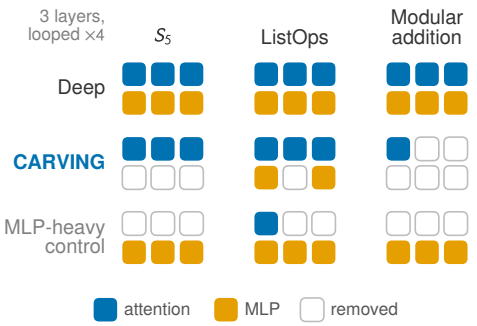
5.2 CARVING finds minimal circuits on symbolic tasks

Learning attention-only architectures (Figure 6). We carve a three-layer looped Transformer, trained on three synthetic tasks: composition of permutations of five elements, S_5 (Liu et al., 2023); ListOps, nested prefix expressions over digits (Nangia & Bowman, 2018); and addition modulo 113 (Appendix E).

Modular addition is small enough to solve exhaustively, where we retrain all sublayer subsets and identify the cheapest that achieves at least 0.95 mean validation accuracy. The cheapest keeps a single attention sublayer, and all three carved models retain this single attention sublayer. On S_5, CARVING can also learn an attention-only topology that matches the deep model's accuracy. Finally, on ListOps, CARVING learns a topology with three attention sublayers and two MLPs that achieves higher mean accuracy than the deep model at lower cost.

6 Related work

Pruning with learned masks. Similar to CARVING, CoFi and Sheared LLaMA prune a trained Transformer with hard-concrete masks (Louizos et al., 2018) which they place on heads, MLP and hidden dimensions, and whole sublayers or layers, under a Lagrangian constraint toward a target shape (Xia et al., 2022; 2024). We use the same gates and a Lagrangian, but learn them from



Task	Model	Exact match accuracy	Relative FLOPs
S_5	Deep	0.999	1.00
	CARVING	**1.000**	**0.35**
	MLP-heavy	0.018	0.65
ListOps	Deep	0.808	1.00
	CARVING	**0.832**	0.88
	MLP-heavy	0.709	**0.57**
Modular addition	Deep	**0.999**	1.00
	CARVING	0.967	**0.16**
	MLP-heavy	0.008	0.68

Figure 6: **CARVING keeps attention only when needed!** Left: retained attention (blue) and MLP (orange) sublayers for the deep, carved, and MLP-heavy models (three layers, four loops). Right: exact match accuracy and inference FLOPs per token relative to deep. Scores average three seeds, except our single-seed MLP controls. The S_5 topology is from seed 1 (Table 17). On S_5 and modular addition, CARVING learns attention-only models, while MLP-only controls remain near chance.

initialization during pretraining and constrain expected FLOPs, instead of thinning out a model into a desired shape.

Differentiable architecture search. DARTS replaces each edge of a small cell with a softmax-weighted sum of candidate operations, optimizes these mixture weights on validation loss, then keeps the top operation per edge. The resulting cell is stacked into the "found" architecture (Liu et al., 2019). SNAS, on the other hand, samples one operation per edge from a concrete relaxation (Xie et al., 2019). CARVING jointly selects sublayers and inter-layer connections across the whole model during pretraining, under a FLOP budget. A gate sampled at zero removes its component during search, whereas DARTS executes every candidate.

CARVING does not just find lottery tickets. The lottery ticket hypothesis claims that a trained dense network contains a sparse subnetwork that reaches the full network's accuracy when trained from the same initialization (Frankle & Carbin, 2019), or, at scale, from weights rewound to an early point in training (Frankle et al., 2020). CARVING, however, differs in what it removes and that it is robust across initializations. First, a lottery ticket masks individual weights, so every matrix keeps its shape and dense kernels run it no faster, whereas CARVING removes whole sublayers and connections. Second, the learned topology retrained from a fresh initialization still matches the deep model in both families (Appendices B.1, B.3).

7 Limitations and Conclusion

Limitations. The main limitation is that CARVING's lower loss does not yet translate into proportionally higher accuracy. The carved models reach lower validation BPB than the deep models they are carved from, yet trail them by 2.8 points on CORE-22 and by 1.5 points on Gen-9 on average across scales. We do not yet know why validation loss does not translate into these downstream gains. Mechanistically, the carved models keep fewer attention sublayers, and few-shot tasks may rely on attention more than the average next-token loss does. Finding which removed components carry this accuracy, and whether the search objective should price them differently, is next most important.

Conclusion. In this work, we showed that through CARVING, a Transformer's topology can be learned from the training loss together with its weights, and that this search takes only the first 15% of training. CARVING finds a smaller and faster model that reaches lower loss than the deep model it was carved from, at half the inference FLOPs and 3.5× the decode throughput. These topologies are also not arbitrary, as across scales, Transformer types, and seeds, all carved models keep attention early and MLPs deeper, a structure that post-hoc pruning cannot recover. Overall, our results suggest that the homogeneous Transformer, with one block repeated through depth, is not the optimal design, and that a Transformer's topology is a choice worth learning.

REPRODUCIBILITY AND AI DISCLOSURE STATEMENT

Reproducibility. Section 2 and Appendix A specify the CARVING search specs, gate distribution, controller update order, what compilation retains, and how we matched training compute. Section 3 specifies the data, model types, training recipes, comparison units, and metrics.

LLM assistance. We used large language models to help iterate, run, and test our experiment code, and to critique drafts of this paper. The research questions, methods, experiments, and conclusions are our own.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron C. Courville, and SeYoung Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2507.10524.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1803.03635.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/1912.05671.

Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. Simply stabilizing the loop via fully looped transformer. *arXiv preprint arXiv:2605.18797*, 2026. URL https://arxiv.org/abs/2605.18797.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.05171.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.17887.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

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, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2203.15556.

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. URL https://arxiv.org/abs/2001.08361.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. Shortened LLaMA: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 2024. URL https://arxiv.org/abs/2402.02834v1.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.11794.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? learning program-of-layers in LLMs. *arXiv preprint arXiv:2606.06574*, 2026. URL https://arxiv.org/abs/2606.06574.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.10749.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1806.09055.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1712.01312.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.11627.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024. URL https://arxiv.org/abs/2403.03853.

MosaicML NLP Team. LLM Foundry. https://github.com/mosaicml/llm-foundry, 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 *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2407.14679.

Nikita Nangia and Samuel R. Bowman. ListOps: A diagnostic dataset for latent tree learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 92–99. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-4013. URL https://aclanthology.org/N18-4013/.

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 *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.17557.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026. URL https://arxiv.org/abs/2604.12946.

David Raposo, Sam Ritter, Blake A. Richards, Timothy Lillicrap, Peter C. Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024. URL https://arxiv.org/abs/2404.02258.

Motahareh Sohrabi, Juan Ramirez, Tianyue H. Zhang, Simon Lacoste-Julien, and Jose Gallego-Posada. On PI controllers for updating lagrange multipliers in constrained optimization. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2406.04558.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03762.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022. URL https://arxiv.org/abs/2204.00408.

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

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1812.09926.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.

A CARVING IMPLEMENTATION DETAILS

A.1 FULLY LOOPED TRANSFORMER

Let d be the model width and d_h the attention-head width. We use $\mathbf{e} = \mathrm{RMS}(\mathrm{Embed}(x))$ for the normalized token embeddings, and $\mathbf{h}_{r,j}$ for the residual stream after layer j in repeat r, so layer j reads $\mathbf{h}_{r,j-1}$. Every repeat starts from the same embeddings, $\mathbf{h}_{r,0} = \mathbf{e}$. The normalized output of repeat r supplies the query source for repeat $r+1$:

$$\mathbf{s}_r = \mathrm{RMS}(\mathbf{h}_{r,L}), \tag{7}$$

In every repeat after the first, all L layers use queries from the previous repeat's output $\mathbf{s}_{r-1}$, while using the keys and values from the current residual stream, that is:

$$\begin{aligned} \mathbf{q}_{r,j} &= W_j^Q \mathbf{s}_{r-1}, \\ \mathbf{k}_{r,j} &= W_j^K \,\mathrm{RMS}(\mathbf{h}^-_{r,j}), \\ \mathbf{v}_{r,j} &= W_j^V \,\mathrm{RMS}(\mathbf{h}^-_{r,j}). \end{aligned} \tag{8}$$

Note: in the first repeat, the query source in Eq. (8) is $\mathrm{RMS}(\mathbf{h}^-_{1,j})$. After projection, queries and keys receive rotary position embeddings and per-head RMS normalization. Values are unnormalized.

Let g_j^A and g_j^M be the gates on the attention and MLP sublayers of layer j. The gated layer is

$$\begin{aligned} \widetilde{\mathbf{h}}_{r,j} &= \mathbf{h}^-_{r,j} + g_j^A \,\mathrm{Attn}_j\big(\mathrm{RMS}(\mathbf{h}^-_{r,j}); \mathbf{s}_{r-1}\big)\,, \\ \mathbf{h}_{r,j} &= \widetilde{\mathbf{h}}_{r,j} + g_j^M \,\mathrm{MLP}_j\Big(\mathrm{RMS}(\widetilde{\mathbf{h}}_{r,j})\Big)\,, \end{aligned} \tag{9}$$

During training, we unroll all R repeats in one forward pass and backpropagate through every repeat, with the loss being applied to the final repeat's output only. The output head projects the final state through the tied embedding table and applies a soft cap with

$$\mathrm{logits} = 15 \tanh\left(\frac{W_{\mathrm{emb}} \mathbf{s}_R}{15}\right). \tag{10}$$

Precision. Parameters and the residual stream are stored in fp32, and the attention and MLP matrix multiplications use bf16 activations.

A.2 Language-model configurations

All language models use rotary position embeddings and no dropout. The FLT uses bias-free $4d$ MLPs with a squared-ReLU activation. The standard Transformer uses $4d$ GELU MLPs with biases and a learned final normalization.

Table 2: Language-model shapes.

Model	d	L	R	Heads	d_h
FLT 140M	768	16	4	12	64
FLT 370M	1,152	20	4	18	64
FLT 770M	1,728	20	4	27	64
FLT 1.3B	2,048	24	4	32	64
Standard 1.3B	2,048	24	1	32	64
Standard 2.7B	2,560	32	1	32	80

Each shallower baseline (Table 3) keeps the smallest number of whole layers whose inference FLOPs meet or exceed those of the paired carved model. Otherwise, every other shape field matches the deep model at that size.

Table 3: Shallower baselines. Inference FLOPs are normalized by the deep model at the same size.

Model	Paired topology	layers	Params (M)	Inference FLOPs
FLT 140M	Ablation	9	101	$0.5864\times$
FLT 370M	CARVING / ablation	11	232	$0.5649\times$
FLT 770M	CARVING / ablation	11	479	$0.5609\times$
FLT 1.3B	CARVING / ablation	13	755	$0.5497\times$
Standard 1.3B	CARVING, $0.56\times$	13	755	$0.5722\times$
Standard 1.3B	CARVING	18	1,007	$0.7667\times$
Standard 1.3B	Ablation	20	1,107	$0.8444\times$
Standard 2.7B	CARVING	18	1,541	$0.5810\times$

Training recipe. Our optimizer and data choices build on Nanochat and Parcae (Prairie et al., 2026), and we validated our borrowed recipe at 770M before applying it at other scales. We use the SmolLM-360M tokenizer to tokenize our corpus. Optimization settings: layer matrices use Muon with learning rate 0.02, momentum 0.95, five orthogonalization steps, and second-moment coefficient 0.9. The input embedding uses AdamW with learning rate 0.003, and in the standard Transformer, the output embedding uses 0.004 and the scalar parameters 0.003. All AdamW groups use $(\beta_1, \beta_2) = (0.8, 0.95)$, $\epsilon = 10^{-10}$, and no weight decay. The gradient norm of the model parameters is clipped to 1.0. We use a constant learning rate for 80% of training and then a linear decay to zero for the last 20%.

A.3 Hard-concrete gates and initialization

For the gates of Eqs. (4)–(5) we use $\beta = 2/3$, $\gamma = -0.1$, and $\zeta = 1.1$, so $t_0 = -\log 11$, with the uniform samples clipped to $[10^{-6}, 1 - 10^{-6}]$ before taking logarithms.

After every optimizer step we also clip the gate parameters to $[a_{\text{lo}}, \log 100]$. See Appendix A.4 for the lower bound $a_{\text{lo}} < t_0$ derivation.

We initialize the attention and MLP gates at $a_c = 2.5$. Layer $j \geq 2$ has $j - 1$ possible connections from earlier layers, and each connection gate is initialized so that

$$\widehat{g}_{i \to j} = \frac{0.5}{j - 1}. \tag{11}$$

The gate parameters use a separate AdamW optimizer with learning rate 0.02, no weight decay, $(\beta_1, \beta_2) = (0.8, 0.95)$, and $\epsilon = 10^{-10}$.

Within each layer, our forward pass sequence is: the incoming connections form the input by Eq. (2), then the gated attention update is applied, then the gated MLP update.

A.4 MULTIPLIER CONTROLLER

Our inequality constraint uses Eq. (3) with $\lambda \geq 0$. The equality constraint tracks $\rho(a) - b_t = 0$ with $\lambda \in \mathbb{R}$, so if the multiplier is negative, it increases the gate values when the expected cost is below the target. Because the term $-\lambda b_t$ has no parameter gradient, our loss being differentiated is $\ell + \lambda\rho(a)$, where ℓ is the language-modeling loss. At each optimizer update, the model and gate parameters do their step first, the gate parameters are clipped, and the controller then reads its error from the clipped gate probabilities. Therefore, gate sampling adds no noise to it.

Lower gate bound and controller target. Let $\Gamma = \sum_c w_c/B$ be the total cost of every gated sublayer and connection divided by B, and let $p_{\text{hi}} = p_c(\log 100)$ be the open probability at the upper clip. We choose the lower bound so that, with every gate at the bound, the expected cost equals the cost of the cheaper of one attention or one MLP sublayer:

$$\begin{aligned} p^\star_{\text{lo}} &= \frac{\min(w_{\text{attn}}, w_{\text{mlp}})}{B\,\Gamma}, \\ a_{\text{lo}} &= \text{round}_6(\text{logit}(p^\star_{\text{lo}}) + \beta t_0)\,, \end{aligned} \tag{12}$$

where round_6 rounds to six decimal places. Write $p_{\text{lo}} = p_c(a_{\text{lo}})$ for the open probability at the lower bound. Because no gate can be fully closed or open, the budget ρ^* maps to the controller target

$$b = \text{round}_6(p_{\text{lo}}\,\Gamma + \rho^*\,(p_{\text{hi}} - p_{\text{lo}}))\,, \tag{13}$$

with $p_{\text{lo}}\Gamma$ being the smallest expected cost and $\rho^*(p_{\text{hi}} - p_{\text{lo}})$ the budgeted increment above it. Table 4 lists each of our targets. Every search uses these derived values.

Target schedule and multiplier update. We use the νPI rule of Sohrabi et al. (2024) with its smoothing coefficient set to zero. Under the equality constraint, the target is annealed from a starting value b_0 to b over S_{sched} controller updates, either linearly ($q = 1$) or cubically ($q = 3$):

$$b_t = b + (b_0 - b)\left(1 - \min\left(\frac{t}{S_{\text{sched}}}, 1\right)\right)^q. \tag{14}$$

Under the inequality constraint the target is fixed at b. With an error of $e_n = \rho(a_{n+1}) - b_t$, the multiplier update is

$$\lambda_{n+1} = \Pi(\lambda_n + k_i\, e_n\, \mathbf{1}[|e_n| > \delta] + k_p\,(e_n - e_{n-1}))\,, \tag{15}$$

with $e_{-1} = e_0$, $\lambda_0 = 0$, and gains $(k_i, k_p) = (0.06, 6.0)$. The projection Π is $\max(0, \cdot)$ under the inequality constraint and $\max(-2.913, \cdot)$ under the equality constraint.

Table 4: Controller settings for the language-model searches. Arrows show the initial and final controller targets; parentheses give the number of annealing updates. The headline looped topology uses 15% of the reference training FLOPs. The separate FLT validation compares 15% and 100% checkpoints under annealed equality.

Experiment	Constraint	CARVING target	Ablation target
FLT, all sizes	Inequality, fixed	140M: 0.541390; 370M: 0.539241; 770M: 0.538240; 1.3B: 0.536210	140M: 0.55; 370M: 0.539237; 770M: 0.538238; 1.3B: 0.536209
FLT 1.3B, validation	Equality, linear	0.983872 → 0.536210 (1,195)	0.983675 → 0.536209 (1,194)
Standard 1.3B	Equality, linear	0.983872 → 0.752954 (616)	0.983675 → 0.801957 (485)
Standard 2.7B	Equality, linear	0.983894 → 0.554682 (1,145)	0.983675 → 0.603936 (1,013)

A.5 Compiling, FLOP matching, and search cost

For a looped model with R repeats, the deep model costs

$$C_{\text{deep}} = RB + c_{\text{read}} \tag{16}$$

training FLOPs per token, where c_{read} is the cost of the output head, and the carved model costs $C_{\text{compiled}} = RB\rho_{\text{real}} + c_{\text{read}}$. To match training FLOPs then, the carved model retrained from scratch trains on

$$N^*_{\text{carve}} = \frac{N_{\text{ref}}\, C_{\text{deep}}}{C_{\text{compiled}}} \tag{17}$$

tokens, where N_{ref} is the deep reference's token count. Fixed-target looped searches stop after spending 15% of the reference training FLOPs. Total looped compute includes this search cost in addition to the matched retraining budget. Standard searches stop within the same 15% ceiling; at 1.3B, search uses 7–10% of the paired deep budget (Table 6).

A.6 FLOP accounting

Let H be the number of attention heads, T the sequence length, and V the vocabulary size. To price the sublayers and connections, we follow Kaplan et al. (2020) to count two FLOPs for each forward multiply-add and four more for its backward pass. The four attention projections require $4d^2$ multiply-adds per token and therefore cost $24d^2$ training FLOPs. The attention scores and their products with values cost $12Hd_hT$ on the same basis, and the two-matrix, $4d$-hidden MLP costs $48d^2$. Each retained connection is charged a fixed $3d$ (one d-wide accumulation in the forward pass and two pass-throughs in the backward pass). The output head is charged $c_{\text{read}} = 6Vd$. Therefore, the sublayer and connection costs are

$$\begin{aligned} w_{\text{attn}} &= 24d^2 + 12Hd_hT, \\ w_{\text{mlp}} &= 48d^2, \qquad w_{\text{conn}} = 3d. \end{aligned} \tag{18}$$

Inference uses the same: $w^{\text{infer}}_{\text{attn}} = 8d^2 + 4Hd_hT$, $w^{\text{infer}}_{\text{mlp}} = 16d^2$, $w^{\text{infer}}_{\text{conn}} = d$, and $c^{\text{infer}}_{\text{read}} = 2Vd$.

Let n_{attn}, n_{mlp}, and n_{conn} be the numbers of retained attention sublayers, MLP sublayers, and connections. During search, Eq. (6) weights each cost by the probability that its gate is open. After compiling, the realized FLOP fraction is

$$\begin{aligned} C_{\text{retained}} &= n_{\text{attn}}w_{\text{attn}} + n_{\text{mlp}}w_{\text{mlp}} + n_{\text{conn}}w_{\text{conn}}, \\ \rho_{\text{real}} &= C_{\text{retained}}/B, \end{aligned} \tag{19}$$

A full pass has $\rho_{\text{real}} = 1$; CARVING can exceed one because connections add cost. With $Hd_h = d$ and $T = 2{,}048$, the MLP's share of a layer's training FLOPs, $w_{\text{mlp}}/(w_{\text{attn}} + w_{\text{mlp}})$, is 46% at 140M, 51% at 370M, 56% at 770M, 57% at 1.3B, and 59% at 2.7B, so attention costs more than an MLP only at 140M. Replacing the training costs with the forward costs gives the carved model's inference cost, $C_{\text{infer}} = R(n_{\text{attn}}w^{\text{infer}}_{\text{attn}} + n_{\text{mlp}}w^{\text{infer}}_{\text{mlp}} + n_{\text{conn}}w^{\text{infer}}_{\text{conn}}) + c^{\text{infer}}_{\text{read}}$.

B Additional results

We list here every language-model run and the protocols behind the decode and pruning comparisons.

B.1 Looped Transformer runs

Table 5 lists the remaining Looped Transformer runs.

B.2 Connection gates matter most for standard Transformers

Without connection gates, the standard-Transformer searches save fewer FLOPs and do worse than the deep model. To isolate the benefit of allowing the model to learn its own connections, we compare a carved sublayer-only ablation against the above carved *overconnected Transformers*. Specifically, we carve models by applying gates *only* to the attention and MLP sublayers, without any additional connections. On the standard Transformers this sublayer-ablation saves 19% and

Table 5: Non-headline Looped Transformer runs. The longer 1.3B runs of Table 1 (91 tokens per parameter) are in Table 13. Top: each size's native budget (100, 100, 20, and 20 tokens per parameter from 140M to 1.3B). Training FLOPs are matched to the paired deep run within 3.5%. Bottom: the 20-tokens-per-parameter runs at 140M and 370M behind Figure 4a, matched within 12%. PPL is token perplexity on the validation window; CORE-11 uses our protocol in Appendix C.2 and – means we did not score a run.

Size	Topology	Seed	Train tokens (B)	Train FLOPs	Val. BPB	PPL	CORE-11	Inference FLOPs
Native budget								
140M	Deep	0	15.099	1.0000×	0.8914	16.27	–	1.0000×
	Deep	1	15.099	1.0000×	0.8917	16.28	–	1.0000×
	9-layer FLT	0	25.751	1.0000×	0.9135	17.43	–	0.5864×
	9-layer FLT	1	25.751	1.0000×	0.9088	17.18	–	0.5864×
	Sublayer-only ablation	0	27.228	1.0000×	0.9143	17.47	–	0.5550×
	Sublayer-only ablation	1	27.228	1.0000×	0.9238	18.00	–	0.5545×
370M	Deep	0	37.513	1.0000×	0.8043	12.39	–	1.0000×
	Deep	1	37.513	1.0000×	0.8022	12.30	–	1.0000×
	11-layer FLT	0	66.404	1.0000×	0.8213	13.06	–	0.5649×
	11-layer FLT	1	66.404	1.0000×	0.8331	13.56	–	0.5649×
	CARVING	0	68.915	1.0000×	0.8165	12.87	–	0.5443×
	CARVING	1	68.915	1.0000×	0.8215	13.07	–	0.5442×
	Sublayer-only ablation	0	71.471	1.0000×	0.8177	12.92	–	0.5249×
	Sublayer-only ablation	1	71.471	1.0000×	0.8200	13.01	–	0.5249×
770M	Deep	0	16.032	1.0000×	0.7961	12.07	26.7	1.0000×
		1	16.032	1.0000×	0.8002	12.23	28.9	1.0000×
	11-layer FLT	0	28.584	1.0000×	0.8006	12.24	25.8	0.5609×
	CARVING	0	30.004	1.0000×	0.7957	12.06	27.9	0.5343×
		1	30.004	1.0000×	0.7956	12.05	27.1	0.5343×
	Sublayer-only ablation	0	29.401	1.0000×	0.8006	12.24	–	0.5453×
1.3B	Deep	0	26.172	1.0000×	0.7494	10.43	33.8	1.0000×
		1	26.172	1.0000×	0.7500	10.45	35.3	1.0000×
	13-layer FLT	0	49.251	1.0344×	0.7531	10.55	33.0	0.5497×
		1	49.251	1.0344×	0.7532	10.56	34.1	0.5497×
	CARVING	0	49.251	1.0016×	0.7487	10.41	32.3	0.5323×
		1	49.251	1.0016×	0.7488	10.41	32.6	0.5323×
	Sublayer-only ablation	0	50.430	1.0029×	0.7486	10.41	–	0.5205×
		1	50.430	1.0029×	0.7490	10.42	–	0.5205×
20 tokens per parameter								
140M	Deep	0	2.815	1.0000×	0.9490	19.48	12.0	1.0000×
	9-layer FLT	0	5.374	1.1195×	0.9540	19.79	15.5	0.5864×
	CARVING	0	5.117	1.0000×	0.9571	19.98	14.2	0.5501×
	Sublayer-only ablation	0	5.374	1.0588×	0.9564	19.94	–	0.5545×
370M	Deep	0	7.832	1.0000×	0.8568	14.60	20.4	1.0000×
	11-layer FLT	0	13.604	0.9812×	0.8664	15.04	22.8	0.5649×
	CARVING	0	13.604	0.9454×	0.8643	14.94	23.0	0.5443×
	Sublayer-only ablation	0	14.428	0.9668×	0.8668	15.06	–	0.5249×

38% of inference FLOPs at 1.3B and 2.7B, but ends 0.0021 and 0.0040 BPB above the deep model, respectively, where CARVING saves 24% and 44% and matches the deep model (Appendix B.3). On the looped Transformers the ablation ends 0.0025 and 0.0050 BPB above CARVING at 370M and 770M, respectively, at the budgets shown in Figure 7. Given that a connection costs $3d$ training FLOPs per token versus $48d^2$ for an MLP, the *overconnected Transformer* offers a cheap yet more expressive route to keep direct paths between Transformer layers at almost no cost.

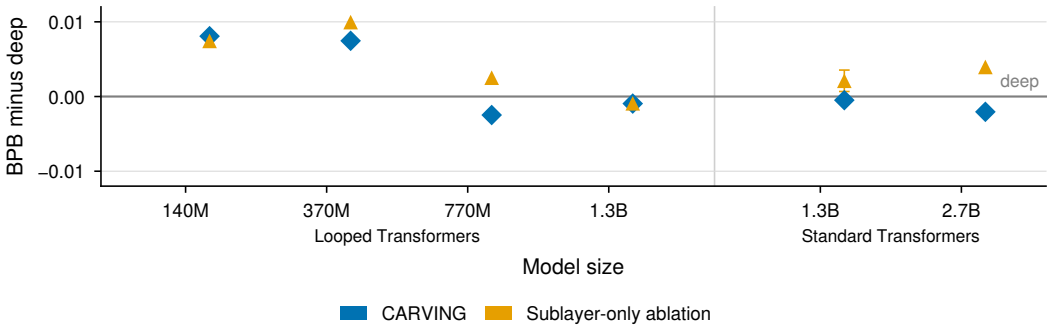


Figure 7: **The sublayer ablation shows that connection gates help improve performance.** Validation bits per byte minus that of the deep model trained for the same FLOPs, for CARVING (sublayer and connection gates) and the sublayer-only ablation at each size. Means of two seeds.

B.3 Standard Transformer runs

Each carved standard Transformer continues from the search weights, counting search inside the training budget (Section 2.3). Table 6 gives the paired deep references and realized FLOP ratios, including deviations from the matched budget.

The shallower standard Transformer baselines (Table 3) are trained from scratch.

Table 6: Standard Transformer runs trained from the search weights. The 40-tokens-per-parameter runs of Table 1 are in Table 14. Steps are optimizer steps at 524,288 tokens each, counted from the start of search; costs are in EF (10^{18} FLOPs), and the ratio compares search plus training with the paired deep reference. The shallower BPB is shown where that baseline was trained: 36,288 steps (18 layers) and 32,128 steps (20 layers) at 1.3B, and 93,565 steps (18 layers) at 2.7B.

			Steps			Cost (EF)				Val. BPB		
Size	Topology	Seed	Compile	Final	Deep	Search	Training	Deep	Ratio	Variant	Deep	Shallower
1.3B	CARVING	0	2,861	35,893	27,821	13.589	118.554	132.147	1.0000×	0.8145	0.8151	0.8169
1.3B	CARVING	1	2,051	35,083	27,821	9.742	118.555	132.147	0.9709×	0.8147	0.8151	0.8169
1.3B	Sublayer-only ablation	0	2,171	32,943	27,131	10.312	118.555	128.869	1.0000×	0.8196	0.8161	0.8157
1.3B	Sublayer-only ablation	1	2,791	33,147	27,131	13.257	118.554	128.869	1.0228×	0.8168	0.8161	0.8157
2.7B	CARVING	0	2,981	93,565	53,380	14.926	472.137	500.055	0.9740×	0.7548	0.7568	0.7575
2.7B	CARVING	1	3,012	95,228	53,380	13.119	472.137	500.055	0.9704×	0.7540	0.7568	0.7575
2.7B	Sublayer-only ablation	0	5,261	86,128	55,660	49.284	472.134	521.414	1.0000×	0.7587	0.7548	–

Table 7 lists the 1.3B carved standard Transformers retrained from scratch at the deep model's training FLOPs, as for looped models, each with a shallower baseline at the same FLOPs (Section 4.1).

Table 7: Carved 1.3B standard Transformers retrained from scratch at the deep model's training FLOPs (24,960 deep steps). Inference FLOPs are relative to the deep standard Transformer; each shallower baseline is trained for the same FLOPs.

Topology	Inference FLOPs	Steps	BPB	Deep BPB	Shallower BPB
CARVING	0.56×	44,923	0.828920	0.819331	0.828956 (13 layers)
CARVING	0.76×	33,032	0.820556	0.819331	0.821691 (18 layers)
Sublayer-only ablation	0.81×	30,772	0.822677	0.819331	–

At 0.76× inference FLOPs, the carved topology retrained from a fresh initialization matches the deep model (0.0012 BPB above). At 0.56× inference FLOPs, however, retraining from scratch falls 0.0096 BPB behind, which indicates that search-weight continuation becomes important at more aggressive compression as is used in our headline comparisons.

B.4 The 15% search budget

The headline 1.3B looped topology uses fixed-target inequality search stopped at 15% of the reference training FLOPs. Separately, we validate early stopping by retraining the topologies saved at 15% and 100% of an annealed-equality search. At matched retraining FLOPs, their BPBs differ by only 0.0017, within our 0.003 matching threshold (Table 8). This validation uses a 10-tokens-per-parameter deep reference; the headline models train longer.

Table 8: Validating the 15% search budget at 1.3B. Topologies saved at 15% and 100% of the annealed-equality search are retrained at matched FLOPs, using a 10-tokens-per-parameter deep reference. Inference FLOPs are relative to the deep FLT. These checkpoints come from a separate validation search; the headline topology comes from the fixed-target inequality search stopped at 15% of reference training FLOPs.

Model	Search checkpoint	Seed	Params (M)	Inference FLOPs	Retrain tokens (B)	Val. BPB
CARVING	15%	0	755	0.5381×	24.320	0.7778
CARVING	100% (full)	0	772	0.5323×	24.675	0.7761
13-layer FLT	–	0	755	0.5497×	23.805	0.7810

B.5 Decode throughput measurement

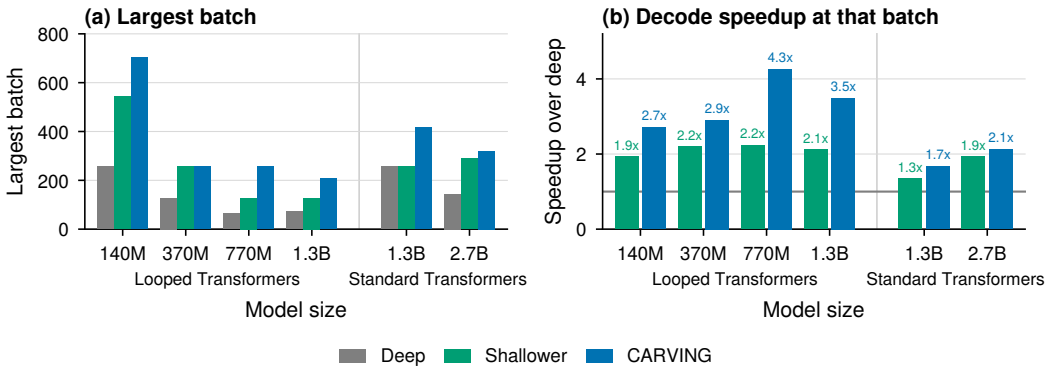


Figure 8: **Carved models fit larger batches and decode fastest at every size.** One H100. **a.** The largest tested batch each model fits. **b.** Decode throughput at that batch relative to the dense model of the same size. The carved model's gain exceeds the shallower dense model's at every size because it keeps fewer attention sublayers and so less key-value cache per token.

Each model and batch size runs in a fresh process on one 80 GB H100. We use JAX with just-in-time compilation, bf16 parameters and KV cache, a 1,024-token context, 256 reserved decode positions, and greedy decoding. We take each point as the median of 64 timed decode steps after three warm-up steps (to adjust for GPU warm-up). We exclude compilation and prefill, each timed step waits for the device to finish, and the KV-cache buffer is reused between steps.

We report the largest tested batch that fits, together with the next tested batch that failed. For the 1.3B models the next tested batches (80, 144, 224, and 320 for the deep, 13-layer, carved, and sublayer-only looped models; 288 and 448 for the deep and carved standard Transformers) ran out of memory. Figure 8 gives the largest tested fitting batch and the speedup at every size. At 4,096- and 8,192-token contexts the 1.3B carved looped model fits 3.2× the deep batch at both and decodes

$3.4\times$ and $3.6\times$ faster. The sublayer-only ablation fits $4.0\times$ and $4.4\times$ and decodes $4.5\times$ and $4.9\times$ faster.

KV cache. For a Fully Looped Transformer, the bf16 KV cache grows by $4Rn_{\text{attn}}H_{\text{KV}}d_h$ bytes per token, where H_{KV} is the number of key/value heads and the factor four counts two bytes each for the key and value entries. At 1.3B, $H_{\text{KV}}d_h = d$, so the deep FLT, the 13-layer FLT, CARVING, and the sublayer-only ablation use 768, 416, 256, and 192 KiB per token.

Quantization. When we quantize the key-value cache using int8 precision, it leaves validation BPB unchanged to four decimals. Table 9 gives the largest tested fitting batch and the throughput at it with this cache. The batch-1 fp8 throughput below comes from a separate PyTorch engine (torch 2.14, torchao 0.18) with per-row fp8 weight-only quantization and a bf16 cache, because the JAX engine has no fused low-precision matmul. Its bf16 baseline is faster than the JAX engine's, so we report the ratios within it for a fair comparison, and its BPB is scored on a one-million-token slice of the validation set. We show that fp8 weights speed the deep 1.3B looped model up by $1.36\times$ at batch 1 for $+0.006$ BPB, CARVING in bf16 is $2.0\times$ faster than deep, and CARVING with fp8 weights is $2.8\times$ faster at $+0.016$ BPB.

Table 9: Largest fitting batch and decode throughput with an int8 key-value cache, under the protocol above. The ratio is to the bf16 throughput of Table 1.

Model	Largest batch	Decode tok/s	Ratio to bf16
1.3B looped Transformer			
Deep (24 layers)	128	2,780	1.60×
Shallower (13 layers)	256	5,914	1.60×
CARVING	416	9,466	1.56×
2.7B standard Transformer			
Deep (32 layers)	288	7,221	1.36×
Shallower (18 layers)	512	13,295	1.29×
CARVING	576	15,049	1.33×

B.6 RECOVERING THE LEARNED TOPOLOGY FROM A DEEP CHECKPOINT

The 140M scores of Table 10 use the fully trained deep FLT checkpoint and validation blocks 0–2,047. Let W_o be a sublayer's output-projection matrix. Output ablation is the increase in cross-entropy when that projection is zeroed; the Taylor score is $-\langle \nabla_{W_o}\ell, W_o\rangle$; the projection norm is $\|W_o\|_F$; and the sublayer norm is the Frobenius norm of all parameters in the sublayer. Each score is computed once for every sublayer, without rescoring after removals. Each ranking then removes the lowest-scoring sublayers until the realized FLOP fraction is at most 0.528846, the searched topology's value. The three random masks are cost-matched controls. No topology in this table was retrained.

Table 10: Agreement between the deep model's checkpoints and the searched topology. $n^{\text{rem}}_{\text{attn}}$ and $n^{\text{rem}}_{\text{mlp}}$ count removed attention and MLP sublayers. We denote J_A, J_M, and J_{all} as the Jaccard overlaps of the removed sets with the set the search removed.

Setting	Ranking	$n^{\text{rem}}_{\text{attn}}$	$n^{\text{rem}}_{\text{mlp}}$	ρ_{real}	J_A	J_M	J_{all}
140M	Output ablation	7	9	0.504808	0.364	0.333	0.348
140M	Taylor	8	7	0.528846	0.333	0.400	0.364
140M	Random, draw 0	9	6	0.524038	0.417	0.300	0.364
140M	Random, draw 1	9	6	0.524038	0.417	0.300	0.364
140M	Random, draw 2	6	10	0.509615	0.400	0.417	0.409

B.7 PRUNING AND RETRAINING THE DEEP MODEL

In Section 4.2, we prune the trained deep model to the carved model's inference FLOPs using output ablation, the increase in validation loss when the sublayer's output is removed (Kim et al., 2024), its first-order Taylor estimate (Ma et al., 2023), or a random order. Each importance score is computed once on the trained deep checkpoint and applied greedily. We keep the last mask whose FLOP fraction is at or above the carved model's (so the pruned model is never cheaper than the carved one) and retrain the candidate with the lowest pre-retraining validation BPB at that fraction (Table 11).

We retrain from the deep weights of the sublayers that pruning kept. Then, we train for the FLOPs of the CARVING search at the pruned model's per-token cost, so that pruning plus retraining equals the carved model's search plus training. Retraining uses the same optimization recipe with peak learning rate 0.0067, a 3% warmup because the deep checkpoint's learning rate had decayed to zero, and a cooldown to 10% of the peak. The distillation variant (Hinton et al., 2015; Muralidharan et al., 2024) replaces the language-model loss with the forward KL divergence from the deep model's next-token distribution to the pruned model's on the same tokens. Table 12 lists the runs.

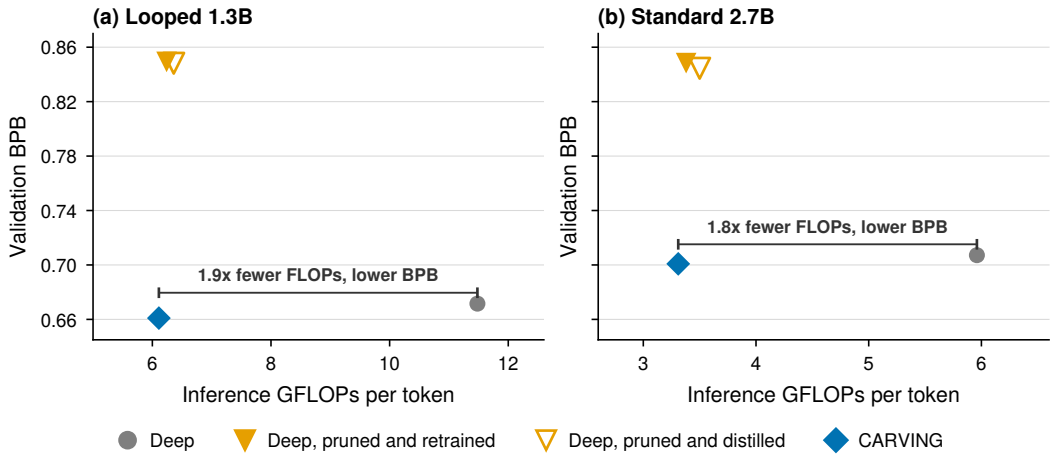


Figure 9: **Pruning and retraining the deep model does not reach CARVING.** Validation bits per byte against inference GFLOPs per token for (a) the 1.3B looped Transformer and (b) the 2.7B standard Transformer. Orange: the deep model pruned to CARVING's inference FLOPs and retrained for the carved model's total compute, with the language-model loss (filled) or by distillation from the deep model (hollow).

Table 11: Validation bits per byte on the fixed validation window of the deep model pruned under each heuristic to the carved model's FLOP fraction, before retraining. The pruned model with the lowest BPB in each column is retrained in Table 12.

Order	Looped 1.3B (0.53×)	Standard 2.7B (0.56×)
Output ablation per FLOP	7.82	**1.51**
Taylor per FLOP	9.59	3.54
Random	**4.28**	3.83

Table 12: Pruning and retraining at CARVING's target inference cost, using the training-compute protocol in Appendix B.7. LM and distilled denote language-model-loss training and distillation from the deep model. BPB uses the fixed validation window; a dash marks an unavailable evaluation.

Model	Pruning order	Sublayers removed	Retraining steps (LM / distilled)	Validation BPB ↓		CORE-22 (LM / distilled)
				LM	Distilled	
Looped 1.3B	Random	23 of 48	23,829 / 14,774	0.8492	0.8482	35.82 / 35.94
Standard 2.7B	Output ablation per FLOP	30 of 64	6,773 / 4,265	0.8483	0.8445	34.48 / 34.72

C Evaluation protocol

C.1 Validation window and bits per byte

Every model is evaluated on the same 16,384 non-overlapping source blocks of 2,048 tokens, beginning at validation block 90,286, which contain 151,467,293 target bytes. Each block is converted back to bytes, split into documents at the source end-of-text token, and retokenized with the evaluated model's tokenizer.

Sequences that are longer than the model's context are scored in overlapping windows, each shifted by half the context, with each target token scored once.

We use bits per byte primarily, as the byte denominator is the same for every tokenizer. Therefore, BPB is comparable across models whereas perplexity is not.

C.2 Downstream evaluation

For task i, let acc_i be accuracy and base_i its task baseline. We report

$$\mathrm{score}_i = \frac{\mathrm{acc}_i - \mathrm{base}_i}{1 - \mathrm{base}_i}, \qquad \text{CORE-11} = \frac{100}{11} \sum_{i=1}^{11} \mathrm{score}_i \,.$$

The eleven tasks are HellaSwag (0-shot), ARC-Easy (10-shot), ARC-Challenge (10-shot), COPA (0-shot), PIQA (10-shot), OpenBookQA (0-shot), WSC273 (0-shot), WinoGrande (0-shot), BoolQ (10-shot), AGIEval LSAT-AR (0-shot), and LAMBADA OpenAI (0-shot). Chance is 25% for HellaSwag, both ARC tasks, and OpenBookQA; 50% for COPA, PIQA, WSC273, and WinoGrande; 20% for LSAT-AR; and 0% for LAMBADA. BoolQ uses its 62% majority-class baseline.

CORE-22 applies the same scoring to the 22 tasks of DCLM's CORE suite (Li et al., 2024) (Table 15), that is, $\text{CORE-22} = 100 \sum_i n_i \ \mathrm{score}_i \,/ \sum_i n_i$, over 91,037 problems. It is the mean over three few-shot seeds, and $\pm$ gives their standard deviation.

Gen-9 decodes instead of scoring log-likelihoods. DCLM scores its nine language-modeling tasks (Jeopardy, BIG-bench QA Wikidata, LAMBADA, BIG-bench Dyck languages, CS algorithms, Operators, Repeat-copy logic, SQuAD, CoQA) teacher-forced, and we render the same prompts (llm-foundry's few-shot format (MosaicML NLP Team, 2023), 10 shots except LAMBADA and CoQA at 0) and decode greedily under llm-foundry's generation protocol. The prompt is left-truncated to 2048 minus the task's token cap, the cap is the longest gold answer in the task and decoding stops at end-of-text. Dyck answers are punctuation only, so Dyck is scored by exact match of the first generated line, Jeopardy is the mean over its five categories, and Gen-9 averages the nine tasks.

pass@k draws k samples per problem on the same nine tasks and on GSM8K, and counts a problem as solved when any of the k samples is correct. We compare models at equal inference FLOPs, such that each cheaper model draws as many samples as fit in the decode FLOPs the deep model spends on k samples.

Our models use lm-eval 0.4.12 with few-shot seed 1234, character-length-normalized accuracy where available and unnormalized accuracy for LAMBADA, BoolQ, COPA, WSC273, and WinoGrande. The 10-shot evaluations use a maximum length of 2,048 and the 0-shot evaluations use each model's own maximum length.

Evaluation	Deep (1.31B)	Shallower (755M)	CARVING (772M)
Validation and CORE			
Validation (bpb) ↓	0.6716	0.6706	**0.6609**
CORE-22	**47.24 ± 0.05**	44.21 ± 0.09	44.59 ± 0.03
CORE-11	**43.62**	41.67	41.64
Gen-9, normalized prefix match (%)	**42.7**	39.8	40.3
pass@k, 9 generative CORE tasks (%)			
pass@1	44.0	**48.7**	48.1
pass@8	67.6	**70.2**	69.5
pass@16	72.9	**74.7**	73.6
pass@k, GSM8K (%)			
pass@1	3.4	2.6	**3.8**
pass@8	17.8	15.0	**19.8**
pass@16	27.2	23.4	**29.7**

Table 13: Full final evaluation, 1.3B looped Transformer.

Evaluation	Deep (2.64B)	Shallower (1.54B)	CARVING (1.49B)
Validation and CORE			
Validation (bpb) ↓	0.7072	0.7045	**0.7008**
CORE-22	**45.44 ± 0.02**	42.83 ± 0.07	42.54 ± 0.08
CORE-11	**41.49**	39.97	39.98
Gen-9, normalized prefix match (%)	**40.9**	38.6	40.3
pass@k, 9 generative CORE tasks (%)			
pass@1	43.4	46.7	**49.6**
pass@8	65.8	66.1	**68.9**
pass@16	71.0	70.3	**72.8**
pass@k, GSM8K (%)			
pass@1	0.7	**1.8**	1.4
pass@8	5.2	**10.9**	8.8
pass@16	9.3	**17.7**	14.6

Table 14: Full final evaluation, 2.7B standard Transformer.

Evaluation	Deep (1.31B)	Shallower (755M)	CARVING (772M)
CORE-22			
HellaSwag, 0-shot	**5.96**	5.37	5.45
Jeopardy	**0.97**	0.87	0.92
BIG-bench QA Wikidata	**14.52**	14.15	14.32
ARC-Easy	**1.79**	1.70	1.75
ARC-Challenge	**0.39**	0.32	0.35
COPA	0.05	0.04	**0.06**
CommonsenseQA	0.18	0.01	**0.24**
PIQA	**1.11**	1.07	1.02
OpenBookQA	**0.12**	0.11	**0.12**
LAMBADA	**3.51**	3.37	3.27
HellaSwag, 10-shot	**6.36**	5.69	5.81
Winograd	**0.17**	0.15	0.15
WinoGrande	0.28	**0.32**	0.30
BIG-bench Dyck languages	**0.17**	0.15	0.15
AGIEval LSAT-AR	0.01	**0.02**	**0.02**
BIG-bench CS algorithms	0.58	0.53	**0.60**
BIG-bench operators	**0.07**	0.06	**0.07**
BIG-bench repeat copy logic	**0.00**	**0.00**	**0.00**
SQuAD	**5.86**	5.54	5.05
CoQA	**3.41**	3.09	3.07
BoolQ	-0.21	-0.24	**-0.10**
BIG-bench language identification	1.95	1.88	**1.97**
CORE-22	**47.24**	44.21	44.59
pass@16 at equal inference FLOPs			
Jeopardy	**10.11**	10.02	9.93
BIG-bench QA Wikidata	11.60	11.82	**12.12**
LAMBADA	11.36	**11.49**	11.34
BIG-bench Dyck languages	5.00	**6.24**	5.51
BIG-bench CS algorithms	12.32	12.57	**12.59**
BIG-bench operators	1.81	1.79	**1.86**
BIG-bench repeat copy logic	0.09	**0.10**	0.06
SQuAD	**10.88**	10.74	10.54
CoQA	9.69	**9.93**	9.69
pass@16, nine tasks	72.9	**74.7**	73.6

Table 15: Per-task evaluation, 1.3B looped Transformer.

D TOPOLOGY STATISTICS

Figure 10 shows 20 searches; the 15% validation checkpoint is not counted as another search.

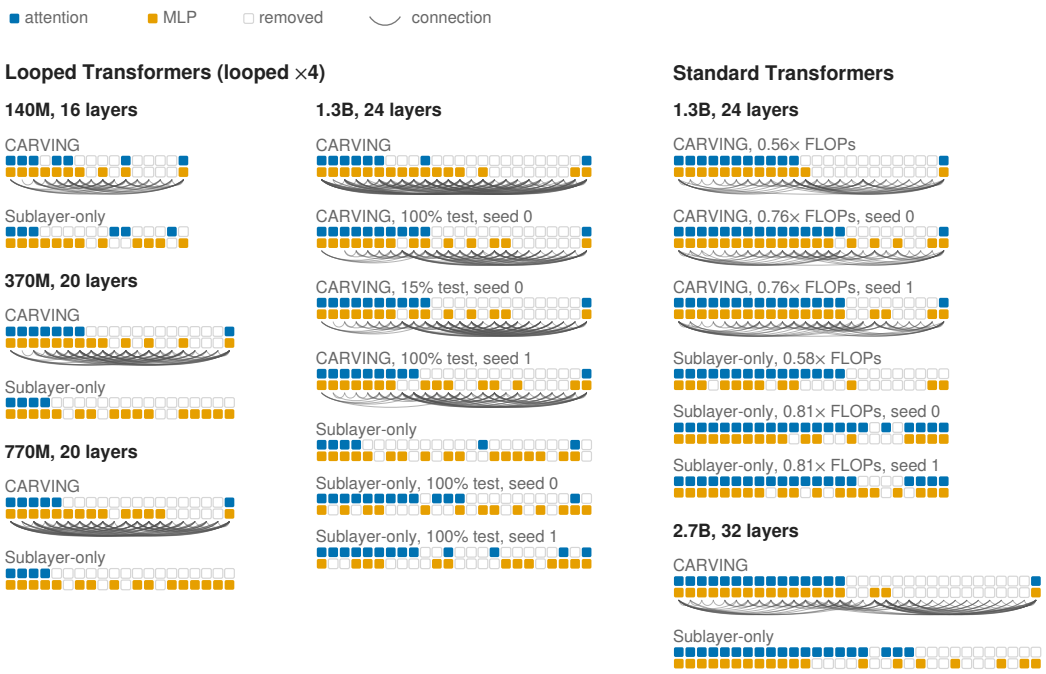


Figure 10: Main-experiment topologies use at most 15% of the reference training FLOPs; validation rows include full-search comparisons. Rows show retained attention (blue, top) and MLP (orange, bottom), from first to last layer (left to right).

Topology numbers from Section 5.1. Over the 20 searches, every topology keeps the first layer's attention and 19 keep the last layer's MLP. In 15, at least three quarters of the retained attention sublayers form a contiguous run from the first layer. In 14 (10 of the 12 FLT and 4 of the 8 standard-Transformer topologies), more MLP than attention sublayers are retained. In all 10 CARVING topologies, the last layer receives the most incoming connections. See Table 16 for seed variance.

Table 16: Topology agreement across matched seed pairs. $\overline{J}_{\text{rand}}$ is the exact expectation for random masks with matching retained counts. A, M, and C denote attention, MLP, and connections.

Pair	J_A	$\overline{J}_{A,\text{rand}}$	J_M	$\overline{J}_{M,\text{rand}}$	J_C	$\overline{J}_{C,\text{rand}}$
FLT, CARVING	0.909	0.286	0.611	0.437	0.493	0.222
FLT, ablation	0.625	0.377	0.368	0.377	–	–
Standard, CARVING	1.000	0.504	0.810	0.657	0.660	0.177
Standard, ablation	0.909	0.776	0.636	0.600	–	–

E SYNTHETIC-TASK DETAILS

Tasks. S_5 is a group-composition task over permutations of five objects. An input has the form $[\texttt{BOS}, g_1, \ldots, g_k, \texttt{QUERY}]$ with $k \leq 8$, where BOS is the beginning-of-sequence token, and the target is the composition of the permutations (Liu et al., 2023). ListOps adapts the prefix-arithmetic task of Nangia & Bowman (2018). Its leaves are digits, its operators are MIN, MAX, median, and sum modulo ten, and the target is the value at the root. We use expressions with nesting depth at most five. For both tasks, we apply the loss only at the fixed QUERY position.

Data. Each task has 262,144 training examples, 4,096 validation examples, and 4,096 test examples, split deterministically by hashing. S_5 has context length of 10 and vocabulary size 128, while ListOps uses context length 329 and vocabulary size 18. S_5 and ListOps report validation results, and modular

addition reports test results after a validation screen (below). After duplicate rows are removed, the validation sets contain 2,017 unique S_5 examples and 3,485 unique ListOps examples.

Model. The synthetic-task FLT has three blocks, four repeats, width 64, four attention heads, and rotary position embeddings. Parameters use fp32 and activations bf16. The search space is the six sublayers, one attention and one MLP per block.

Search recipe. The search uses a batch of 256 examples and the optimizers of Appendix A.2, with learning-rate multipliers 1 for the matrices and 0.15 for the input embedding and a gate learning rate of 0.02. The learning rate has no warm-up and decays linearly over the final 20% of training (same as our language modeling setting). The controller uses the equality constraint of Appendix A.4 and anneals its target from 0.983675 to the stated value on a cubic schedule over 3,871 controller updates (5,071 for the S_5 search at target 0.35).

Search, retraining, and controls. CARVING uses target FLOP fraction 0.5 on S_5 and ListOps, with an additional 0.85 setting on ListOps, and 0.22 on modular addition. All use seeds 0–2. We discard search weights and retrain each compiled topology with the search optimizer and schedule, matching the deep FLT's training FLOPs to the nearest whole step below the budget. The S_5 control keeps three MLPs and no attention; the ListOps control keeps one attention and three MLPs.

Table 17 reports exact match over the examples defined above.

Table 17: Synthetic-task topologies and retraining results at matched FLOPs. A dash means none. Modular addition reports test accuracy (three unordered CARVING retrains) and the relative FLOPs of Figure 6.

Task	Condition	Seed	Target	ρ_{real}	Retained attention	Retained MLP	Train tokens	Val. exact match
S_5	Deep	0	–	1.0000	1, 2, 3	1, 2, 3	25.603M	0.9985
S_5	Deep	1	–	1.0000	1, 2, 3	1, 2, 3	25.603M	1.0000
S_5	Deep	2	–	1.0000	1, 2, 3	1, 2, 3	25.603M	0.9990
S_5	CARVING	0	0.50	0.5668	1, 2, 3	1	44.705M	1.0000
S_5	CARVING	1	0.50	0.3503	1, 2, 3	–	71.322M	1.0000
S_5	CARVING	2	0.50	0.5668	1, 2, 3	3	44.705M	0.9995
S_5	CARVING	0	0.35	0.2335	2, 3	–	105.029M	0.9960
S_5	Hand-designed: MLP-only	0	–	0.6497	–	1, 2, 3	39.117M	0.0178
ListOps	Deep	0	–	1.0000	1, 2, 3	1, 2, 3	842.324M	0.7716
ListOps	Deep	1	–	1.0000	1, 2, 3	1, 2, 3	842.324M	0.8379
ListOps	Deep	2	–	1.0000	1, 2, 3	1, 2, 3	842.324M	0.8141
ListOps	CARVING	0	0.50	0.5470	1, 2	1	1.538B	0.7661
ListOps	CARVING	1	0.50	0.4270	1, 2	–	1.968B	0.7383
ListOps	CARVING	2	0.50	0.4270	1, 3	–	1.968B	0.7386
ListOps	CARVING	0	0.85	0.8803	1, 2, 3	1, 3	956.616M	0.8301
ListOps	CARVING	1	0.85	0.8803	1, 2, 3	1, 3	956.616M	0.8228
ListOps	CARVING	2	0.85	0.8750	1, 2, 3	1, 3	956.616M	0.8432
ListOps	Hand-designed: MLP-heavy	0	–	0.5727	1	1, 2, 3	1.469B	0.7090

Modular addition (test)	Seeds	Relative FLOPs	Retained attention	Retained MLP	Test exact match
Deep	3	1.00×	1, 2, 3	1, 2, 3	0.999 (mean)
CARVING	3	0.16×	1	–	1.0000, 1.0000, 0.9025
MLP-heavy control	1	0.68×	–	1, 2, 3	0.008

Modular addition and the exhaustive search. We predict $(a + b) \bmod 113$ from `BOS`, a, b, `QUERY` (length 4, vocabulary 116), using the same FLT recipe. Content hashes split the 113^2 pairs 3:1:1 into training, validation, and test. We retrain all 63 non-empty sublayer subsets from scratch, three seeds each, at the deep baseline's training FLOPs. Mean validation exact match ≥ 0.95 over unique examples solves the task; values within 0.005 are tied. We test each run once (Figure 6). Three target-0.22 searches retain only the first attention sublayer: an Attention-only Transformer!