
FractalPI: Spending Parameters on a Routed Multiscale Hierarchy Instead of Depth

Krzysztof Urbanowicz
QWID Foundation

krzysztof.urbanowicz@qwid.org

Andrzej Urbanowicz
QWID Foundation

andrzej.urbanowicz@qwid.org

Mateusz Mach
Nextrope

mm@nextrope.com

Abstract

Dense transformers convert non-embedding parameters into quality along a tight power law, and that law is largely indifferent to how the parameters are arranged into depth and width (Kaplan et al., 2020). A smaller model is therefore bought at a predictable price, and rearranging a dense stack does not change the exchange rate. We ask whether a different block structure can sit *below* that curve. We introduce *FractalPI*¹², a block that recursively splits the channels of a residual stream into a self-similar tree: at every level a learned, per-token, context-conditioned assignment routes channel groups into branches, each branch is processed at its own width, and the branches are merged by the exact transpose of the same assignment. Because level ℓ spans the same d channels in m^ℓ branches of width d/m^ℓ , the parameters of the tree form a geometric series: depth in the tree is nearly free in parameters while adding compute passes. Measured against a depth ladder of dense GPT-2 baselines trained under an identical protocol ($L = 1.553 N^{-0.0443}$, $R^2=0.978$ over seven depths), the fractal family lies below the dense law on non-embedding parameters and on top of it on compute: it matches a 3-layer baseline with 22% fewer non-embedding parameters and a 4-layer one with 9% fewer, while at equal quality it needs 2-3% *more* FLOPs per token. The saving is an offset rather than a better exponent, and we locate where it ends: in parameter-matched pairs at 101M and 126M non-embedding parameters the dense model wins by 0.005 and 0.014 bits per byte, so the advantage holds over roughly 25-63M and reverses above it. At the top of the measured range a 122M fractal matches the 127M dense baseline within seed noise (1.282 ± 0.002 vs. 1.279 ± 0.0004 bits per byte) with a higher mean final-word accuracy. Paired learning-rate sweeps coincide over two decades, so the parity is not a tuning artifact, and a six-task zero-shot suite shows no detectable difference. We state the costs with the same emphasis: training wall-clock is $1.37\text{-}1.86\times$ that of the baseline and the ratio grows with width, the learning-rate optimum is narrower, and under data repetition the architecture behaves like the dense model rather than regularizing it. FractalPI is parameter-efficient, compute-neutral and wall-clock-expensive - an alternative worth taking when memory, not time, is the binding constraint.

1 Introduction

The cost of a language model is paid in three currencies - parameters, arithmetic and wall-clock time - and a dense transformer converts the first of them into quality along a remarkably tight power law. On

¹The name records the construction: the block applies the same split-process-merge motif at every scale of the channel axis, so a level is a scaled copy of the one above it, and *PI* abbreviates the per-token permutation-induced routing that decides which channels meet at each scale.

²Code: <https://github.com/wonabru/fractalpi>

our own depth ladder, five GPT-2 baselines from two to ten layers trained under an identical protocol fit $L = 1.553 N^{-0.0443}$ with $R^2=0.978$, where N counts non-embedding parameters. What makes this curve a constraint rather than an observation is a second, older result: at a fixed N the loss is almost indifferent to how those parameters are distributed between depth and width, varying by a few per cent across a fortyfold range of aspect ratios (Kaplan et al., 2020). Within the dense family, then, there is no rearrangement to be found. A model half the size costs what the curve says it costs.

This paper asks whether the curve itself can be moved by changing what a block *is*. Our proposal starts from an observation about where a transformer spends its parameters: every layer mixes all d channels with every other, at full width, and pays $\Theta(d^2)$ for the privilege, layer after layer. A self-similar hierarchy spends them differently. FractalPI splits the residual stream into m branches of width d/m , splits each of those again, and so on for D levels; every level still spans the same d channels, only at a finer granularity, so the width of level ℓ falls as d/m^ℓ while its branch count rises as m^ℓ . The parameters of successive levels therefore form a geometric series - each level costs a fixed fraction of the one above it - and the whole tail below the first level costs less than a single dense layer. Depth in the tree is nearly free in parameters; what it costs is compute passes.

A hierarchy of narrow branches is only useful if something decides *which* channels share a branch, and that decision has to depend on the token rather than on the channel index. A learned router produces, for every token and every split, a doubly-stochastic assignment of channel groups to branch slots; the upward path merges by the exact transpose of the same assignment, so the round trip through the tree inverts. Removing this component - freezing the assignment into a static permutation - costs 0.225 bits per byte, more than any other ablation we ran by an order of magnitude, and a control in which the routing is present but has nothing to mix collapses to the identity. Context-conditioned routing is the load-bearing part of the design, not a decoration on it.

Our main result is a comparison of two families rather than two models (Fig. 1). Across 57 runs at a fixed 300M-token budget, the fractal configurations sit below the measured dense law on the parameter axis and on it on the compute axis. In direct measurements, a fractal matches the three-layer dense baseline using 22% fewer non-embedding parameters, the four-layer one using 9% fewer, and the fitted frontier implies a saving of 5-40% at equal quality over that region, largest at the small end; note that this fit does not extrapolate past it (see below). The same comparison run on FLOPs per token gives 2-3% against us. The advantage is thus specific and one-dimensional: a better use of parameters, not of arithmetic.

At the top of the range the two families meet. A 122M fractal reaches 1.282 ± 0.002 bits per byte against 1.279 ± 0.0004 for the 127M dense baseline - a statistical tie rather than a win - with 6% fewer non-embedding parameters and a higher mean final-word accuracy on LAMBADA. We report this as parity because that is what three seeds support, and we note the corollary: since our frontier is *flatter* than the dense law ($N^{-0.0359}$ against $N^{-0.0443}$), the advantage closes as models grow, and beyond the measured range extrapolation favors the dense model. The place to use a fractal is below the parity point, not above it.

Two checks argue that the parity is structural rather than an accident of tuning. Learning-rate response curves for both architectures, measured under the same protocol and with dropout removed from both, coincide to within seed noise over the range $\times 1$ to $\times 30$ (Fig. 4); the two differ only at $\times 100$, where the dense model’s optimum is flat and ours is not. And on a six-task zero-shot suite evaluated at matched held-out loss, the mean difference in length-normalized accuracy is -0.001 ± 0.011 (Fig. 5).

We are equally explicit about what the architecture does not buy. Training wall-clock is $1.37\text{-}1.86\times$ that of the dense baseline in fp32 and the ratio *grows* with width; in bf16 it is worse still, because many small grouped matmuls and per-token routers are memory- and launch-bound where a dense stack issues few large GEMMs (Fig. 7). The learning-rate optimum is narrower. Under repeated data the architecture tracks the dense baseline rather than protecting against memorization, and ordinary dropout is a stronger regularizer than the structure is (Table 2). Finally, the densest baseline we trained remains unbeaten: at 75.6M non-embedding parameters no fractal in our sweep reaches its 1.279.

Contributions.

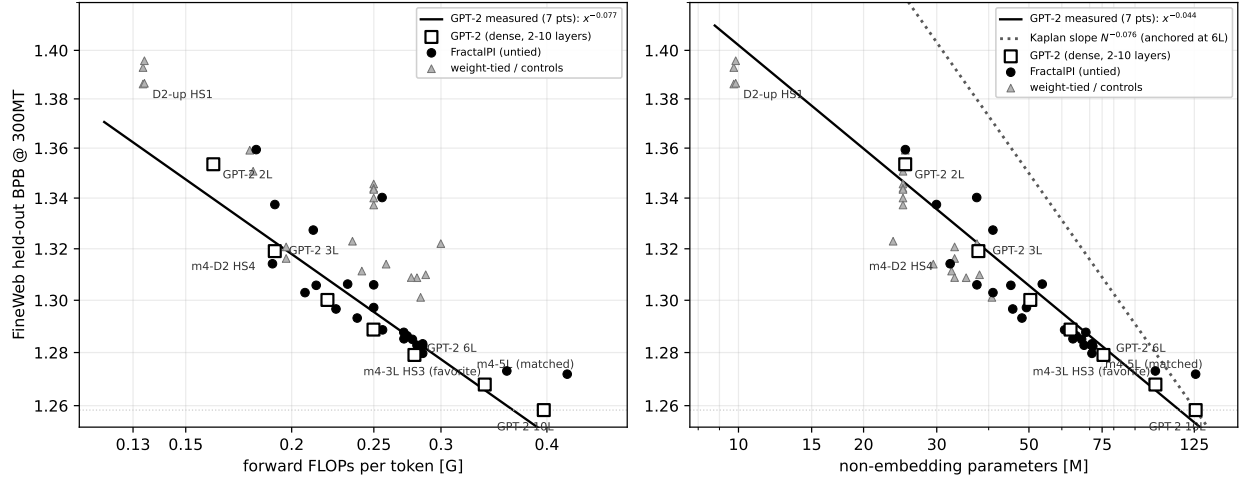


Figure 1: **Quality against cost, 57 runs** (300MT, tuned rate $\times 30$, no dropout, single seed unless noted; log-log axes). Held-out FineWeb BPB against forward FLOPs per token (left) and against non-embedding parameters (right), the axis of Kaplan et al. (2020); the embedding is excluded because the tiktoken vocabulary of 50,257 makes the tied embedding 51.5M, some 40% of every model here. Open squares: dense GPT-2 at depths 2 to 10 layers; filled circles: untied fractals; gray triangles: weight-tied variants and structural controls. Solid line: power law fitted to all seven GPT-2 depths, $L = 1.553 N^{-0.0443}$ ($R^2=0.978$) on the parameter axis and $L \propto C^{-0.077}$ on the FLOP axis. Dotted line: the exponent $N^{-0.076}$ of Kaplan et al. (2020), anchored at our 6L point, since that work reports nats per token on WebText2 while we report BPB on FineWeb. The fractal frontier quoted in the text is the Pareto envelope of the filled circles alone, fitted over the 17 configurations that define it; weight-tied variants and controls are plotted for context and do not enter that fit. The two rightmost circles are the parameter-matched pairs of Section 5.1: both lie above the dense line, and above the frontier fitted to the smaller configurations.

- A block that spends parameters on a routed self-similar channel hierarchy instead of depth, with a geometric rather than linear parameter cost in the number of levels.
- A measured scaling law for the dense baseline under our own protocol, and a family-versus-family comparison against it: a parameter saving of 3-22% at equal quality in direct matches, with no saving in compute, holding over roughly 25-63M non-embedding parameters and reversing above it (Fig. 1).
- Evidence that the resulting parity at 122M is not an artifact of learning rate or of the choice of benchmark, from paired learning-rate sweeps and a zero-shot understanding suite.
- An accounting of the costs - wall-clock, learning-rate tolerance and behavior under data repetition - and the ablations and negative controls that identify context-conditioned routing as the component the design rests on.

2 Related work

Sparse expert routing. Sparsely-gated MoE (Shazeer et al., 2017) introduced learned top- k gating with an auxiliary load-balancing loss; Switch Transformer (Fedus et al., 2022) simplified it to top-1 routing with the balancing objective we adopt. BASE layers (Lewis et al., 2021) formulate token-to-expert allocation as a linear assignment problem, guaranteeing balance by construction, and recent work routes with Sinkhorn normalization directly, removing auxiliary losses (SSR (Nguyen et al., 2025); sparsity-constrained OT (Liu et al., 2023)). Soft MoE (Puigcerver et al., 2024) replaces discrete dispatch with fully differentiable weighted mixing, Expert Choice (Zhou et al., 2022) inverts the direction so that experts select tokens, and Mixture-of-Depths (Raposo et al., 2024) routes tokens between compute and skip paths. All of these route *tokens* -

or, for model-conversion approaches such as DOT-MoE (Bamba et al., 2026) and M-SMoE (Li et al., 2024), statically assign *neurons*. We route *channel groups of each token*, context-dependently, inside one block. The consequence is a different economics: sparse experts add parameters that are idle for most tokens and buy quality per unit of *compute*, whereas every branch of our tree processes every token and the routing buys quality per unit of *parameter* - which is the axis on which Fig. 1 shows a gain and the axis on which it shows none.

Differentiable permutations and Sinkhorn layers. Learning permutation matrices via the Gumbel-Sinkhorn relaxation was introduced by Mena et al. (2018); Sinkhorn normalization itself is the standard entropic optimal-transport solver (Cuturi, 2013). Sinkformers (Sander et al., 2022) use it in place of the attention softmax; DSelect- k (Hazimeh et al., 2021) builds differentiable discrete selection on a similar relaxation; capsule networks (Sabour et al., 2017) route lower-level capsules to higher-level ones by iterative agreement. Our router is closest in spirit to capsule routing and DSelect- k , with two differences that matter for the result: the assignment acts on channel groups of individual tokens, and it is inverted on the merge path, so the round trip through a level is an identity rather than a learned map.

Hierarchical and multi-scale architectures. Hierarchical mixtures of experts date back to Jordan & Jacobs (1994), where a tree of gates makes coarse-to-fine decisions - the same intuition as our depth-dependent routing granularity. Hourglass transformers (Nawrot et al., 2022) downsample and upsample along the *sequence* axis, which reduces compute quadratically but leaves the parameter count of a layer untouched; we contract the *channel* axis instead, which is what makes the parameter cost of a level geometric in depth (Eq. (3)) and is the source of the saving reported here. FractalNet (Larsson et al., 2017) and fractal generative models (Li et al., 2025) exploit self-similar recursion in different senses - expansion-join rules and nested generative modules respectively - without a routed channel partition. Closest in spirit without routing are nested-width models such as MatFormer (Devvrit et al., 2024) and Flextron (Cai et al., 2024), which train a family of sub-networks sharing the same channels so that a smaller model can be extracted from a larger one; the shared substrate is the same idea, but the nesting is static and chosen at extraction time, whereas our partition is re-decided per token and inverted on the way back.

Structured channel mixing and weight sharing. The split-route-merge pattern of our block is structurally related to butterfly and Monarch factorizations (Dao et al., 2019; 2022), which interleave block-diagonal maps with channel permutations, and to channel shuffling in ShuffleNet (Zhang et al., 2018). Two differences are load-bearing. Our interleaving assignment is per token and context-conditioned rather than fixed, which the ablation of Section 6.1 shows to be worth 0.225 bits per byte; and the blocks between split and merge are complete transformer subnetworks rather than linear maps, which the negative controls of the same section show to be necessary - a permutation with nothing to mix reduces to the identity. Cross-layer parameter sharing is itself a known regularizer (ALBERT (Lan et al., 2020); Universal Transformers (Dehghani et al., 2019)), and a tied full-width recurrent control lets us separate the hierarchy of scales from generic weight sharing: sharing alone reaches neither the tree’s quality nor its position on the parameter axis (Table 2, Fig. 10).

3 Method

3.1 Preliminaries

Let $x \in \mathbb{R}^{B \times S \times d}$ be the token stream entering a block, with batch B , sequence length S and model width d . The block is built from transformer decoder units Dec (Vaswani et al., 2017): causal multi-head attention with RoPE (Su et al., 2024), a GELU feed-forward network of hidden scale H , both in pre-norm residual form with RMSNorm. A model stacks n such blocks. Two integers describe the tree inside a block: the branching factor m and the depth D . The configuration we report as our main result uses $m=4$, $D=1$, $H=3$ and $n=3$; the sweeps cover $m \in \{2, 3, 4, 8, 16\}$ and $D \in \{1, \dots, 5\}$.

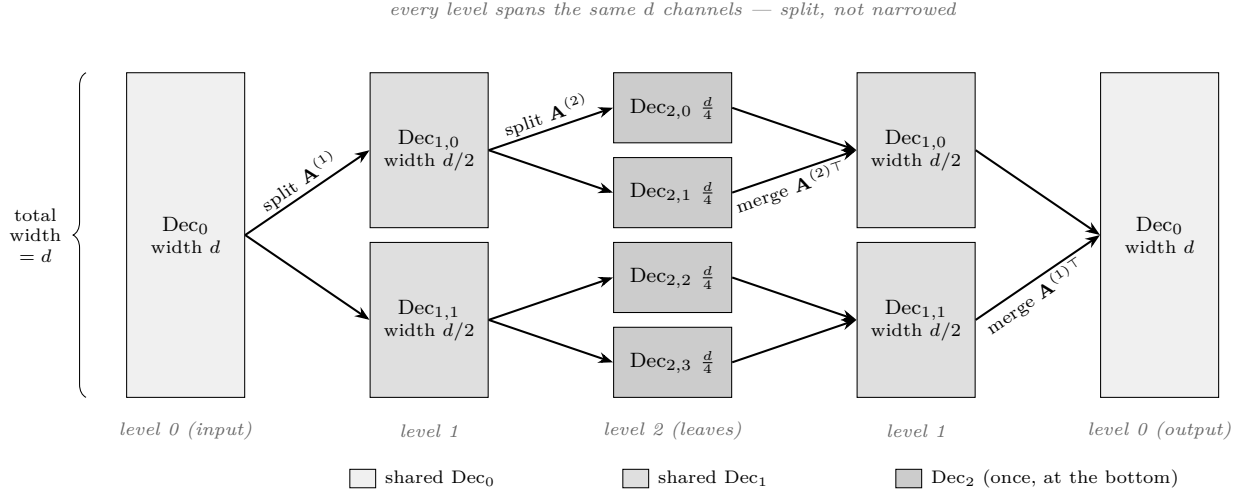


Figure 2: **Fractal tree block** ($D=2$). Down path: channels are routed and split into 2^ℓ subnetworks of width $d/2^\ell$; up path: chunks are merged with the transpose $\mathbf{A}^\top$ of the same per-token assignment. Every level spans the same d channels in total. Gray shades mark shared weights; the figure shows the weight-tied variant, in which one subnetwork processes a level in both directions. In the untied variants each of the $2D+1$ passes carries its own weights. Routing matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}$ are computed per token by the ContextRouter (Fig. 3).

3.2 A self-similar tree over channels

At level $\ell \in \{0, \dots, D\}$ the stream is split along the channel axis into m^ℓ chunks of width $w_\ell = d/m^\ell$, processed in parallel by m^ℓ subnetworks $\{\text{Dec}_{\ell,k}\}$, and concatenated back:

$$x \leftarrow \gamma_\ell \cdot \text{concat}_k[\text{Dec}_{\ell,k}(\text{chunk}_k(x))], \quad \gamma_\ell \in \mathbb{R} \text{ (learned)}. \quad (1)$$

The *down path* applies Eq. (1) for $\ell = 0, 1, \dots, D$ and the *up path* applies it again for $\ell = D-1, \dots, 0$, so a block performs $2D+1$ level passes and traces an hourglass, $1 \rightarrow m \rightarrow \dots \rightarrow m^D \rightarrow \dots \rightarrow m \rightarrow 1$ (Fig. 2). Every level spans the same d channels: the representation is repartitioned at each scale, never narrowed. The scalar γ_ℓ scales the whole output of a level and is initialized to one.

The weights of the up path may be *tied* to the corresponding down-path level - pass k sharing with pass $2D-k$, which makes the up path free in parameters - or *untied*, giving each of the $2D+1$ passes its own weights. Both variants appear in our sweeps; every configuration we headline is untied, because tying costs 0.02-0.03 bits per byte in every pairing we measured.

3.3 Why depth in the tree is cheap

A decoder unit of width w holds $(4+2H)w^2$ weights: four projections for attention and two matrices for a feed-forward network of hidden scale H . Level ℓ holds m^ℓ such units of width d/m^ℓ , so its cost is

$$P_\ell = m^\ell (4+2H) \left(\frac{d}{m^\ell} \right)^2 = \frac{(4+2H)d^2}{m^\ell}, \quad (2)$$

a geometric sequence in ℓ : each level costs a fixed fraction $1/m$ of the one above it. Summing the down path,

$$\sum_{\ell=0}^D P_\ell = (4+2H)d^2 \frac{1-m^{-(D+1)}}{1-m^{-1}} < (4+2H)d^2 \frac{m}{m-1}, \quad (3)$$

so the entire tree below level 0 costs less than one further full-width unit for $m \geq 2$, and less than a third of one for $m=4$ - *independently of D* . Adding a level multiplies the parameter cost by a factor approaching one

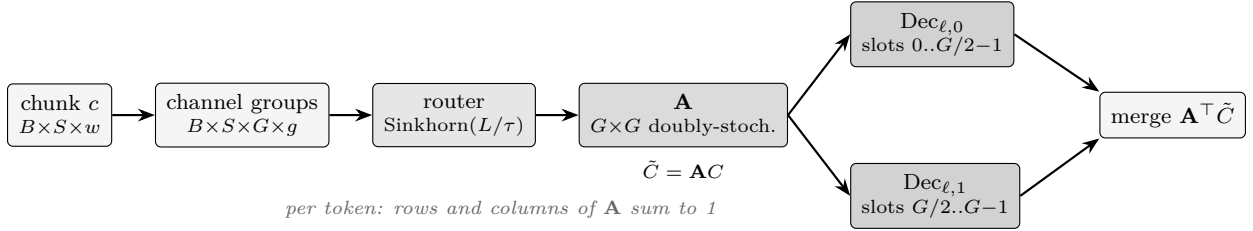


Figure 3: **ContextRouter** at one split. Each token gets its own doubly-stochastic assignment $\mathbf{A}$ (rows: child slots; columns: parent channel groups, Eq. (4)); children process the slot ranges, and the up path merges back by $\mathbf{A}^\top$.

while adding two compute passes. This is the structural claim behind the paper: a dense stack pays linearly in depth, a tree pays a convergent series, and the difference is what the parameter axis of Fig. 1 measures.

3.4 ContextRouter: per-token assignment of channel groups

A hierarchy of narrow branches is only useful if something decides which channels share a branch, and that decision should depend on the token rather than on the channel index. At every split (level $\ell-1 \rightarrow \ell$, parent width $w = d/m^{\ell-1}$) we divide the parent chunk $c \in \mathbb{R}^{B \times S \times w}$ into G *channel groups* of width g , with g equal to the attention head width, so $G = w/g$ (e.g. $G=64$ at the first split of a $d=1024$, $d_{\text{head}}=16$ model). For each token independently the router scores the affinity of every group to every *slot* (Fig. 3):

$$Q = W_q C \in \mathbb{R}^{B \times S \times G \times h}, \quad L = Q E^\top + \Pi \in \mathbb{R}^{B \times S \times G \times G}, \quad \mathbf{A} = \text{Sinkhorn}(L/\tau), \quad (4)$$

where C is the chunk reshaped to groups, h is the router’s hidden width (we use $h=64$), $E \in \mathbb{R}^{G \times h}$ are learned slot embeddings, $\Pi \in \mathbb{R}^{G \times G}$ is a learned identity prior initialized to αI ($\alpha=0.5$), and τ is the routing temperature. Sinkhorn normalization (alternating row and column normalization in log space, $K=5$ iterations) yields a doubly-stochastic matrix whose rows and columns each sum to one *per token*.

Rows of $\mathbf{A}$ index child slots and columns index parent channel groups. Its entries then behave as probabilities: each row is slot i ’s distribution over the groups it may draw from, and by double stochasticity each column is group j ’s distribution over slots. We call $\mathbf{A}$ a *hard permutation* when it is binary - exactly one unit entry per row and per column - in which case $\mathbf{A}^\top \mathbf{A} = I$ and the transpose is its exact inverse; otherwise $\mathbf{A}$ is a *soft permutation*, each group is spread over several slots, and inversion by the transpose is only approximate. In the forward pass we use the hard assignment obtained by solving a linear assignment problem on $\mathbf{A}$, refreshed every 32 optimizer steps, and let gradients flow through the soft $\mathbf{A}$ (straight-through). The routed chunk mixes groups according to $\mathbf{A}$, and consecutive slot ranges feed consecutive children:

$$\tilde{C}_{s,i} = \sum_j A_{s,ij} C_{s,j}, \quad \text{child}_r = \tilde{C}_{:, rG/m : (r+1)G/m}, \quad r = 0, \dots, m-1. \quad (5)$$

3.5 Inverse assignment on the up path

After the children have processed their chunks we restore the canonical channel order with the transpose of the *same* assignment - exact for a permutation, approximate while $\mathbf{A}$ is soft:

$$C'_{s,j} = \sum_i A_{s,ij} Y_{s,i}, \quad (6)$$

where Y are the child outputs regrouped as slots. Since every Dec is residual, $Y \approx \tilde{C} + \Delta$ and hence $C' \approx \mathbf{A}^\top \mathbf{A} C + \mathbf{A}^\top \Delta$; for a near-permutation $\mathbf{A}$ this is $C' \approx C + \mathbf{A}^\top \Delta$, so the stream passes through unmixed and each branch’s update lands back in exactly the channels it was computed from. Channel semantics are therefore stable across levels and, in particular, at the output projection, which reads a fixed channel basis. Untying the weights of the up path does not change this: the merge always reuses the assignment computed at the corresponding split.

Algorithm 1 Forward pass of one FractalPI block (depth D , branching m)

```
1  input stream  $x$ , positions  $pos$ ; state temperature  $\tau$ 
2  for  $\ell = 0, 1, \dots, D$  // down path
3    if  $\ell > 0$ : for each parent chunk  $c$  at level  $\ell-1$ 
4       $\mathbf{A} \leftarrow \text{Sinkhorn}(L/\tau)$  from Eq. (4) // soft, per token
5      if training and step  $\equiv 0 \pmod{32}$ :  $\mathbf{A}^{\text{hard}} \leftarrow \text{LAP}(\mathbf{A})$ ; cache it
6       $\mathbf{A}^* \leftarrow \mathbf{A}^{\text{hard}} + (\mathbf{A} - \text{sg}[\mathbf{A}])$  // hard forward, soft gradient
7       $c \leftarrow \mathbf{A}^* c$ ; store  $\mathbf{A}^*$  for the merge
8       $x \leftarrow \gamma_\ell \cdot \text{concat}_k[\text{Dec}_{\ell,k}(\text{chunk}_k(x))]$  // Eq. (1)
9  for  $\ell = D-1, \dots, 0$  // up path
10    for each parent chunk:  $c \leftarrow (\mathbf{A}^*)^\top c$  // same  $\mathbf{A}^*$ , transposed
11     $x \leftarrow \gamma_\ell \cdot \text{concat}_k[\text{Dec}_{\ell,k}^{\text{up}}(\text{chunk}_k(x))]$ 
12  return  $x$ 
```

At evaluation there are no optimizer steps, so line 5 never fires: the assignment is recomputed from Eq. (4) at the temperature τ reached at the end of training and hardened once per forward pass. Because τ is runtime state and is not stored in a checkpoint, loading a model at a different temperature silently changes its assignments; our evaluation entry points set it from the training schedule. $\text{sg}[\cdot]$ is the stop-gradient operator and LAP solves the linear assignment problem on $\mathbf{A}$.

3.6 What a branch must contain

The routing is only meaningful if something between the split and the merge *mixes channels within a branch*, so that which channels became neighbours has consequences. Two negative controls delimit this. A branch whose only nonlinearity is elementwise leaves the composition $\mathbf{A}^\top \mathbf{A} \approx I$ intact and performs no work: such a model matches an attention-only baseline to within 0.0003 bits per byte while its router keeps spinning. A branch containing a single matrix followed by a nonlinearity is also insufficient, since the nonlinearity then acts in the canonical channel basis; models built that way are unstable and, where they survive, markedly worse. The smallest branch that works is a complete two-matrix feed-forward network inside the branch, which is what Eq. (1) instantiates. A related constraint applies to placement: an elementwise nonlinearity inserted on the residual stream itself, at a level boundary, costs 0.06 bits per byte and is not repaired by removing its mean, so the residual path is kept strictly as an identity highway.

3.7 Training objective

Let $\mathbf{A}^{(b)}$ collect all assignment matrices of block b . The total loss is

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda_{\text{ent}} \mathcal{H}(\mathbf{A}) + \lambda_{\text{bal}} \mathcal{B}(P), \quad (7)$$

with a per-token decisiveness penalty (normalized row entropy, low = decisive)

$$\mathcal{H}(\mathbf{A}) = \mathbb{E}_{s,i} \left[- \sum_j A_{s,ij} \log A_{s,ij} \right] / \log G, \quad (8)$$

and a Switch-style marginal load-balancing term

$$\mathcal{B}(P) = G \sum_j \bar{P}_j^2 - 1 \geq 0, \quad \bar{P}_j = \mathbb{E}_{s,i} [P_{s,ij}], \quad P = \text{softmax}_{\text{row}}(L/\tau), \quad (9)$$

which vanishes exactly when every parent channel group contributes equal routing mass across the batch. Because the Sinkhorn-normalized $\mathbf{A}$ has columns summing to one per token, its marginals are $1/G$ by construction; $\mathcal{B}$ is therefore computed on the row-normalized affinities, before Sinkhorn’s column normalization, where the imbalance is still visible. The two terms pull in opposite directions: per-token decisiveness against marginal uniformity. We use $\lambda_{\text{ent}} = 0.01$ and $\lambda_{\text{bal}} = 0.05$, and ablate both in Fig. 8.

This is not token-level load balancing as in mixture-of-experts. After Sinkhorn every branch receives the same mass for every token by construction, since slots are fixed fractions of the width, so all branches

process all tokens. What varies per token is *which channel subspace* a branch receives: branches specialize in context-selected subspaces, not in subsets of tokens.

Both regularizers are scheduled. Applying the decisiveness penalty from step zero makes the router crystallize into a static balanced permutation - the joint minimum of $\mathcal{H}$ and $\mathcal{B}$ - before the representations contain anything worth routing on, after which softmax saturation cuts the content gradients. We therefore disable λ_{ent} for the first 2% of updates and ramp it linearly to full strength by 30%, and add Gaussian jitter ($\sigma=0.03$) to the router logits during training. The temperature τ is held constant at 0.1 in all runs reported here; the alternative of annealing it from a softer start is examined in Fig. 9.

3.8 Complexity

The router is negligible in both currencies. Its parameters number $gh + Gh + G^2$ per split, about 9k at the first split of a $d=1024$ model, against roughly 2M for the static-permutation variant that learns a full permutation matrix at every split. Its compute runs on $G \times G$ matrices with $G \leq 64$, not on the parent width. The dominant cost of the block is the $2D+1$ level passes themselves: relative to a down-only variant, the up path repeats levels $D-1, \dots, 0$, which multiplies projection and feed-forward work by $(2 \sum_{\ell=0}^{D-1} m^{-\ell} + m^{-D}) / \sum_{\ell=0}^D m^{-\ell}$ and attention, whose cost per level pass is constant, by $(2D+1)/(D+1)$. Measured parameter and FLOP totals for every evaluated configuration are reported alongside its results (Tables 1-2, Fig. 1).

4 Experimental setup

Data. All models are trained on a prefix of FineWeb (Penedo et al., 2024), configuration `sample-10BT`, consumed in corpus order without shuffling, so that a token budget always selects the same prefix and every repetition regime revisits exactly the same subset. Documents are tokenized with the GPT-2 byte-pair encoding through `tiktoken`, giving a vocabulary of 50,258 symbols (50,257 plus one padding token), and packed into full sequences of length 1024 without padding: long documents are cut into windows and short ones are concatenated through an end-of-text separator. The held-out set is the tail of the much larger `sample-100BT` configuration, the last 2048 documents; since training consumes a small fraction of the head of that corpus, the two are disjoint by construction, and we verify the margin numerically for every budget we report. Out-of-domain evaluation uses WikiText-103 (Merity et al., 2017), PG-19 (Rae et al., 2020) and LAMBADA (Paperno et al., 2016).

Models and baselines. Both architectures share $d=1024$, an attention head width of 16, a sequence length of 1024 and tied input and output embeddings, which at this vocabulary account for 51.5M parameters - roughly 40% of every model in this study, and the reason parameter comparisons in this paper are reported on the non-embedding count. The baseline is a dense GPT-2-style decoder using the *same* attention implementation (RoPE, fused causal scaled-dot-product attention) and the same GELU feed-forward block as the fractal subnetworks; the two families differ only in how blocks are structured. We train the baseline at depths two to ten layers, which gives the scaling law used throughout, and in with- and without-dropout variants; FractalPI is trained without dropout in every run reported here.

Optimization. Matrices are updated by Muon (Jordan et al., 2024) at a fixed rate of 0.02, everything else - embeddings, output head, normalization scales, gains - by AdamW ($\beta = (0.9, 0.95)$, weight decay 0.01) at a base rate of 10^{-4} , and the router parameters by a separate AdamW group at 0.01 without weight decay. Both main groups follow one cosine decay to a tenth of their peak, computed over the entire budget rather than per epoch. A learning-rate multiplier $\times N$, used throughout the paper, scales the AdamW group only; the Muon and router rates are held fixed, so the sweeps of Fig. 4 vary a single quantity. The effective batch is 262k tokens per update (micro-batch 2, sequence 1024, 128 gradient-accumulation steps). All comparison runs are in fp32 on a single NVIDIA L40S with 48 GB; a bf16 copy of each checkpoint is written for inspection but not used for the numbers reported here.

Budgets. The single-pass budget is 300M tokens, chosen so that a full sweep fits the available compute while remaining large enough for the dense baselines to separate cleanly by depth. The data-constrained

regime keeps the compute fixed and shrinks the data: four epochs over the same 75M-token subset. Unless a row states otherwise, results are single runs at seed 23; where three seeds are available (the dense anchor and the fractal configuration of Table 1) we report the mean and standard deviation.

Evaluation. We report bits per byte, which unlike perplexity per token is comparable across tokenizations. Each evaluation set is scored with rolling windows of 1024 tokens at stride 512, over up to 2048 documents and at most 2M scored tokens; only the second half of each window is scored, so every token is predicted from at least 512 tokens of context. The *train/validation gap* is the held-out FineWeb figure minus the training-set figure measured on the same checkpoint, and is meaningful only in-distribution. LAMBADA is evaluated as teacher-forced greedy prediction of the final word, which is stricter than the usual last-token accuracy and therefore not comparable with published numbers; we use it only to compare models within this paper. The zero-shot understanding suite comprises HellaSwag, PIQA, SciQ, ARC-Easy, Winogrande and RTE, scored by length-normalized likelihood over 1000 examples per task.

One detail matters for reproducibility. The routing temperature τ is runtime state and is not stored in a checkpoint, so an evaluation must set it to the value the schedule had reached at the end of training; loading a model at a different temperature changes its assignments and silently changes its loss. Our evaluation entry points set it automatically.

5 Results

5.1 Two families in the cost-quality plane

Our central measurement compares families rather than models (Fig. 1). The dense baselines, trained at depths two to ten under the protocol of Section 4, trace a power law in non-embedding parameters,

$$L(N) = 1.553 N^{-0.0443}, \quad R^2 = 0.978, \quad (10)$$

with N in millions and L in bits per byte. The fit is tight enough that the dense family can be treated as a curve rather than a set of points, which is what makes a family-versus-family comparison possible: the question is no longer whether some fractal beats some baseline, but whether the fractal cloud sits below the line.

It does, on parameters. Taking each dense depth in turn and asking for the smallest fractal in our sweep that reaches its quality, a fractal matches the three-layer baseline with 22% fewer non-embedding parameters (29.4M against 37.8M), the four-layer one with 9% fewer (45.7M against 50.4M), and the two-layer one with 6% fewer while also improving quality by 0.03 bits per byte; across the four dense depths in this region the direct saving ranges from 3% to 22%. Fitting the Pareto frontier of the fractal points gives $L \propto N^{-0.0359}$, which places the saving at equal quality between 5% and 40% over the 25-63M region where it is validated, largest at the small end.

It does not, on compute. The same dense ladder fits $L \propto C^{-0.0769}$ in forward FLOPs per token and the fractal frontier $L \propto C^{-0.0876}$; the two lines cross inside the measured range, and the direct matching test gives -0.6% , $+2.3\%$ and $+2.5\%$ against the three-, four- and five-layer baselines. Within measurement error the families are equally efficient in arithmetic. The advantage is therefore specific: FractalPI is a better container for parameters, not a cheaper way to compute.

Two features of the picture bound the claim. The interior of the cost range is populated exclusively by fractal configurations - the dense family contributes points, the fractal family contributes a frontier - which is the practical content of a richer design space. But the fractal frontier is *flatter* than the dense law (0.0359, bootstrap 95% CI [0.0352, 0.0403], against 0.0443), so the advantage narrows as models grow.

Where the advantage ends. Because the exponents differ, the interesting question is not whether the curves converge but where. We therefore extended the dense ladder to eight and ten layers and trained two fractals matched to them in non-embedding parameters to within 0.1% - the first configurations in this study larger than the dense anchor. The dense model wins both pairs, and by a growing margin: 1.2679

Table 1: **Single-pass language-modeling results** on the evaluation suite of Section 4 (up to 2048 documents and 2M scored tokens per set, rolling windows of 1024 tokens at stride 512). BPB = bits per byte of the evaluated text; FineWeb held-out is in-distribution, the remaining sets are out-of-domain; LAMBADA reports final-word accuracy (teacher-forced greedy). All rows: 300MT, no dropout, tuned learning rate ($\times 30$ for both architectures). The train/val gap is the FineWeb held-out BPB minus the final train BPB, both at the final checkpoint. (Nr) = $\text{mean} \pm \text{s.d.}$ over N independent seeds; remaining rows are single-seed. Top block: dense GPT-2 with 6 layers and the fractal configuration of three stacked untied blocks (branching $m=4$, depth $D=1$, hidden scale $H=3$). Bottom block: tree-depth sweep at branching $m=2$, a single fractal block with all $2D+1$ level passes untied.

Model	Params	FLOPs/tok	BPB $\downarrow$			LAMBADA train/val	
			FineWeb	WikiText	PG-19	acc $\uparrow$	gap $\downarrow$
GPT-2 6L, no dropout (3r)	127.0M	0.280 G	1.279$\pm$.0004	1.343	1.650	14.2%	+0.015
FractalPI 3 blocks, $m=4$, $D=1$, $H=3$, untied (3r)	122.4M	0.285 G	1.282 $\pm$.0017	1.345	1.650	14.7%	+0.014
<i>Tree-depth sweep ($m=2$, single fractal block, untied, seed 23)</i>							
FractalPI depth $D=2$	92.4M	0.207 G	1.303	1.367	1.675	12.7%	+0.015
FractalPI depth $D=3$	97.2M	0.226 G	1.297	1.359	1.660	12.6%	+0.016
FractalPI depth $D=4$	99.6M	0.239 G	1.293	1.364	1.660	13.1%	+0.015
FractalPI depth $D=5$	100.8M	0.250 G	1.297	1.361	1.668	13.0%	+0.015

against 1.2730 at 100.8M non-embedding parameters, and 1.2583 against 1.2718 at 125.9M. The crossing therefore falls just above our earlier anchor, and the saving reported above is bounded to roughly 25-63M non-embedding parameters. Two observations sharpen this. First, the two new fractals land 0.012 and 0.022 bits per byte *above* the frontier fitted to the smaller configurations, so that fit does not extrapolate: the family loses ground faster than its own trend suggests. Second, the larger baselines flatten the dense law itself: fitted to the five depths up to six layers it reads $N^{-0.0512}$, and to all seven $N^{-0.0443}$, a reminder that a scaling law measured over one decade is a local object.

5.2 The parity point

Table 1 reports the top of that range. Three stacked untied fractal blocks with branching $m=4$, depth $D=1$ and hidden scale $H=3$ reach 1.282 ± 0.002 bits per byte over three seeds, against 1.279 ± 0.0004 for the six-layer dense baseline over three seeds - a difference of 0.0026, about one and a half standard deviations of our own seed spread. We read this as a statistical tie, not a win. The fractal carries 4% fewer total and 6% fewer non-embedding parameters at essentially equal FLOPs (0.285 against 0.280 GFLOPs per token), matches the baseline on WikiText and PG-19, and is ahead on final-word accuracy on average (14.7% against 14.2%, a mean paired difference of +0.4 points over three seeds, positive in two pairs of the three). With three seeds this is a direction, not a significant effect, and we report it as such. The train/validation gap is +0.014 against +0.015: at a single pass over this much data neither architecture is under overfitting pressure, so that column becomes informative only under repetition (Section 5.4).

The lower block of Table 1 varies tree depth at fixed branching $m=2$ with every level pass untied. Quality improves from 1.303 at $D=2$ to 1.293 at $D=4$ and then stops, while the parameter count rises only from 92.4M to 99.6M - the geometric series of Eq. (3) in practice, each level costing a quarter of the previous increment. The saturation is consistent with branches becoming too narrow to carry a useful subnetwork: at $d=1024$ and $m=2$ the fifth level operates on 32 channels, and a separate configuration at $m=4$, $D=4$, where branches reach four channels, diverges outright.

5.3 The parity is not an artifact of tuning

A comparison at one learning rate may favour whichever architecture happened to be tuned for it. Figure 4 answers this with paired sweeps: both architectures, dropout removed from both, swept over two decades under an identical schedule. The curves coincide from $\times 1$ to $\times 30$ to within 0.005 bits per byte - inside the seed spread of the fractal - and both reach their optimum at $\times 30$. They differ in exactly one place, at $\times 100$,

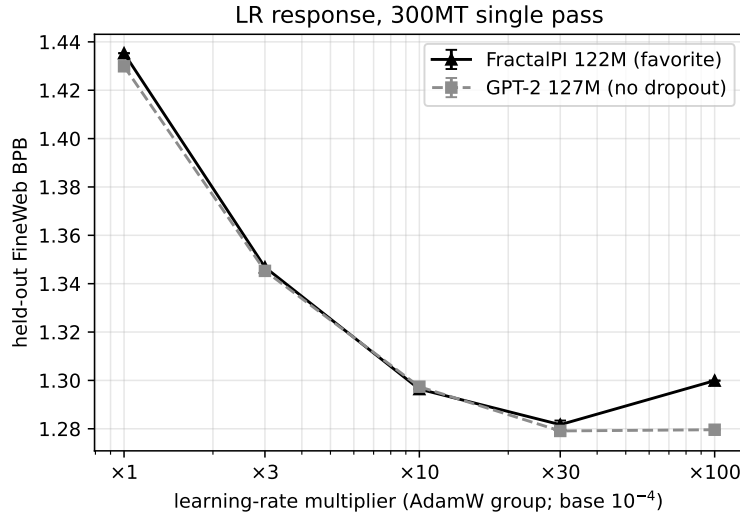


Figure 4: **Learning-rate response at 300MT** (held-out FineWeb BPB; error bars: s.d. over seeds where more than one was run). The multiplier scales the AdamW parameter group only, from a base rate of 10^{-4} ; the Muon and router rates are held fixed.

where the dense optimum is flat ($1.279 \rightarrow 1.280$) while the fractal rebounds by 0.020. The parity is therefore a property of the two architectures at their respective optima rather than of a rate that suits one of them, and the single asymmetry runs against us: our optimum is narrower, and overshooting the learning rate costs us more than it costs the baseline.

A second question is whether a difference invisible in held-out loss shows up in what the models can do. On a six-task zero-shot suite scored by length-normalized likelihood, evaluated on the matched pair of Section 5.2, the mean difference in accuracy is -0.001 ± 0.011 (Fig. 5); the largest single-task gaps are 1.6 points on HellaSwag in favour of the baseline and 1.5 points on ARC-Easy in favour of the fractal. Winogrande and RTE sit at chance for both models, as expected at this scale, and we draw no conclusion from them. This is a bounded statement - the suite is small and the test has a power of roughly three points - not a claim of equivalence.

5.4 Repeated data

Keeping compute fixed and shrinking the data to four passes over the same 75M tokens (Table 2, Fig. 6) separates the models along two axes at once. In final quality the ordering follows the single-pass one: the fractal and the dropout-free dense baseline end together at 1.339 and 1.340, the dropout arm 0.018 behind them, and the two smaller structural controls further back. None of the five turns upward within four passes.

The gaps tell a different story from the losses. The fractal holds a smaller train/validation gap than the dropout-free baseline ($+0.158$ against $+0.176$), and does so at every epoch rather than only at the end. But ordinary dropout, which costs 0.018 bits per byte of quality, buys a much larger reduction ($+0.104$), and weight tying buys a comparable one ($+0.114$) at a larger quality cost. The honest summary is that the self-similar structure supplies a small part of what an explicit regularizer supplies, at no cost in quality, and that dropout remains the stronger instrument when a small gap is the objective. The param-matched dense two-layer control ends both worst in quality and with a gap comparable to ours, so a smaller parameter count does not by itself buy a smaller gap.

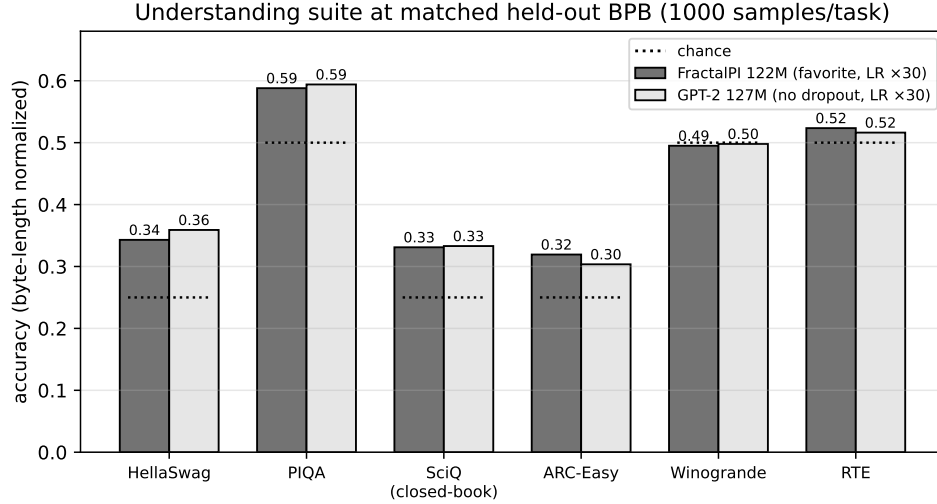


Figure 5: **Understanding suite at matched held-out BPB** (byte-length-normalized accuracy, 1000 samples per task; dotted marks: chance level for each task).

Table 2: **Data repetition**: four epochs over the same 75M-token subset, i.e. 300MT of compute, all at the tuned rate $\times 30$ and without dropout unless noted; final checkpoint, per-epoch curves in Fig. 6. Columns and evaluation protocol as in Table 1.

Model	Params	FLOPs/tok	BPB ↓			LAMBADA train/val	
			FineWeb	WikiText	PG-19	acc ↑	gap ↓
FractalPI (3 blocks, $m=4$, $D=1$, $H=3$)	122.4M	0.285 G	1.339	1.422	1.715	12.7%	+0.158
GPT-2 6L, no dropout	127.0M	0.280 G	1.340	1.424	1.722	11.8%	+0.176
GPT-2 6L, dropout 0.1	127.0M	0.280 G	1.358	1.429	1.729	15.7%	+0.104
tied recurrent (ALBERT-style)	64.1M	0.250 G	1.387	1.500	1.779	7.7%	+0.114
dense 2-layer	76.7M	0.162 G	1.401	1.530	1.806	5.9%	+0.149

5.5 The cost side

The parameter saving is not free in time. Measured on a single L40S with identical batch and sequence length within each pair, a training step of the configuration of Table 1 takes $1.37\times$, $1.74\times$ and $1.86\times$ the time of the equally wide dense baseline at $d = 1024$, 2048 and 4096 in fp32, and $1.54\times$ and $2.51\times$ at the first two widths in bf16 (Fig. 7). The ratio grows with width and is worse in the precision one would use at scale. The cause is granularity rather than arithmetic: at FLOP parity the block issues $2D+1$ level passes of small grouped matrix products plus a router per split, and these are memory- and launch-bound where a dense stack issues few large matrix multiplications that saturate the tensor cores.

6 Analysis

6.1 What the design rests on

Table 3 and Fig. 8 remove one component at a time. The result is unusually lopsided. Replacing the context-conditioned assignment with a static learned permutation - keeping every matrix, every level and every pass, and changing only whether the assignment depends on the token - costs 0.372 bits per byte at the base rate and 0.225 at the tuned rate, and takes final-word accuracy to zero. Every other component we removed costs between 0.003 and 0.015: the entropy penalty, the balance penalty and the identity prior are each near-free individually, and dropping the up path entirely costs 0.015 while saving 28% of the FLOPs, which makes it an efficiency variant rather than an ablation.

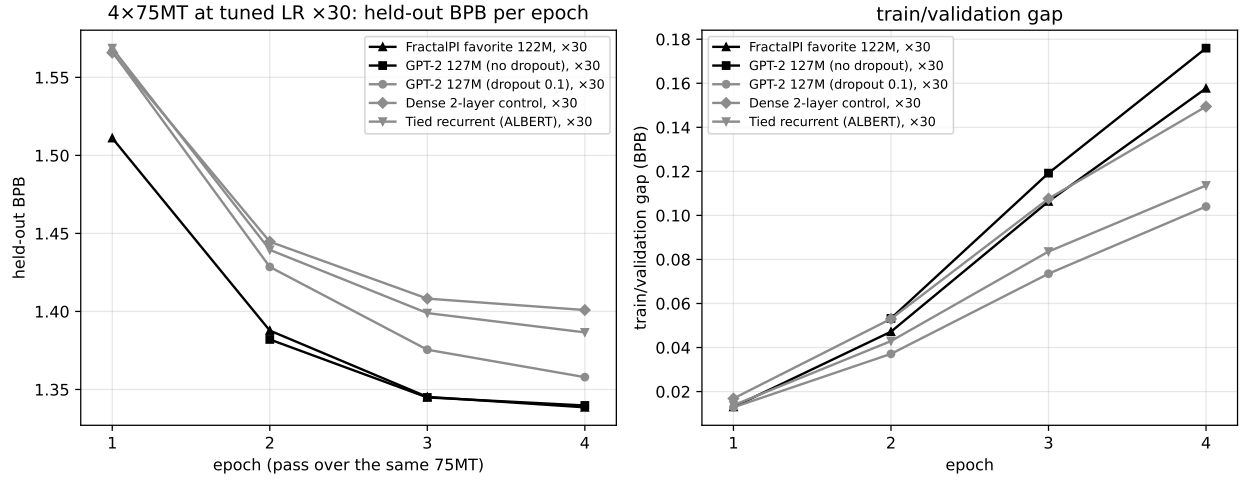


Figure 6: **Data-constrained repetition** (4 epochs over the same 75MT, tuned rate $\times 30$, no dropout unless stated). Held-out FineWeb BPB per epoch (left) and the train/validation gap (right).

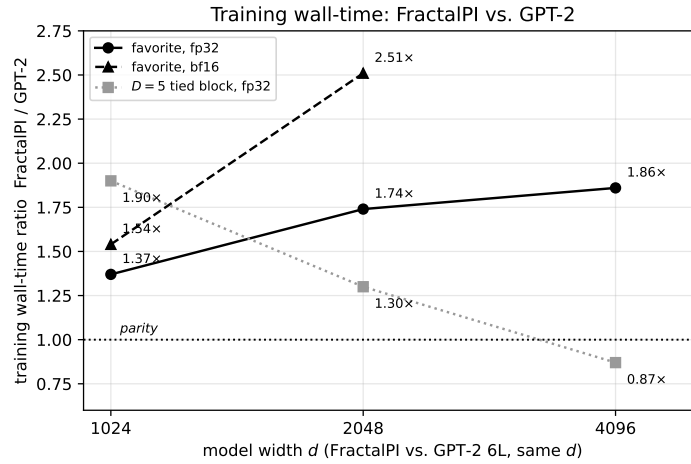


Figure 7: **Training wall-time ratio against model width** (FractalPI over GPT-2 6L at the same d , identical batch and sequence length within each pair, single L40S, forward+backward, 15 steps after 4 warm-up steps). Solid and dashed black: the headline configuration in fp32 and bf16; gray dotted: a $D=5$ weight-tied block in fp32.

The routing is thus the component the architecture rests on, and two negative controls say what kind of work it does. In the first, the router is present and active but the branches contain nothing that mixes channels within a branch; the assignment then composes with its own transpose to the identity and the model matches an attention-only baseline to within 0.0003 bits per byte, while its router keeps producing per-token assignments with a higher spread than any working model we trained. Routing without a mixer is motion without work. In the second, each branch contains a single matrix followed by a nonlinearity rather than a complete feed-forward network; the nonlinearity then acts in the canonical channel basis, and models built this way are unstable in deep stacks and, where they survive, several tenths of a bit worse. Together the two controls locate the mechanism: routing has value only in combination with in-branch mixing, and the gains reported in Section 5 are attributable to the pair, never to the permutation alone.

A third control concerns placement rather than content. Inserting an elementwise nonlinearity on the residual stream immediately after a split costs 0.061 bits per byte, and subtracting its mean - which removes the

Table 3: **Ablations** of the FractalPI tree at the 300MT budget; each row removes or alters one component of the full model. Absolute LAMBADA accuracies are far below those of Table 1 because these runs use an earlier configuration at the base learning rate; only differences within this table are meaningful.

Model	Params	FLOPs/tok	Tokens	BPB ↓			LAMBADA train/val	
				FineWeb	WikiText	PG-19	acc ↑	gap ↓
FractalPI tree (full model)	76.4M	0.25 G	300MT	1.521	1.659	1.868	1.4%	+0.010
– up path (down-only)	76.4M	0.18 G	300MT	1.536	1.677	1.885	1.3%	+0.011
– ContextRouter (static perm.)	78.3M	0.26 G	300MT	1.893	2.200	2.431	0.0%	+0.002
– balance $\mathcal{B}(P)$ ($\lambda_{\text{bal}}=0$)	76.4M	0.25 G	300MT	1.525	1.656	1.871	1.1%	+0.011
– entropy $\mathcal{H}(\mathbf{A})$ ($\lambda_{\text{ent}}=0$)	76.4M	0.25 G	300MT	1.524	1.659	1.857	1.2%	+0.010
– identity prior ($\Pi=0$)	76.4M	0.25 G	300MT	1.528	1.665	1.871	1.5%	+0.010

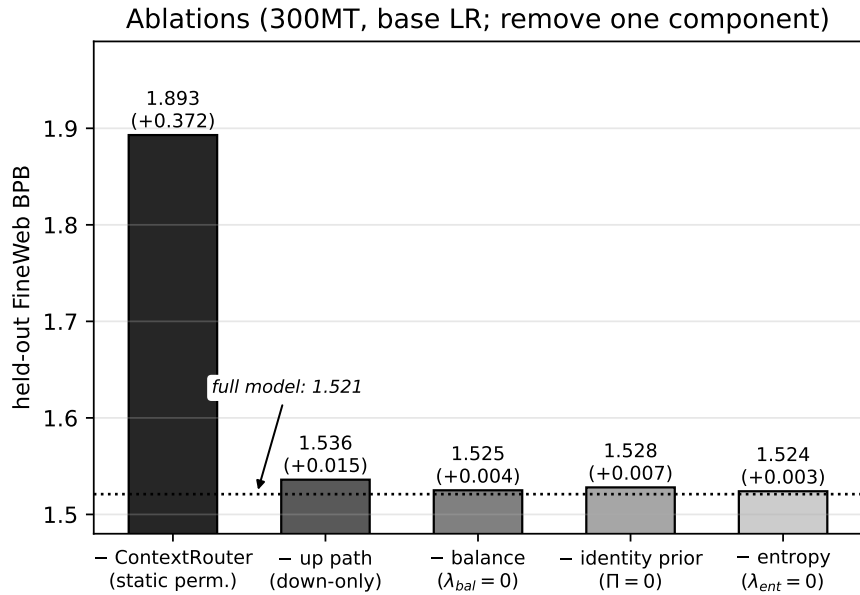


Figure 8: **Ablations** (300MT, base rate; held-out FineWeb BPB; dotted line: the full model). Each bar removes or alters one component of the tree.

positive drift that such a nonlinearity injects, verified to machine precision - recovers 0.0003 of that. The cost is therefore not the drift but the loss of an identity path across level boundaries. We keep the residual stream strictly additive for that reason.

6.2 The routing temperature

The Sinkhorn temperature τ controls how sharply the assignment commits. Figure 9 sweeps it at two budgets. At 75M tokens the optimum is a broad plateau between 0.02 and 0.1 and quality degrades quickly above it, while at 300M tokens the curve is unimodal with a minimum at 0.1; the standard recipe of annealing from a soft start to a sharp one is worse than holding the sharp value throughout at both budgets. The direction of the shift is consistent with the router crystallizing on a step-driven rather than budget-driven schedule: a longer run needs a softer temperature to keep the assignment adapting alongside the representations it routes. We hold τ constant at 0.1 in every run reported in this paper, which is the optimum at our main budget, and note that this is a hyperparameter whose optimum moves with training length - a practical cost of the design.

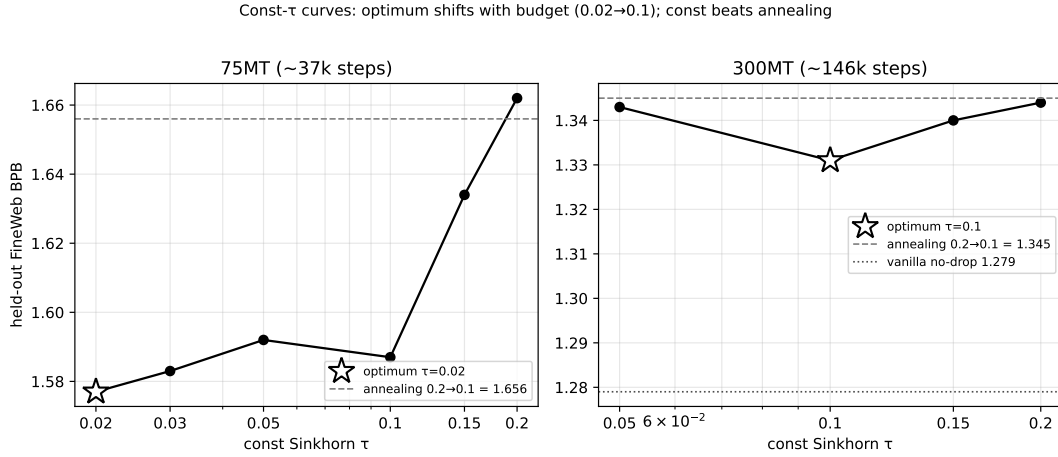


Figure 9: **Constant versus annealed Sinkhorn temperature.** Held-out BPB against constant τ at 75MT (left) and 300MT (right); the dashed line marks the annealed 0.2 $\rightarrow$ 0.1 schedule. Sweep points are single runs; the $\tau=0.1$ point at 300MT is a mean over three seeds.

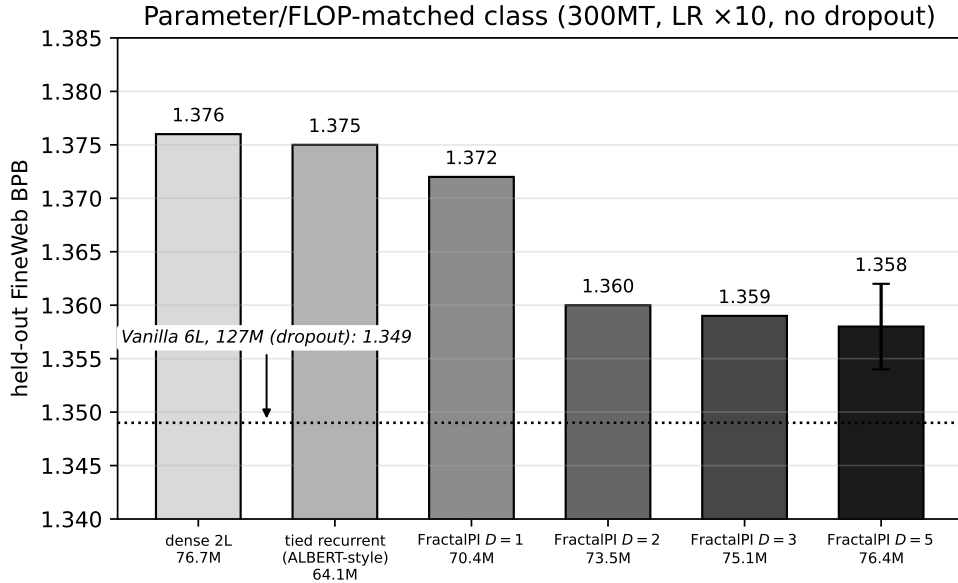


Figure 10: **Parameter- and FLOP-matched class** (300MT, LR $\times 10$, no dropout; held-out FineWeb BPB): param-matched dense, ALBERT-style tied recurrence, and fractal trees of depth $D=1$ to $D=5$, all in the 64-77M range. The dotted line marks the 1.7 $\times$ larger dense baseline at the same rate.

6.3 Structural alternatives at matched size

Figure 10 places the tree against the obvious structural alternatives in a narrow size band (64-77M, tuned rate $\times 10$). A param-matched two-layer dense model reaches 1.376 and an ALBERT-style tied-recurrent model 1.375, against 1.358 for the tree at $D=5$; within the tree family, quality improves from $D=1$ to $D=2$ and then flattens at this width. Two readings are available and we prefer the weaker one: the tree is the best of these structures in this band, but the band is narrow and the alternatives are single runs, so this figure supports the frontier of Section 5.1 rather than standing on its own.

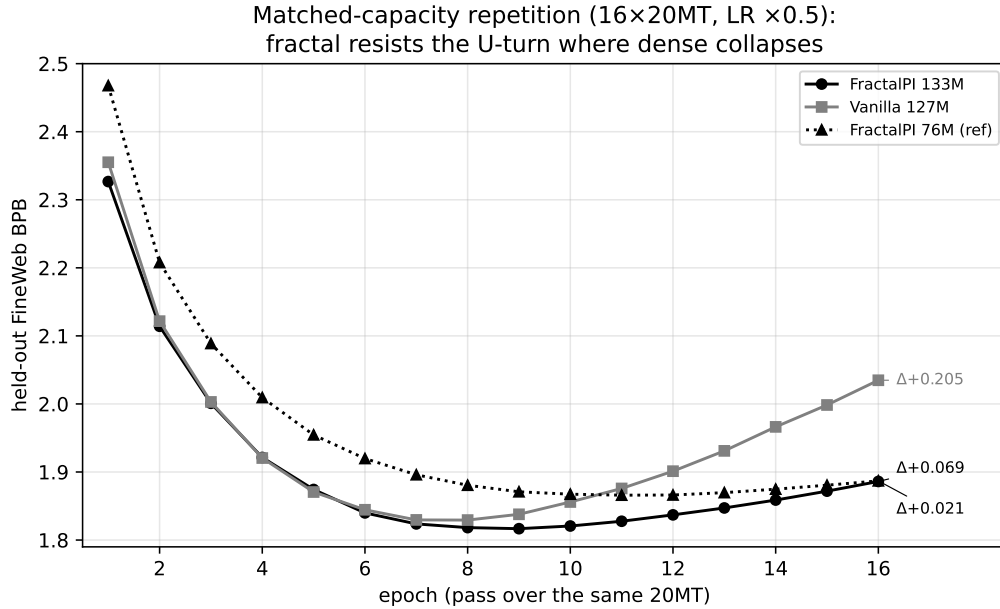


Figure 11: **Matched-capacity repetition** (16×20MT, conservative rate ×0.5 held identical for both models). Held-out FineWeb BPB per epoch for a 133M fractal ($d=1536$), a 127M dense baseline, and a 76M fractal for reference.

6.4 Extreme repetition, and a claim we do not make

Pushing repetition further - sixteen passes over a 20M-token subset - produces the most-cited behaviour of this architecture in our earlier experiments and the one we now report with the greatest caution. At matched capacity and a deliberately conservative rate, a 133M tied fractal and the 127M dense baseline descend to nearly the same minimum and then separate: the dense model loses 0.205 bits per byte from its own optimum by the sixteenth pass while the fractal loses 0.069 (Figs. 11 and 12). Read narrowly, this says that at a conservative rate and equal parameter count the dense model’s held-out loss collapses where the fractal’s drifts.

Read as a property of the architecture, it does not survive. The protection is a property of the learning rate, not of the tree: at its own tuned rate the configuration we headline, run in the same regime, memorizes the repeated subset almost perfectly (training loss 1.03 bits per byte) while its held-out loss climbs to 2.38 - a deeper collapse than any dense run we measured. The same reversal appears within the earlier family, where the tuned-rate fractal degrades by 0.235 against 0.026 at a conservative rate. We therefore make no claim of repetition resistance. What Section 5.4 reports instead is the moderate regime, four passes at the tuned rate, where the architectures end together and the gap difference is small and in our favour.

7 Discussion

What the measurements amount to. Three currencies, three verdicts. On non-embedding parameters the tree is a better container than depth over a bounded range: matching a dense baseline between 25 and 63M non-embedding parameters costs us 3-22% fewer of them, most at the small end, and above that range the ordering reverses. On arithmetic the two families are indistinguishable within measurement error. On wall-clock time the tree is worse by $1.37\text{--}1.86\times$ and the ratio grows with width. A practitioner should read this as a conditional recommendation rather than a replacement: when the binding constraint is memory - optimizer state during training, weights during serving, or a device that a dense model of the required quality does not fit on - the tree buys real room, and when the binding constraint is time it does not.

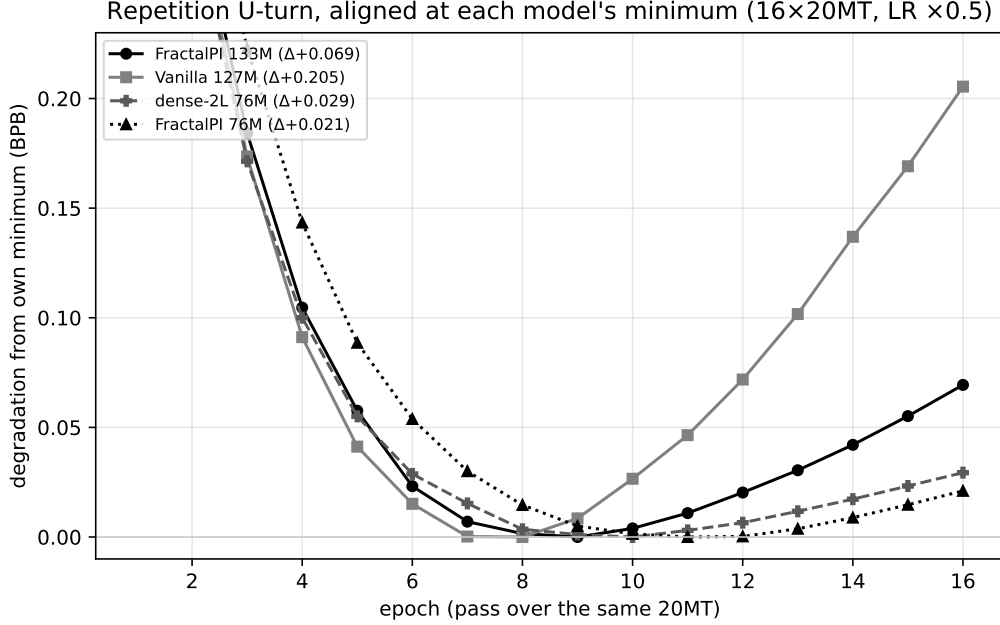


Figure 12: **Repetition curves aligned at each model’s own minimum** ($16 \times 20\text{MT}$, conservative rate): degradation from that minimum against epoch, for the models of Fig. 11 and the param-matched dense-2L control.

Where the saving comes from, and why it shrinks. The mechanism is visible in Eq. (3): levels of the tree form a geometric series, so a dense stack pays linearly for depth while a tree pays a convergent sum. That is why the interior of the cost range in Fig. 1 belongs to the fractal family - between two dense depths there is nothing to build, whereas the tree offers a continuum of branching factors, depths, hidden scales and tying patterns that land in between. It also explains the limit of the effect. The saving is an *offset*: our frontier exponent (0.0359) is smaller than the dense one (0.0443), so the two curves converge as models grow. That is no longer an extrapolation - we measured where they cross. Above the anchor the ordering reverses and the reversal widens with size, and the fractals trained there fall short of their own fitted frontier, which says the family loses ground faster than its trend. Whether the crossing point moves with width - the geometric series becomes more favourable as d grows and the tree can afford more levels - is the obvious next experiment and one we have not run.

The same caution applies to the other axis, and more strongly, because there we have no gradient at all. Everything above is measured at a single token budget, at which every model is trained on fewer than four tokens per parameter and is therefore far from a compute-optimal allocation. A block that uses parameters efficiently is most valuable exactly when parameters are the scarce resource, so the regime we measure is the one that flatters it; whether the offset survives when data rather than capacity becomes binding is untested. Neither extrapolation - along width or along tokens - is settled by anything in this paper, and the two could move the crossing point in opposite directions.

Routing is the mechanism, not the decoration. The single largest ablation cost in this study, by an order of magnitude over the next one, is the replacement of a per-token context-conditioned assignment with a static one. Read together with the two negative controls of Section 6.1, this places the architecture in a specific position relative to conditional computation: unlike a mixture of experts, every branch processes every token and the routing decides *which channel subspace* a branch receives rather than which tokens it sees, so there is no capacity that sits idle and no load-balancing problem in the usual sense. The price is a small combinatorial machine - a Sinkhorn normalization and a hard assignment per split - whose parameters are negligible but whose kernels are not, which is precisely the wall-clock cost discussed above.

8 Limitations

Scale. Every result here is at $d=1024$ and 300M training tokens, two to three orders of magnitude below the models these questions ultimately concern. The dense scaling law we fit is our own, measured under our protocol over a single decade of non-embedding parameters, and both curves should be read as local. Our wall-clock measurements reach $d=4096$; nothing here establishes the behaviour of either family beyond that.

Token budget. The budget is small not only in absolute terms but relative to the models trained on it. At 300MT our models see between 1.7 and 3.9 tokens per parameter - the largest, a ten-layer dense baseline at 177M parameters, sees 1.7 - against the roughly 20 tokens per parameter of a compute-optimal allocation. Every model in this study is therefore substantially under-trained, which is the likely reason our fitted exponent (0.0443) falls below the 0.076 reported at larger scale (Kaplan et al., 2020): a data-limited regime flattens the parameter law, because additional capacity cannot be used. The consequence for our main claim is that it is measured at a single point on the data axis. Whether a parameter-efficient block retains its advantage when parameters are no longer the scarce resource is not something we can answer here; settling it for the largest model of this study would require about 3.5B tokens, twelve times the budget used throughout. The one run we have near that regime - a 76M fractal trained on 2B tokens, 26 tokens per parameter - reaches 1.307 bits per byte without sign of degradation, but it has no dense counterpart and therefore supports no comparison.

Statistical strength. Two configurations have three seeds: the dense anchor (± 0.0004) and the fractal of Table 1 (± 0.0017). Every other point in Fig. 1, every ablation, every temperature in Fig. 9 and every repetition row is a single run. Differences below roughly 0.005 bits per byte between single runs should not be read as ordering, which affects in particular the depth sweep in Table 1 ($D=3$ to $D=5$ span 0.004) and the structural comparison of Fig. 10. The frontier exponent inherits that noise, and we quantify it: resampling every fractal point with $\sigma=0.002$ and refitting gives a 95% interval of $[0.0352, 0.0403]$ around the point estimate 0.0359, which excludes the dense exponent 0.0443. The interval is asymmetric because the frontier is a lower envelope: resampling changes which configurations define it, and a point can only join the envelope by falling below the current one. The difference in slopes is therefore not an artifact of single runs, though the individual per-point savings, which are minima over a sparse grid, move by up to three points under the same resampling.

The advantage is bounded, and we measured the boundary. The saving holds over roughly 25-63M non-embedding parameters and ends there: at 100.8M and 125.9M, where we trained parameter-matched pairs, the dense model wins by 0.005 and 0.014 bits per byte. What we cannot say is how that boundary moves. Both families were measured at a single width, and the geometric series of Eq. (3) becomes more favourable as d grows, so the crossing may sit further right at $d=2048$ than at $d=1024$; a single parameter-matched pair at a larger width would test the direction. We report the boundary we measured, not an asymptotic claim.

Repetition. We withdraw the claim that self-similarity resists repeated data (Section 6.4). At a conservative learning rate a tied fractal degrades three times less than a dense model of equal capacity over sixteen passes, but at its tuned rate our own headline configuration degrades more than any dense run we measured. In the moderate regime of Section 5.4 the two architectures end together with a small gap difference in our favour, and ordinary dropout achieves more gap reduction than our structure does, at a cost of 0.018 bits per byte in quality. Dropout combined with the tree is a control we have not run.

Tuning tolerance. The learning-rate optimum is narrower than the baseline’s: both peak at the same rate, but overshooting by a further factor of three costs the fractal 0.020 bits per byte where it costs the dense model nothing (Fig. 4). The routing temperature adds a hyperparameter whose optimum moves with training length. A practitioner adopting this block inherits both.

Data order. Training consumes the corpus in its stored order without shuffling, so that a token budget always selects the same prefix and repetition regimes revisit exactly the same subset. This makes runs

comparable to one another but couples every result to one particular slice of FineWeb; we did not run a shuffled control, so we cannot separate the architecture’s behaviour from that slice.

Evaluation. LAMBADA is scored here as teacher-forced greedy prediction of the whole final word, which is stricter than the standard last-token metric and not comparable with published numbers; we use it only within this paper. The zero-shot understanding suite has a power of roughly three accuracy points, so its result is an absence of a detectable difference rather than evidence of equivalence.

A All runs

Table 4 lists every single-pass run entering Fig. 1: all were trained on 300M tokens at the tuned rate $\times 30$ without dropout, in fp32, at seed 23 unless a seed is named in the configuration. Parameters and FLOPs are measured from the constructed model rather than estimated; the non-embedding column is the one used for every parameter comparison in the paper. The same table is available as a CSV file in the code repository ([analysis_eval/runs_300mt.csv](#)). One run of a one-layer dense baseline diverged at this learning rate and is excluded here and from every fit.

Table 4: Every 300MT run: configuration, family, parameters, compute and results.

Configuration	Family	Params [M]	Non-emb. [M]	GFLOPs /token	FineWeb BPB	LAMBADA %	gap
vanilla_2l_hs4_lrx30	dense	76.7	25.2	0.162	1.3535	6.9	+0.0159
vanilla_attnonly_lrx30	dense	76.7	25.2	0.178	1.3591	10.4	+0.0155
vanilla_nodrop_3l_lrx30	dense	89.2	37.8	0.191	1.3191	11.2	+0.0157
vanilla_nodrop_4l_lrx30	dense	101.8	50.4	0.220	1.3001	12.7	+0.0149
vanilla_nodrop_5l_lrx30	dense	114.4	63.0	0.250	1.2887	12.6	+0.0151
control_vanilla_nodrop_lrx30	dense	127.0	75.6	0.279	1.2787	14.9	+0.0155
vanilla_nodrop_lrx30_seed24	dense	127.0	75.6	0.279	1.2795	14.0	+0.0152
vanilla_nodrop_lrx30_seed25	dense	127.0	75.6	0.279	1.2791	13.8	+0.0152
vanilla_nodrop_8l_lrx30	dense	152.2	100.8	0.338	1.2679	15.2	+0.0153
vanilla_nodrop_10l_lrx30	dense	177.4	125.9	0.397	1.2583	15.7	+0.0153
fractal_m4_d2_2l_down_ffnswap_hs1	fractal	61.2	9.7	0.134	1.3929	6.6	+0.0157
fractal_m4_d2_2l_up_ffnswap_hs1	fractal	61.2	9.7	0.134	1.3861	10.8	+0.0155
fractal_m4_d3_2l_down_ffnswap_hs1	fractal	61.3	9.8	0.134	1.3957	7.3	+0.0160
fractal_m4_d3_2l_up_ffnswap_hs1	fractal	61.3	9.8	0.134	1.3864	9.6	+0.0161
fractal_300mt_ffnonly_tau0p1	fractal	72.3	20.8	0.196	1.4316	1.9	+0.0153
fractal_300mt_ffnonly_tau0p2	fractal	72.3	20.8	0.196	1.4585	1.2	+0.0129
fractal_m3_d2_2l_tied_hs2_lrx30	fractal	74.2	23.6	0.236	1.3230	11.8	+0.0144
ablation_down_only_lrx30	fractal	76.4	24.9	0.180	1.3507	8.5	+0.0147
ablation_lrx30	fractal	76.4	24.9	0.250	1.3456	9.6	+0.0141
fractal_300mt_lrx30	fractal	76.4	24.9	0.250	1.3439	10.2	+0.0149
fractal_300mt_lrx30_tauconst0p15	fractal	76.4	24.9	0.250	1.3400	9.6	+0.0148
fractal_300mt_lrx30_tauconst0p2	fractal	76.4	24.9	0.250	1.3435	9.4	+0.0146
fractal_300mt_tauconst0p1_seed24	fractal	76.4	24.9	0.250	1.3370	8.3	+0.0152
fractal_300mt_tauconst0p1_seed25	fractal	76.4	24.9	0.250	1.3375	9.2	+0.0147
fractal_m4_attnl0_ffn_hs0_lrx30	fractal	76.7	25.2	0.182	1.3594	10.9	+0.0154
ablation_static_perm_lrx30	fractal	78.3	26.9	0.256	1.5703	0.5	+0.0123
fractal_m3_d2_2l_tied_hs3_lrx30	fractal	80.1	29.4	0.258	1.3140	11.9	+0.0147
fractal_m4_attnl0_ffn_hs3_lrx30	fractal	81.4	29.9	0.191	1.3374	12.4	+0.0153
fractal_m4_d2_untied_hs4_lrx30	fractal	83.8	32.3	0.190	1.3142	11.9	+0.0157
fractal_m3_d1_2l_tied_hs4_lrx30	fractal	83.2	32.6	0.242	1.3113	10.9	+0.0150
fractal_m4_d2_2l_downonly_hs4_lrx30	fractal	84.6	33.1	0.197	1.3207	8.7	+0.0162
fractal_m4_d2_2l_stdied_hs4_lrx30	fractal	84.6	33.1	0.277	1.3088	12.3	+0.0148
fractal_m4_d2_2l_uponly_hs4_lrx30	fractal	84.6	33.1	0.197	1.3162	12.4	+0.0156
fractal_m3_d2_2l_tied_hs4_lrx30	fractal	86.0	35.3	0.281	1.3087	12.3	+0.0149
fractal_spec_untie0_300mt_lrx30	fractal	88.9	37.4	0.256	1.3402	9.5	+0.0157
fractal_untie0_300mt_seed23	fractal	89.0	37.5	0.250	1.3057	11.8	+0.0152
fractal_untie0_300mt_seed24	fractal	89.0	37.5	0.250	1.3057	12.1	+0.0153
fractal_untie0_300mt_seed25	fractal	89.0	37.5	0.250	1.3065	12.1	+0.0151
fractal_hs7p05_300mt	fractal	89.0	37.5	0.300	1.3220	8.3	+0.0153
fractal_m3_d1_4l_tied_hs1p5_lrx30	fractal	88.7	38.0	0.288	1.3099	12.5	+0.0143
fractal_m3_d1_3l_tied_hs3_lrx30	fractal	91.4	40.7	0.284	1.3011	12.8	+0.0148
fractal_d2_untied_all_spec_300mt	fractal	92.4	41.0	0.212	1.3273	10.2	+0.0160
fractal_d2_untied_all_300mt	fractal	92.4	41.0	0.207	1.3029	12.7	+0.0150
fractal_m3_d2_untied_hs6_lrx30	fractal	95.9	45.2	0.214	1.3058	12.4	+0.0159
fractal_d3_untied_all_lrx30	fractal	97.2	45.7	0.226	1.2966	12.6	+0.0157
fractal_d4_untied_all_lrx30	fractal	99.6	48.1	0.239	1.2932	13.1	+0.0152
fractal_d5_untied_all_lrx30	fractal	100.8	49.3	0.250	1.2972	13.0	+0.0145
fractal_m4_d2_untied_hs8_lrx30	fractal	105.3	53.8	0.233	1.3063	12.8	+0.0160
fractal_d3_untied_hs6_lrx30	fractal	112.4	60.9	0.256	1.2886	13.5	+0.0153
fractal_m4_d1_3l_untied_hs2p5_lrx30	fractal	115.3	63.8	0.271	1.2852	13.1	+0.0145
fractal_m16_d1_3l_untied_hs3_lrx30	fractal	116.5	65.0	0.274	1.2863	13.7	+0.0149

Table 4 continued

Configuration	Family	Params	Non-emb.	GFLOPs	FineWeb	LAMBADA	gap
fractal_m8_d1_3l_untied_hs3_lrx30	fractal	118.4	67.0	0.278	1.2850	14.7	+0.0144
fractal_m3_d2_2l_untied_hs4_lrx30	fractal	118.5	67.9	0.281	1.2828	13.2	+0.0149
fractal_d3_untied_hs7_lrx30	fractal	120.0	68.5	0.271	1.2877	13.2	+0.0151
fractal_m2_d1_3l_untied_hs2p5_lrx30	fractal	122.4	70.9	0.285	1.2833	15.0	+0.0141
fractal_m4_d1_3l_untied_hs3_lrx30	fractal	122.4	70.9	0.285	1.2797	14.4	+0.0146
fractal_m4_d1_3l_untied_hs3_seed24	fractal	122.4	70.9	0.285	1.2825	14.8	+0.0146
fractal_m4_d1_3l_untied_hs3_seed25	fractal	122.4	70.9	0.285	1.2828	14.8	+0.0141
fractal_m3_d1_3l_untied_hs3_lrx30	fractal	121.9	71.2	0.284	1.2824	15.2	+0.0147
fractal_m4_d1_4l_untied_hs333	fractal	152.2	100.7	0.359	1.2730	16.0	+0.0145
fractal_m4_d1_5l_untied_hs333	fractal	177.4	125.9	0.423	1.2718	13.8	+0.0147

References

- U. Bamba, A. Chavan, A. Thakur, S. Teig, and D. Gupta. DOT-MoE: Differentiable Optimal Transport for MoEification. In *ICML*, 2026. arXiv:2606.01666.
- R. Cai, S. Muralidharan, G. Heinrich, H. Yin, Z. Wang, J. Kautz, and P. Molchanov. Flextron: Many-in-One Flexible Large Language Model. In *ICML*, 2024. arXiv:2406.10260.
- M. Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *NeurIPS*, 2013. arXiv:1306.0895.
- T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Ré. Learning Fast Algorithms for Linear Transforms Using Butterfly Factorizations. In *ICML*, 2019. arXiv:1903.05895.
- T. Dao, B. Chen, N. S. Sohoni, A. Desai, M. Poli, J. Grogan, A. Liu, A. Rao, A. Rudra, and C. Ré. Monarch: Expressive Structured Matrices for Efficient and Accurate Training. In *ICML*, 2022. arXiv:2204.00595.
- M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser. Universal Transformers. In *ICLR*, 2019. arXiv:1807.03819.
- Devvrit, S. Kudugunta, A. Kusupati, T. Dettmers, K. Chen, I. Dhillon, Y. Tsvetkov, H. Hajishirzi, S. Kakade, A. Farhadi, and P. Jain. MatFormer: Nested Transformer for Elastic Inference. In *NeurIPS*, 2024. arXiv:2310.07707.
- W. Fedus, B. Zoph, and N. Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *JMLR*, 23, 2022. arXiv:2101.03961.
- H. Hazimeh, Z. Zhao, A. Chowdhery, M. Sathiamoorthy, Y. Chen, R. Mazumder, L. Hong, and E. Chi. DSelect- k : Differentiable Selection in the Mixture of Experts with Applications to Multi-Task Learning. In *NeurIPS*, 2021. arXiv:2106.03760.
- K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An Optimizer for Hidden Layers in Neural Networks. Blog post, <https://kellerjordan.github.io/posts/muon/>, 2024.
- M. Jordan and R. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2), 1994.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling Laws for Neural Language Models. *arXiv preprint*, 2020. arXiv:2001.08361.
- Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *ICLR*, 2020. arXiv:1909.11942.
- G. Larsson, M. Maire, and G. Shakhnarovich. FractalNet: Ultra-Deep Neural Networks without Residuals. In *ICLR*, 2017. arXiv:1605.07648.
- M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer. BASE Layers: Simplifying Training of Large, Sparse Models. In *ICML*, 2021. arXiv:2103.16716.

-
- P. Li, Z. Zhang, P. Yadav, Y.-L. Sung, Y. Cheng, M. Bansal, and T. Chen. Merge, Then Compress: Demystify Efficient SMoE with Hints from Its Routing Policy. In *ICLR*, 2024. arXiv:2310.01334.
- T. Li, Q. Sun, L. Fan, and K. He. Fractal Generative Models, 2025. arXiv:2502.17437.
- T. Liu, J. Puigcerver, and M. Blondel. Sparsity-Constrained Optimal Transport. In *ICLR*, 2023. arXiv:2209.15466.
- G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning Latent Permutations with Gumbel-Sinkhorn Networks. In *ICLR*, 2018. arXiv:1802.08665.
- S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer Sentinel Mixture Models. In *ICLR*, 2017. arXiv:1609.07843.
- P. Nawrot, S. Tworkowski, M. Tyrolski, Ł. Kaiser, Y. Wu, C. Szegedy, and H. Michalewski. Hierarchical Transformers Are More Efficient Language Models. In *Findings of NAACL*, 2022. arXiv:2110.13711.
- D. A. Nguyen, H. B. Ta, N. Le Duc, T. M. Nguyen, and T. Tran. Selective Sinkhorn Routing for Improved Sparse Mixture of Experts, 2025. arXiv:2511.08972.
- D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context. In *ACL*, 2016. arXiv:1606.06031.
- G. Penedo, H. Kydliček, L. Ben Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. Von Werra, and T. Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *NeurIPS Datasets and Benchmarks*, 2024. arXiv:2406.17557.
- J. Puigcerver, C. Riquelme, B. Mustafa, and N. Houlsby. From Sparse to Soft Mixtures of Experts. In *ICLR*, 2024. arXiv:2308.00951.
- J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap. Compressive Transformers for Long-Range Sequence Modelling. In *ICLR*, 2020. arXiv:1911.05507.
- D. Raposo, S. Ritter, B. Richards, T. Lillicrap, P. Humphreys, and A. Santoro. Mixture-of-Depths: Dynamically Allocating Compute in Transformer-Based Language Models, 2024. arXiv:2404.02258.
- S. Sabour, N. Frosst, and G. Hinton. Dynamic Routing Between Capsules. In *NeurIPS*, 2017. arXiv:1710.09829.
- M. E. Sander, P. Ablin, M. Blondel, and G. Peyré. Sinkformers: Transformers with Doubly Stochastic Attention. In *AISTATS*, 2022. arXiv:2110.11773.
- N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *ICLR*, 2017. arXiv:1701.06538.
- J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing*, 2024. arXiv:2104.09864.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention Is All You Need. In *NeurIPS*, 2017. arXiv:1706.03762.
- X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*, 2018. arXiv:1707.01083.
- Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon. Mixture-of-Experts with Expert Choice Routing. In *NeurIPS*, 2022. arXiv:2202.09368.