

A Steeper Slope Is Not Yet a New Law: What a Claimed Architectural Change to a Scaling Exponent Has to Survive, Why Studies of Looping, Growth, Routing and Optimizers Reach Opposite Verdicts, and the Floor, Recipe and Ladder Controls That Would Decide It

Pranay Mahendrakar

SONYTECH · ORCID 0009-0003-7224-029X

pranaymahendrakar@sonytech.in · mahendrakarpranay@gmail.com · +91 6361723454

Abstract. A long-standing working rule of language-model scaling holds that architectural changes move the constant of a scaling law and leave its exponent alone, so that a better design buys a fixed factor of compute at every budget rather than a factor that compounds. A September 2026 study reports the opposite for three interventions on looped transformers - a normalise-and-reinject boundary operator, depth growth during training, and their combination - with compute multipliers over a standard transformer that rise across two decades of compute, and a concurrent study reports a steeper frontier for a looped mixture-of-experts model. Another 2026 study finds that recurrence loses to unique depth at matched compute by a margin that does not close, and a third finds that looping pays at a fixed parameter count without showing an advantage once size and data are optimised for compute. In this analysis the author reviews and synthesises published results to show that these verdicts are not yet answers to one question, because a reported exponent is indexed by five choices that the studies make differently: the axis compute is counted on, how the irreducible floor is fitted, whether each arm has its own training recipe, which shape and allocation rule the ladder follows, and how wide the fitted window is. The paper derives that a lower floor and a steeper exponent both produce a multiplier that rises with scale over a finite window, and that fitting every arm against the baseline's floor reads the first as the second by construction. It shows from the anchor study's own ablations that the boundary operator's exponent gain disappears under the baseline's recipe, and that when executed depth is held fixed its gain in the two-pass pair (Untied-2 against Deep Vanilla) becomes a constant, so that the measured exponent belongs to a ladder and a recipe as much as to a block; the corresponding controls for growth are not reported. It finds the same protocol dependence in two precedents, routing and optimizers, where the trend with scale changed between studies, in the routing case from shrinking to widening. The paper consolidates 24 published results in a table, maps eleven designs onto the five indices, states a classification procedure, and specifies the four controls that would separate a new exponent from a new floor.

Keywords: scaling laws; scaling exponents; looped transformers; model growth; compute-optimal training; irreducible loss; hyperparameter transfer; mixture of experts;

pretraining

SONYTECH ORCID: 0009-0003-7224-029X

1. Introduction

Scaling laws are summaries. Kaplan et al. (2020) fitted the cross-entropy of transformer language models as a power law in parameters, data and compute over several orders of magnitude; Hoffmann et al. (2022) refitted the parameter and data terms over more than 400 models with an explicit irreducible floor and showed that compute-optimal training should grow parameters and tokens in equal proportion. Written in the compute-optimal form that most recent comparisons use, the summary is a three-number object: a floor E that no amount of compute removes, an amplitude A , and an exponent γ that sets how fast the reducible part $L - E$ shrinks as compute C grows.

The three numbers do different jobs. A change in A moves the whole curve down by a fixed factor in reducible loss, which, inverted, means that one design reaches a given loss with a fixed fraction of the other's compute at every budget. A change in γ tilts the curve, so the compute saved grows as a power of the budget. A change in E moves the asymptote. Practitioners care about the difference because they spend the most at the largest budgets, which are also the ones they cannot afford to measure. A constant factor measured at 10^{19} FLOPs is worth the same factor at 10^{25} ; an exponent change measured at 10^{19} is worth much more there, and a floor change is worth almost nothing until the baseline gets close to its floor, after which it is worth everything.

A working rule has organised how architectural results are read. Hestness et al. (2017), across machine translation, language modelling, image classification and speech, reported that improved models shift the learning curve but do not appear to change its exponent. Bansal et al. (2022) found that swapping a transformer for a transformer-LSTM hybrid or a decoder-only model barely moved data-scaling exponents in translation. Kaplan et al. (2020) found that within a wide range the shape of a

transformer hardly matters once non-embedding parameters are fixed. The rule that follows is that architecture moves A while data can move γ . The paper that now disputes its first half states that half as "the conventional wisdom that changes to the architecture generally only affect the scaling constants" (Chen et al., 2026).

That paper is Chen et al. (2026). Working with the prelude-core-coda formulation of looped transformers from Geiping et al. (2025), it fits a separate compute-optimal recipe for each of eight architectures, trains ladders from about 10^{18} to 10^{20} FLOPs on FineWeb, and reports that a boundary operator which normalises the residual stream and re-injects the prelude output, and a schedule that doubles the number of core passes partway through training, each raise the scaling exponent. Its headline is a compute multiplier over a standard transformer that climbs from 1.30 to 1.55 across the ladder for the best variant, a 7.4B-parameter run at eight times the largest fitted budget that lands on the extrapolated curve, and a projection of 2.7 at 10^{25} FLOPs. A concurrent study, Wang et al. (2026b), loops the middle half of a mixture-of-experts transformer while matching FLOPs, parameters and key-value cache, fits a separate law for each architecture, and reports a frontier exponent 5.5 percent larger than its baseline's.

Against these sit two looped-model studies from the spring of 2026 that reach the opposite verdict. Schwethelm et al. (2026) hold executed depth at 20 layers and replace unique layers with recurrences; they estimate that r passes of a shared block are worth about $r^{0.46}$ unique blocks, so the looped models trail the baseline at matched compute by margins that do not close inside their grid or at a check run about twenty times beyond it. Prairie et al. (2026) find looping helps at fixed parameter and data budgets but, as Chen et al. (2026) note, do not show an advantage once model size and data are jointly optimised for compute.

The question this paper asks is what an exponent claim of this kind measures, and what it would take to establish one. In this analysis the author reviews and synthesises published results to show that the answer turns on five choices that the studies above make differently, that one of those choices - the floor - can manufacture an apparent exponent change without any change in the architecture's behaviour at a given size, and that two others - the recipe and the ladder - can remove one, as the anchor study's own appendix ablations already show for its boundary operator. The paper makes four contributions.

First, it states the five indices of an exponent claim - axis, floor, recipe, ladder and window - and derives, as elementary algebra, what each does to a compute multiplier (Sections 4 and 5).

Second, it reads the looped-model disagreement through those indices and finds that the four studies differ on at

least two indices each, so their verdicts are compatible (Section 6), and it does the same for growth (Section 7) and for depth-repair operators (Section 8), where the anchor's own controls show the exponent gain depends on depth growing along the ladder.

Third, it shows that the same structure has already played out twice: for routed models, where a fixed-token protocol gave diminishing returns and a token-optimised protocol gave widening returns (Clark et al., 2022; Krajewski et al., 2024), and for matrix-preconditioned optimizers, where one hyperparameter transfer rule gave a constant 1.4 times speedup and a different one, a hyperparameter law fitted on smaller runs with its own base tuning and data-to-model ratios, gave one that shrank to 1.1 (Qiu et al., 2025; Wen et al., 2025) (Section 9).

Fourth, it consolidates the measured values in Table 1, maps eleven designs onto the indices in Table 2, states a classification procedure as Algorithm 1, and specifies the controls that would settle the open part (Sections 12 to 14).

The paper's conclusion is narrower than either side's. The working rule was never a theorem: the theory of scaling exponents ties them to the spectrum of the data as it is seen through the learner's features, which leaves room for architecture to matter (Maloney et al., 2022; Bordelon et al., 2024b). The anchor study is careful and its multipliers are measured, not extrapolated, inside its window. But what it has shown is that, on a fixed-aspect-ratio ladder, with per-architecture recipes and a floor shared with the baseline, the multiplier rises. Whether that is a new exponent, a new floor, or a property of how the ladder grows depth is not yet determined by any published result.

2. What the anchor study reports

Chen et al. (2026) build every model as one three-stage network: a prelude of blocks that embeds the input into a representation e , a core of blocks applied K times, and a coda. The standard transformer is $K = 1$ with identity boundaries. The boundary operator they study replaces the identity before each core pass and before the coda with an RMS normalisation of the state plus a scaled copy of e . Weight tying across passes gives a looped transformer; untying gives each pass its own core, which is a deeper transformer with the operator. Growth starts training at two core passes and switches to four partway through, either reusing the tied core or copying the untied cores, which is the depthwise stacking of Du et al. (2024). Matched pairs isolate one change at a time: Vanilla against Operator-1 for the operator, Loop-2 against Untied-2 for weight sharing, and the fixed variants against their Grow counterparts for growth, with two operator-free controls (Deep Vanilla and Deep Vanilla Grow).

The recipe is fitted per architecture in four stages: base hyperparameters at a depth of eight blocks, tokens per stored parameter at five budgets, the growth transition, and a learning-rate scaling rule. The fitted token ratios are 5 for Vanilla, 6 with the operator, and 7 to 8 with growth; growth lands at one-half to four-fifths of training. The ladders hold the width-to-depth ratio at 128, so that depth and width rise together with model size, from 120M to 1.8B stored parameters for Vanilla, and all ladders end near 10^{20} FLOPs. Batch size is fixed, and the authors note that their recipes do not address the joint optimisation of batch size with scale.

The fitting procedure matters for what follows. The floor E is fitted once, on the Vanilla ladder, with a Huber objective; every other arm is then fitted as a straight line in $\log(L - E)$ against $\log C$, sharing that floor, and the reported exponent is the slope of that line. The regression standard error of each slope is below 10^{-3} . Compute multipliers are computed separately, by interpolating between measured points in log loss and log compute, without extrapolation, and indexed by Vanilla's compute.

The single-epoch results are these. Operator-1, which matches Vanilla in parameters and FLOPs, has a multiplier of 1.12 at 10^{18} FLOPs and 1.25 at 10^{20} . Untied-2 goes from 1.19 to 1.34. Untied-Grow goes from 1.30 to 1.55, so growth over Untied-2 goes from 1.09 to 1.16. Untying is a constant: Untied-2 over Loop-2 is 1.08 then 1.06, and Untied-Grow over Loop-Grow is 1.16 then 1.14. Deep Vanilla, which executes as many blocks as Untied-2 without the operator, sits flat near 1.08, which the authors read as a constant gain from a better width-to-depth ratio. On FineWeb-Edu with the same recipe, the fitted laws, again with a shared floor, give an Untied-Grow multiplier of 1.6 at 10^{20} , the end of the measured ladders, 1.8 at 1.23×10^{21} , the budget of a held-out 7.4B run, and 2.7 at 10^{25} . The held-out run itself lands 0.03 below the forecast loss; no Vanilla model was trained at that budget, so the 1.8 is read off the fitted Vanilla law. Its CORE score of 0.3865 is on par with an estimated 0.3852 for GPT-3 13B at roughly one-twentieth of the compute, a comparison the authors themselves call indicative rather than controlled, since the data and evaluation pipelines differ.

Two further sets of results are in the appendices, and they carry much of this paper's argument. The first concerns the recipe. Transferring Vanilla's base hyperparameters to Operator-1 costs 7.8×10^{-3} in loss at depth eight and, in the authors' words, "causes the exponent improvements to become constant ones". Under constant base recipes with no learning-rate scaling, the looped and grown models look better against Vanilla than they do under scaled recipes. Adding output-multiplier or weight-decay scaling to Vanilla's learning-rate rule changes Vanilla's own multiplier by at most about 3 percent. The second concerns shape. A still deeper ladder (depth-to-width 1:64) of Operator-1 loses

Operator-1's increasing advantage; and at a fixed executed depth of eleven blocks with only width scaled, Untied-2's advantage over Deep Vanilla becomes a better constant rather than a better exponent. The authors take the second result as support for their hypothesis that the operator and growth improve the exponent because the fraction of depth they rescue grows with depth, and compute-optimal models get deeper with compute.

A third appendix result concerns the optimizer: Muon has a constant advantage over Adam when only width is scaled but an advantage that shrinks with scale when width and depth are scaled together. A fourth concerns the multi-epoch regime, where the authors fix 100M unique tokens, train for ten epochs, and find that the compute-optimal number of loops rises from about 1.4 to 6.7 across budgets and that scaling loops matches the best weight-decay-tuned model-size scaling with 2.2 times less compute. That is a regularisation result in a data-constrained regime and is a different claim from the single-epoch exponent claim; this paper returns to it only in Section 10.

3. Where the working rule came from, and what theory says about it

The empirical case for the rule is older and narrower than its use suggests. Hestness et al. (2017) measured generalisation error against training-set size, fitting the best model size at each data shard. For word language models on the Billion Word benchmark, LSTMs and recurrent highway networks had nearly identical data exponents in the text's fit (-0.0656 , to within one percent), although the summary table lists -0.066 for the LSTM and -0.070 for the recurrent highway network, and for character models, SGD and Adam gave -0.0936 and -0.0954 , with Adam shifting the curve down by about five percent. These are data exponents measured on recurrent networks with tuned sizes, not compute-optimal exponents of transformer families. Bansal et al. (2022) extended the observation to translation with transformers and found that architectural variants and synthetic noise barely moved the data exponent, while back-translated data did degrade it. Henighan et al. (2020) found power-law-plus-constant laws in four further domains, with nearly universal exponents for the optimal model size as a function of compute.

Kaplan et al. (2020) contributed the shape result - aspect ratio can vary by a factor of 40 at a cost of a few percent - and two comparisons that matter here. LSTMs matched transformers on early tokens of the context but fell behind on later ones. And recurrent transformers, which reuse parameters, did slightly better than standard ones per parameter but slightly worse per unit of compute. That second observation is the axis problem of Section 5.1 in its earliest form.

More recent evidence for the rule comes from loss-to-loss relationships. Brandfonbrener et al. (2024) showed that

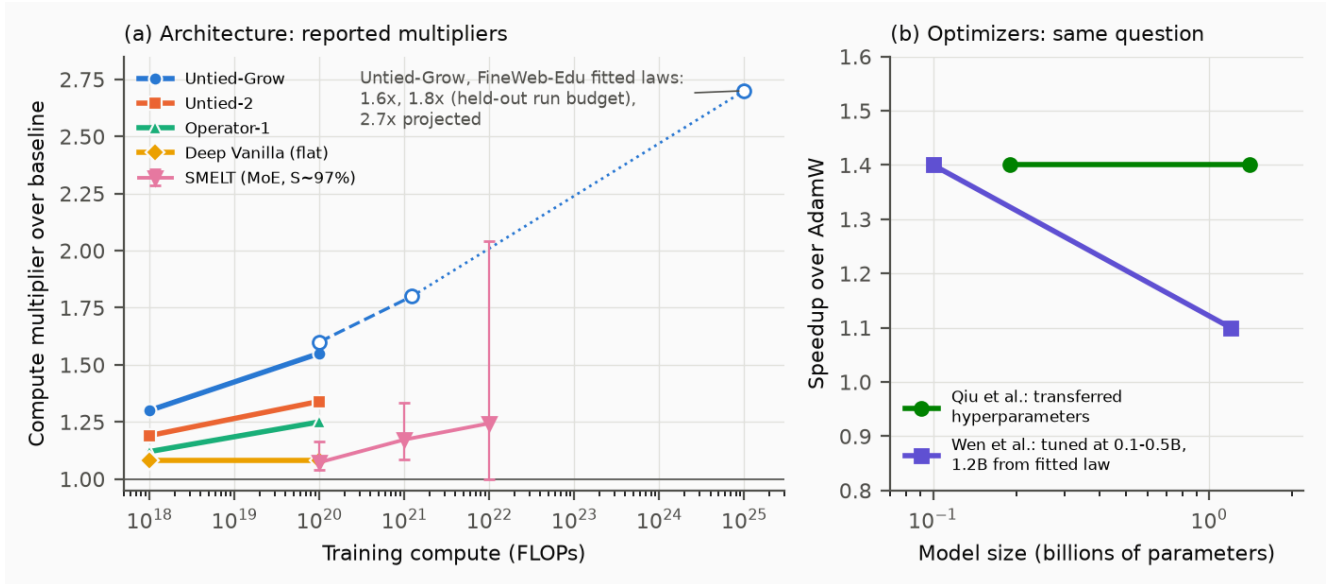


Figure 1. Published compute multipliers and optimizer speedups, plotted as reported. (a) Multipliers over a standard transformer from Chen et al. (2026): filled markers are measured FineWeb ladder endpoints at 10^{18} and 10^{20} FLOPs (Deep Vanilla reported as flat near 1.08); open markers are Untied-Grow on FineWeb-Edu as given by the fitted laws with a shared floor: 1.6x at 10^{20} , 1.8x at the budget of the held-out 7.4B run, and 2.7x projected at 10^{25} . SMELT (Wang et al., 2026b), sparsity about 97 percent: reported compute savings of 6.8, 14.7 and 19.6 percent at 10^{20} , 10^{21} and 10^{22} (the last extrapolated), converted here to multipliers $1/(1-s)$, with the reported bootstrap 95 percent intervals. The x-axis is training compute, but the series are indexed differently, as their sources report them: the FineWeb endpoints are at Vanilla’s budgets, the 1.8x point is at the compute of the 7.4B variant run, and SMELT’s savings are at SMELT’s own budget. (b) Speedup of matrix-preconditioned optimizers over AdamW: about 1.4x from 190M to 1.4B parameters with transferred hyperparameters (Qiu et al., 2025), against 1.4x at 0.1B falling to 1.1x at 1.2B with optimizers tuned per scale from 0.1B to 0.5B and the 1.2B settings predicted by a hyperparameter law fitted on those runs (Wen et al., 2025).

the losses of models trained on different datasets are related by shifted power laws when paired by compute, and Mayilvahanan et al. (2025) found that the pretraining data, not the architecture, sets such loss-to-loss trends, with Llama-style transformers and Mamba-style state-space models largely on the same line. That is evidence that architecture does not change how gains on one distribution translate into gains on another. It is not evidence about how fast loss falls with compute, which is a different relationship.

The rule is not a consequence of theory. Sharma and Kaplan (2020) derived that if a network is effectively regressing on a data manifold of intrinsic dimension d , the parameter exponent should be about $4/d$, a property of the data. Bahri et al. (2024) separated variance-limited from resolution-limited regimes and observed empirical links between exponents and both datasets and architectural aspect ratio. Maloney et al. (2022) solved a random-feature model in which the power law in the data’s spectrum is extended by the nonlinear feature map and translated into the loss, with a plateau when the spectrum runs out. Michaud et al. (2023) tied the exponent to a power law in how often discrete skills are used. Hutter (2021) built a toy model that can produce any exponent and used it to ask whether exponents are universal or set by the data distribution. In each of these accounts the exponent is a property of the data as filtered by the learner, which means the learner can matter.

Two lines of theory say how. Bordelon et al. (2024a) showed in a solvable model that the exponents in training time and in model size differ, and that the late-time approach to the infinite-width limit has an exponent that depends on architecture and task. Bordelon et al. (2024b) showed that when the target lies outside the function space of the network’s initial kernel - a hard task - feature learning can nearly double the compute exponent, while for easier tasks the exponent is unchanged. Paquette et al. (2024) mapped compute-optimal exponents across phases set by data complexity, target complexity, optimizer noise and the embedding of features. None of these models is a transformer, but together they say that an architectural change which alters what the network can resolve per unit of compute could alter the exponent, and that the effect should be task-dependent. The working rule is therefore an empirical regularity about the interventions that happened to be tested, not a law.

That matters for how the anchor study should be read. It does not need to overturn theory, only an induction from earlier tests. The question is whether its own measurement is of the kind that would distinguish an exponent from the alternatives. A recent survey of more than fifty scaling studies, Li et al. (2025), records the disagreement without settling it, setting Hestness et al. (2017) against Poli et al. (2024), whose hybrid architectures beat transformers on both compute-optimal and over-trained scaling; it also documents that most

scaling studies under-report the fitting details needed to reproduce them. Its paraphrase of Hestness et al. is that architectural improvements "only shift the irreducible loss". The original sentence says improved models "only shift the error". The slip is small, but it runs together two different parameters, A and E , which is the confusion at the centre of this paper.

4. A multiplier has three ways to rise

The anchor study's evidence for an exponent change is a compute multiplier that rises with scale: the left and middle panels of its Figure 3, and Figure 1a here. This section derives what can make a multiplier rise. The derivations are elementary and use only the compute-optimal form the anchor itself adopts.

Write the baseline as $L_A(C) = E_A + A_A C^{(-g_A)}$ and a variant as $L_B(C) = E_B + A_B C^{(-g_B)}$. The multiplier at a given loss l is $\pi(l) = C_A(l) / C_B(l)$, the ratio of the compute each needs to reach l .

Case 1, constant. If $E_B = E_A$ and $g_B = g_A = g$, then $\pi = (A_A / A_B)^{(1/g)}$ at every loss. The multiplier is flat. Deep Vanilla's 1.08 and the untying factors of 1.06 to 1.16 are of this kind.

Case 2, exponent. If $E_B = E_A$ and $g_B > g_A$, then solving $A_A C_A^{(-g_A)} = A_B C_B^{(-g_B)}$ gives C_A proportional to $C_B^{(g_B/g_A)}$, so π grows as a power of compute, with exponent $g_B/g_A - 1$ when indexed by the variant's compute, and slightly less when indexed by the baseline's. This is the case the anchor study claims.

Case 3, floor. If $g_B = g_A = g$ and $A_B = A_A = A$, but $E_B = E_A - \delta$ with $\delta > 0$, then setting the two losses equal gives $(\pi C)^{(-g)} = C^{(-g)} - \delta / A$, where C is the variant's compute, so $\pi = (1 - (\delta / A) C^g)^{(-1/g)}$. This multiplier rises with compute and becomes unbounded when the target loss reaches the baseline's floor E_A - that is, when the variant's reducible loss has fallen to δ - because below that loss the baseline cannot follow at any budget. Over a window in which the target loss is still well above E_A , this curve rises smoothly and is hard to tell from Case 2.

A fourth case combines the two in opposite directions. If $g_B > g_A$ but $E_B > E_A$, the multiplier rises while the exponent dominates and must eventually fall below one, since at large enough compute the variant cannot go below E_B while the baseline approaches E_A . This is not a hypothetical: Section 5.2 shows that SMELT's own fitted parameters have this pattern.

Two consequences follow, and both are derivations, not measurements. First, a rising multiplier inside a finite window does not identify Case 2. It identifies "Case 2 or Case 3 or a mixture", and the two extrapolate very differently: Case 2 compounds at a constant rate in log compute, while Case 3 rises slowly at first and then diverges. Second, a procedure that fits E once on the baseline and imposes it on every arm sets δ to zero

by assumption. Under that procedure a variant with a lower true floor will show a $\log(L - E_A)$ curve that bends downward, and a straight-line fit through it reports a steeper slope. The regression standard error of that slope measures how well a line fits the points given E_A ; it does not include uncertainty in E_A , or the possibility that E differs between arms.

The size of the claimed exponent changes is worth stating in the same terms. Under Case 2, a multiplier that rises from 1.12 to 1.25 across a factor of 100 in compute implies g_B / g_A of about 1.024: Operator-1's exponent would be roughly 2.4 percent steeper than Vanilla's. The same arithmetic gives about 2.6 percent for Untied-2, about 3.9 percent for Untied-Grow, and about 1.4 percent for growth over Untied-2. SMELT's reported frontier exponents, 0.237 and 0.250, differ by 5.5 percent. These are the author's arithmetic on the published multipliers, under the pure-power-law assumption, and they are offered as an order of magnitude only. For Untied-Grow the anchor also states its fitted exponents, against a Vanilla-fitted floor for each corpus: "0.1168 versus 0.1113 on FineWeb, and 0.1146 versus 0.1095 on FineWeb-Edu", which is 4.9 and 4.7 percent steeper, somewhat more than the multipliers imply. Exponent differences of two to five percent are what a decision between Case 2 and Case 3 has to resolve. Section 5 asks what else, besides a floor, could produce them.

5. Five indices of an exponent claim

5.1 Axis: what compute is counted, and against what

An exponent is a slope against something, and the something is chosen. Kaplan et al. (2020) found that recurrent transformers look slightly better than standard ones per parameter and slightly worse per FLOP. Tay et al. (2022) measured ten architectures with the same 2^{19} -step protocol and fitted linear slopes of upstream quality against FLOPs and against parameters. The Universal Transformer, which shares parameters across depth (Dehghani et al., 2019), had a steeper slope than the vanilla transformer against parameters (0.56 against 0.47) and a shallower one against FLOPs (0.50 against 0.54). Whether looping "improves the exponent" in that study depends on which column one reads. ALBERT, another parameter-sharing model, had slopes of 0.13 against parameters and 0.08 against FLOPs, the lowest in the study on the FLOP axis.

The anchor study counts FLOPs by executed depth, with an estimator that includes attention, so a looped model is charged for every pass. That is the correct axis for a claim about training cost, and it is why Untied-2 and Loop-2 have the same compute. It is not the only axis that matters: Chen et al. (2026) also report that the gains survive measured wall-clock time on their hardware, and Schwethelm et al. (2026) show how far the parameter and compute axes can part: at four recurrences, a 410M

looped model performs like a 580M non-looped one but costs as much to train as a 1B one. A claim of a new exponent should name its axis and should be stated on the compute axis if it is to support the anchor study's practical conclusion.

What is counted as compute can also bias the exponent within one axis. Pearce and Song (2024) traced much of the gap between the Kaplan and Chinchilla allocation exponents (0.73 against 0.50 for the optimal parameter count) to counting non-embedding rather than total parameters at small scale. Porian et al. (2024) found three further causes of the same discrepancy - the compute of the last layer, warmup duration, and optimizer settings that were not retuned with scale - and recovered the Chinchilla law once they were corrected. An exponent comparison between two architectures inherits all of these choices, and inherits them differently if the architectures differ in embedding share or per-layer cost.

5.2 Floor: one shared E or one per arm

Section 4 derived why a shared floor matters. The published looped-model studies split on it. The anchor fits E on Vanilla and shares it. Wang et al. (2026b) fit each architecture's own E , and their reducible-loss plot subtracts each architecture's own floor. Schwethelm et al. (2026) also fit per-architecture laws.

The per-arm fits are informative because they show how unstable E is across nearby arms. Schwethelm et al. (2026) report fitted floors of 1.56, 1.60, 1.56 and 1.69 nats for recurrence counts 1, 2, 4 and 8, with capacity exponents of 0.251, 0.216, 0.191 and 0.235 - neither sequence monotone in r . The authors say directly that the fitted exponents are not comparable across r , because sampling on iso-FLOP curves puts each architecture in a different region of the parameter-data plane where the exponents are weakly identified, and they cite Besiroglu et al. (2024) for that weakness. They compare architectures on the compute-optimal frontier instead. Wang et al. (2026b) fit SMELT's floor at 1.4493 against the baseline's 1.4439, and its capacity and data exponents above the baseline's (0.3892 against 0.3703, and 0.7011 against 0.6594). Taken literally, those two laws are an instance of the fourth case in Section 4: a steeper approach to a higher floor, under which SMELT's saving grows across the fitted range and must eventually reverse. A floor difference of 0.0054 nats fitted on a grid of four model sizes is very unlikely to be resolved by those data, so this is not a prediction that SMELT will lose at scale. It is a demonstration that, once floors are fitted per arm, the asymptotic sign of the comparison is set by the floors and not by the exponents, and the floors are the least well determined parameters in the fit.

Besiroglu et al. (2024) showed how optimistic scaling-fit uncertainty can be: the confidence intervals reported by Hoffmann et al. (2022) for their third method would have required over 600,000 experiments, against fewer

than 500 likely run. Choshen et al. (2024), estimating more than a thousand laws from 485 published models, found that seed-to-seed variability is large enough that several small models can be worth more than one large one. Ivgi et al. (2022) found that revealing scaling laws at all required careful tuning and repeated runs to estimate uncertainty. Alabdulmohsin et al. (2022) argued that fits should be judged by extrapolation error, not by goodness of fit. Against that background, a slope standard error below 10^{-3} conditional on a shared floor is a statement about the line, not about the exponent.

5.3 Recipe: whose hyperparameters, scaled how

The anchor study's most careful design choice is fitting a recipe for every arm, and its appendices show why: under Vanilla's base hyperparameters, Operator-1's exponent gain becomes a constant gain. Under constant recipes without learning-rate scaling, the looped and grown models look better than under scaled recipes. Both results say that the measured exponent of an architecture is a joint property of the architecture and the rule that sets its hyperparameters along the ladder.

That is consistent with a growing literature. Yang et al. (2022) showed that a suitable parameterisation lets optimal hyperparameters stay stable as width grows, and Everett et al. (2024), across tens of thousands of models up to 26.8B parameters, found that the best learning-rate scaling rule was often one that earlier assumptions would have excluded. Mlodozieniec et al. (2025) extended transfer across width, depth, batch size, training duration and individual modules. Porian et al. (2024) identified optimizer settings that were not retuned with scale as one of the three causes of the gap between two published compute-optimal allocation exponents. Hagele et al. (2024) showed that a constant learning rate with a cooldown scales like cosine decay, which changes what a ladder of intermediate checkpoints can be used for. Each of these results moves the baseline's curve. If a baseline's recipe is slightly worse at large scale than at small, its fitted exponent is too shallow, and every architecture compared against it acquires an apparent exponent gain.

The anchor's tuning is extensive - a separate four-stage fit per arm - but it is not unlimited: the learning-rate rule is fitted at depths eight to ten and the base hyperparameters at depth eight, and both are then carried up the ladder. A rule fitted on small models and extrapolated is exactly the place where Qiu et al. (2025) found drifting optima for preconditioned optimizers. The anchor's own Figure 16(a), in which scaling Vanilla's learning rate "substantially improves its ladder", shows how much the baseline's slope depends on this one choice. The anchor's check on the other hyperparameters is narrower than a bound on the architecture comparisons: adding output-multiplier or weight-decay scaling to Vanilla's learning-rate rule changes Vanilla's own multiplier by at most about 3 percent. Whether the

other hyperparameters, kept at their base-tuned values, bound the residual error in the exponent gap is not reported.

5.4 Ladder: how shape and allocation grow with compute

An exponent is fitted along a ladder: a rule that says, for each budget, how large the model is, what shape it has, and how many tokens it sees. Change the rule and the slope changes, even for the same architecture. The anchor's ladder holds width-to-depth at 128, so depth rises with the cube root of the non-embedding parameter count. Its own controls show that this matters to the claim. At a fixed executed depth with only width scaled, Untied-2's advantage over Deep Vanilla is a constant. On a deeper 1:64 ladder, Operator-1's rising advantage disappears. The fixed-depth result is what the authors' explanation - that the operator rescues a fraction of depth that grows with depth - predicts. The 1:64 result is harder to square with it, since a deeper ladder gives the operator more depth to rescue, and the authors do not reconcile the two. Either way, both controls show the exponent gain depending on the ladder: it is a property of the pairing between the operator and a particular rule for growing depth with compute. On a ladder that scales only width, the same block shows a constant.

This matters because ladder choice is not settled. Kaplan et al. (2020) found shape nearly irrelevant at fixed parameters; Levine et al. (2020) derived a width-dependent transition between depth-efficient and depth-inefficient regimes in self-attention; Tay et al. (2021) found that shape matters for downstream fine-tuning and that good scaling protocols differ across compute regions; Liu et al. (2024) found deep and thin designs better at sub-billion scale; and the anchor itself finds that Deep Vanilla's shape is worth a constant 1.08 contrary to Kaplan et al. (2020). If the compute-optimal aspect ratio itself changes with scale, then a fixed-ratio ladder is not compute-optimal in shape at every budget, and the gap between a fixed-ratio baseline and a variant that is less sensitive to shape could widen for reasons unrelated to the variant's mechanism.

Token allocation is the other half of the ladder. The anchor fits one token-to-parameter ratio per architecture and finds no drift with scale for any family, which is the right check. Growth complicates it: a grown model has a different parameter count before and after the transition, and its token budget is recomputed after choosing the transition point. Liew and Kato (2025) found that when a model is reused for further pretraining, the exponent with respect to second-stage tokens falls logarithmically with the number of first-stage tokens, so the benefit of growth depends on where along training it happens. The anchor's reported Untied-Grow and Loop-Grow ladders use transition fractions fitted per size; only Deep Vanilla Grow uses one fixed fraction. Replacing the fitted fractions with a fixed $\rho = 0.30$ changed loss by

-0.0009 to +0.0019. Whether the fitted transition would still be optimal at 10^{25} FLOPs is not known.

5.5 Window: how many decades, how many points

The anchor's ladders span two decades of compute. Schwethelm et al. (2026) span about 1.7 (a factor of 50). Wang et al. (2026b) fit four model sizes and report bootstrap intervals that widen quickly beyond the fitted range: at the 97 percent sparsity setting, SMELT's saving is 6.8 percent [4, 14] at 10^{20} , 14.7 percent [8, 25] at 10^{21} , and 19.6 percent [0, 51] at 10^{22} , the last extrapolated. The interval at 10^{22} includes no saving at all. Caballero et al. (2022) showed that scaling curves across many tasks are better described by smoothly broken power laws, whose slope changes at break points, than by single power laws. Over two decades, a curve with a gentle break and a curve with a slightly different exponent are hard to tell apart.

The anchor's strongest evidence against a window artefact is its held-out run at eight times the largest fitted budget, which landed 0.03 below the forecast. That is a real check, and a forecast that is beaten rather than missed. It is also one point, on one arm, without a matching Vanilla run at that budget, and a single variant point cannot separate Case 2 from Case 3 at that distance: both predict that the variant keeps falling along a curve fitted to its own ladder, and the multiplier of 1.8 at that budget is computed against the extrapolated Vanilla law, not measured. The projection to 2.7 at 10^{25} is model-dependent in the way Section 4 describes, and the authors' text says, of the downstream version of the same projection, that it assumes the small-scale relationships continue.

6. Looping: four studies, four protocols

Read against the five indices, the looped-model studies are not in conflict. They measured four different things.

Schwethelm et al. (2026) hold executed depth at 20 layers at every budget, sweep width, and compare recurrence counts of 1, 2, 4 and 8 between 4.64×10^{17} and 2.15×10^{19} FLOPs in 116 runs. Recurrence trades unique parameters for passes at constant per-token compute, and their joint law prices r passes of a shared block at about $r^{0.46}$ unique blocks. The looped models trail by 0.03 to 0.06 nats at $r = 2$, 0.05 to 0.08 at $r = 4$ and 0.09 to 0.12 at $r = 8$; the gap is wider at the lower budgets and flattens at the top of the grid, changing by at most 0.006 nats between 10^{19} and 2.15×10^{19} FLOPs. A check run at about 4×10^{20} FLOPs trails by 0.061 nats at $r = 4$, inside the band measured on the grid. On the anchor's terms this is a penalty that narrows and then flattens with compute, not one that grows, and it is measured on a ladder that does not grow depth - the ladder on which the anchor's own explanation predicts no exponent gain.

Prairie et al. (2026) hold parameters fixed and spend extra compute on loops; their stabilised design improves CORE by 2.99 points over transformer baselines with the same parameter and data budgets at 1.3B parameters and reaches up to 87.5 percent of the quality of a transformer twice the size. That is a parameter-efficiency result. It says nothing about compute-optimal exponents and does not claim to.

Wang et al. (2026b) match per-token FLOPs, non-embedding parameters and key-value cache by looping the middle half of a sparse model twice, fit per-architecture floors, and report a steeper frontier. Their result is on a different family (mixture-of-experts at high sparsity), with own floors that come out in the opposite order to their exponents, and with savings whose interval at the first extrapolated budget, 10^{22} FLOPs, includes zero at the 95 and 97 percent sparsity settings; at 85 percent it is 23.5 percent [8, 42].

Chen et al. (2026) grow depth along the ladder, fit per-arm recipes, share Vanilla's floor, and find that the operator and growth tilt the multiplier while weight sharing alone is a constant penalty - consistent with Schwethelm et al. (2026) on the narrow question both answer: at matched compute, tying weights costs a fixed factor.

So the four studies agree more than their headlines suggest. Looping as weight sharing costs a constant at matched compute (anchor; Schwethelm et al.). Looping as parameter efficiency helps at fixed parameters (Prairie et al.; also the recursive conversions of Bae et al., 2024, and the token-adaptive recursion of Bae et al., 2025, which reports a better perplexity-throughput frontier at equal training FLOPs from 135M to 1.7B parameters). What separates the anchor from the rest is not looping but two other things: growth of executed depth during training, and a boundary operator evaluated on a ladder whose depth grows with compute. The exponent question is about those two, and it is open.

Two further looped results bound what the literature can say. Geiping et al. (2025) trained a 3.5B recurrent-depth model on 800B tokens that improves on reasoning benchmarks as test-time recurrence increases, up to compute equivalent to a 50B-parameter model; Saunshi et al. (2025) showed that a k-layer model looped L times nearly matches a kL-layer model on many synthetic reasoning tasks and that looped and non-looped models scale with effective depth. These are claims about depth as a resource at inference, not about pretraining exponents. Wang et al. (2026a) report that a 1B-parameter hierarchical recurrent model trained on 40B tokens of instruction-response pairs with a task-completion objective performs competitively with 2B to 7B open models; as the anchor notes, the objective and data differ so much from standard pretraining that the contribution of recurrence cannot be isolated.

7. Growth: a schedule is part of the architecture

Growth has a longer history than looping as a training accelerator, and its results have been read as constants. Du et al. (2024) compared four growth operators and found that depthwise stacking reached the loss of a conventionally trained 7B model trained on 300B tokens with 194B tokens, a 54.6 percent speedup, and they noted a debate over whether efficient training strategies that converge faster at first end up no better, or worse, than standard training given ample compute. Kaddour et al. (2023) tested layer stacking on BERT and T5 at fixed compute and found its gains vanished against a baseline whose learning rate was fully decayed. Liew and Kato (2025) found the saturation effect noted in Section 5.4.

The anchor's growth result is stronger than these because it is measured on compute-optimal ladders with per-arm recipes, and it is the largest of its effects: growth over Untied-2 rises from 1.09 to 1.16, and Untied-Grow over Vanilla from 1.30 to 1.55. Two features of the design bear on whether this is an exponent. First, growth changes the parameter count during a run, so the ladder of a grown family is not a ladder of fixed architectures; its "model size" is a schedule. The fitted transition point (one-half to four-fifths of training, fitted per size for the reported growth ladders) and the token budget are re-chosen per architecture, which is the correct control for tuning but means the grown ladder has more free choices per point than the baseline's. Second, the anchor's hypothesis is that growth helps because the share of depth a network cannot yet use early in training rises with depth. If so, the gain is again tied to a ladder whose depth rises with compute, and the same test that turned the operator's exponent gain in the two-pass pair into a constant - fixing executed depth - is the natural test for growth. That test is not reported for the growth arms.

Kaddour et al. (2023) is a warning about baselines, not a contradiction: the anchor's runs are each trained to the end of their own schedule, with a warmdown in the recipe, so the comparison Kaddour et al. warn against is not the one it makes. It is still the closest published case in which a growth method's measured gain was explained by the baseline's training schedule rather than by growth.

8. Depth-repair operators and the ladder they are measured on

The boundary operator belongs to a family of changes aimed at what Sun et al. (2025) call the curse of depth: in pre-normalisation transformers the variance of the residual stream grows with depth, so that deeper blocks act increasingly like the identity and contribute little. Liu et al. (2020) had earlier tied the instability of deep transformer training to how strongly each layer leans on its residual branch, and noted that leaning too weakly

limits what the trained model can reach. Gromov et al. (2024) found that up to half the layers of some open models can be removed with small loss after brief healing, and Csordas et al. (2025) found that layers in the second half of Llama, Qwen and OLMo models contribute much less than those in the first, with no evidence that extra depth is used to compose multi-step results. Remedies include scaling the normalisation output by depth (Sun et al., 2025), learned weighted averages of earlier layers (Pagliardini et al., 2024), learnable connection strengths across depth (Zhu et al., 2024) and attention over earlier layer outputs (Kimi Team, 2026).

Where these remedies report scaling behaviour, they report constants. Kimi Team (2026) state that their scaling experiments show an improvement consistent across model sizes. Schwethelm et al. (2026) find that hyper-connections raise their recurrence-equivalence exponent from 0.46 to 0.65, which they read as a genuine capacity gain, but that is an exponent on recurrence count at fixed depth, not on compute. To the author's knowledge, and by the anchor's own account of prior depth remedies, the anchor is the first study to report a compute-exponent gain from an operator of this kind, and its explanation is the one Section 5.4 examined: the operator rescues a fraction of depth that grows with depth. The anchor's KL effective-depth measurements support the mechanism - grown models reach roughly twice Vanilla's effective depth at 10^{20} FLOPs, and Untied-2 reaches 24 against Deep Vanilla's 20 despite executing the same blocks - and its fixed-depth control, on the two-pass pair, shows the exponent gain disappearing when depth stops growing.

That combination has a precise reading. If the curse of depth worsens with depth and the operator removes it, then on a ladder that grows depth with compute, the baseline's exponent is depressed by a defect that grows along the ladder, and the operator restores it. The measured exponent gap is then the size of the baseline's defect, not a new capability of the variant. On that reading the working rule survives in modified form: the operator does not give the architecture a better exponent than a defect-free baseline would have, it removes a defect whose cost grows along the ladder. The practical conclusion survives too, since practitioners do grow depth with scale and do use pre-norm baselines. What changes is the claim's generality: a different baseline (one with depth-scaled normalisation, say) or a different ladder would be expected to show a smaller gap. No published study runs the operator against such a baseline on a compute-optimal ladder.

9. Two precedents in which the scale trend followed the protocol

9.1 Routing

Clark et al. (2022) fitted scaling laws for routed language models across five orders of magnitude with a fixed dataset of 130B tokens, and found that the improvement from adding experts shrinks as the base model grows, with a size beyond which routing stops helping. Krajewski et al. (2024) argued that this conclusion rested on two implicit assumptions - a fixed training duration and a fixed expert size - and, once tokens and expert granularity were optimised per budget across models from 129M to 3.7B parameters trained on 16B to 130B tokens, found that mixture-of-experts models beat dense ones at every budget with a gap that widens with scale. The anchor cites the second as an instance of an architectural exponent gain. Abnar et al. (2025) found that the best sparsity level itself depends on whether parameters or compute are the binding constraint.

The structure is the one in Section 6. One protocol gave diminishing returns, another gave growing returns, and the difference was in the ladder (tokens fixed versus optimised) and in the recipe (expert size fixed versus tuned). Neither study fitted the comparison with independent floors on a compute axis over a wide window and checked the multiplier against a held-out run far beyond it, so the widening gap is better described as protocol-dependent than as an established exponent difference.

9.2 Optimizers

The second precedent is closer to the anchor, because it involves some of the same authors and the same insistence on recipes. Liu et al. (2025) reported that Muon reaches about twice the computational efficiency of AdamW in compute-optimal training. Qiu et al. (2025) studied how learning rate and weight decay should scale with width and depth for Shampoo, SOAP and Muon, and found that with the right transfer rules these optimizers keep a speedup near 1.4 over AdamW from 190M to 1.4B parameters, while the speedup "vanishes rapidly with scale under incorrect scaling". Wen et al. (2025) tuned ten optimizers by coordinate descent at 0.1B to 0.5B parameters, set the 1.2B hyperparameters of a subset of them with a law, fitted on those smaller runs, that predicts each scaling-sensitive hyperparameter from model size and data budget, and found that the speedup of matrix-based optimizers falls from 1.4 at 0.1B to 1.1 at 1.2B, attributing earlier larger claims to unequal tuning and to evaluating before the end of training. Figure 1b plots the two.

The two studies differ less on facts than on what the baseline should be. Both tune at small scale and carry the settings up by a rule; the rules differ. Qiu et al. (2025) scale the learning rate by μP and independent weight decay as $1/\text{width}$ from a tuned base model, on

compute-optimal ladders; Wen et al. (2025) tune each optimizer from 0.1B to 0.5B and fit a per-optimizer law for each sensitive hyperparameter, at 1 to 8 times the Chinchilla data ratio, and checked the predicted setting against a full sweep at 1.2B only for AdamW, where it came "within $3e-3$ of the optimal configuration". One reading, which neither study tests directly, is that the two rules leave the AdamW baseline and the matrix optimizers at different distances from their optima at the largest size, so the speedups differ. The anchor's appendix adds a third version: Muon's advantage is constant under width-only scaling and shrinks under coupled width-and-depth scaling. An intervention's scale trend, in other words, has been measured as constant or as shrinking depending on how the baseline's hyperparameters and the ladder's shape were chosen. The lesson for architecture is direct. An exponent gap between two arms is an exponent gap between two tuned arms, and its size is bounded by how close each arm's recipe is to optimal at each budget.

10. Other architectures, and the data side

Comparisons of whole architecture families have produced every verdict. Tay et al. (2022) found that among ten architectures under a shared T5-style protocol the vanilla transformer had the steepest slope against FLOPs, that the best model at one scale was not always the best at another, and that some linear-attention models scaled poorly. Shen et al. (2024), training three linear-complexity models and a Llama baseline from 70M to 7B parameters on 300B tokens, found similar scaling to the transformer. Poli et al. (2024), training over 500 models up to 7B, found hybrid designs that beat strong transformer and state-space baselines on compute-optimal and over-trained scaling. Beck et al. (2024) report that xLSTM compares favourably with transformers and state-space models in both performance and scaling. Waleffe et al. (2024) compared 8B Mamba, Mamba-2 and transformer models on the same data up to 3.5T tokens and found pure state-space models behind on copying and in-context tasks while a hybrid was ahead on all twelve standard tasks evaluated. Narang et al. (2021) found that most published transformer modifications did not help when reimplemented in a shared setting, and that those which did tended to come from the same codebase. None of these studies uses per-arm floors, per-arm recipes and a held-out extrapolation together.

On the data side, exponent changes are better established. Sorscher et al. (2022) showed in theory and on image classifiers that a good data-pruning metric can beat power-law scaling in dataset size. Bansal et al. (2022) found back-translated data degraded the data exponent in translation. DeepSeek-AI (2024) found that higher data quality shifts the compute-optimal allocation toward larger models. In the data-constrained regime, Muennighoff et al. (2023) found that up to four epochs

of repetition costs almost nothing and that beyond that the value of extra compute decays toward zero; Kim et al. (2025) found that regularisation and ensembling lower the loss asymptote achievable from fixed data; and Lovelace et al. (2026) modelled the excess loss of repetition with a single overfitting coefficient that strong weight decay reduces by about 70 percent. The anchor's multi-epoch looping result sits in this literature: it is a claim about a regulariser that adds depth without adding parameters to overfit, and it is evaluated against tuned weight decay, which is the right comparator. It is not evidence about the single-epoch exponent.

11. Downstream: the CORE extrapolation

The anchor's downstream forecast chains three fitted maps - compute to task likelihood, likelihood to accuracy, and a sum to the CORE score - and its held-out run lands within 0.002 of the forecast answer likelihood and at 0.387 against a forecast CORE of 0.384. That is good calibration on one point. The broader literature is more cautious. Lourie et al. (2025), in a meta-analysis, found predictable downstream scaling in only 39 percent of cases and found that seemingly benign changes to the setup could change the scaling behaviour completely. Schaeffer et al. (2024) traced part of the difficulty to the transformations that turn likelihoods into multiple-choice accuracy. Gadre et al. (2024), on the other side, predicted the top-1 error averaged over downstream tasks for a 1.4B and a 6.9B model from a power law linking perplexity to error, fitted on experiments using 20 times less compute. The anchor's GPT-3 comparison is labelled indicative by its authors and should be read that way; the controlled downstream claim is the Vanilla-to-Untied-Grow ratio at the GPT-3 reference scores, which the authors project at 2.5 to 3.5 times and which inherits every caveat of the loss extrapolation.

12. Consolidated evidence

Read together, the tables show three patterns. Where a study fixed the ladder's depth, or held other choices at the baseline's, the effect was a constant or a penalty (Schwethelm et al.; the anchor's own fixed-depth and baseline-recipe ablations; Kaplan's per-FLOP comparison). Where the protocol tuned the intervention along the ladder rather than borrowing the baseline's settings, the effect held or widened with scale (Krajewski et al.; Chen et al.; Qiu et al., whose speedup held only under correct scaling). This pattern has two exceptions. Wen et al. also tuned every optimizer, at 0.1B to 0.5B, and carried the settings to 1.2B by a fitted law of their own, and their speedup shrank. SMELT trained the looped arm with the baseline's optimizer and schedule and still reported a saving that widens across its fitted range (Wang et al. 2026b). And where independent floors were fitted, the exponent and floor comparisons came out in opposite directions or were declared not comparable (Wang et al. 2026b; Schwethelm et al.). No

Table 1. Published results on whether an intervention changes a scaling exponent or a constant, one row per cited result. "Floor" is how the irreducible term was treated; "-" means not applicable or not reported in the cited text.

Source	Intervention	Held fixed / protocol	Floor	Window	Reported result
Hestness et al. 2017	LSTM vs RHN, word LM	best size per data shard	-	data 0.1-40% of corpus	text: -0.0656 +/- 1%; summary table: -0.066 vs -0.070
Hestness et al. 2017	SGD vs Adam, char LM	best size per shard	-	data 0.01-4%	-0.0936 vs -0.0954; Adam shifts curve ~5%
Bansal et al. 2022	hybrid, decoder-only NMT	data scaling	-	-	data exponent barely moved; back-translation lowers it
Kaplan et al. 2020	recurrent vs standard transformer	-	-	-	slightly better per parameter, slightly worse per FLOP
Tay et al. 2022	10 architectures	2 ¹⁹ steps, shared optimizer	none (linear)	Tiny-XL	FLOP slope 0.54 vanilla, 0.50 UT, 0.25 Performer
Tay et al. 2022	Universal Transformer	same	none	Tiny-XL	parameter slope 0.56 vs 0.47 vanilla
Clark et al. 2022	routing	130B tokens fixed	-	5 orders of size	routing gain shrinks with size
Krajewski et al. 2024	fine-grained MoE	tokens, granularity tuned	-	129M-3.7B	MoE gap widens with budget
Chen et al. 2026	boundary operator	per-arm recipe, ratio 128	Vanilla's, shared	1e18-1e20	multiplier 1.12 to 1.25
Chen et al. 2026	untied growth	same	shared	1e18-1e20	1.30 to 1.55; over Untied-2 1.09 to 1.16
Chen et al. 2026	untying	same	shared	1e18-1e20	1.08 to 1.06 (constant)
Chen et al. 2026	operator, Vanilla's recipe	baseline hyperparameters	shared	ladder	exponent gain becomes constant
Chen et al. 2026	operator, fixed depth 11 (Untied-2 vs Deep Vanilla)	width-only ladder	shared	ladder	constant, not exponent
Chen et al. 2026	Untied-Grow 7.4B run	8x largest fitted compute	shared	1.23e21	0.03 below forecast; 1.8x vs fitted Vanilla law
Wang et al. 2026b	loop middle half (MoE)	FLOPs, params, KV matched	own: 1.4493 vs 1.4439	4 sizes	gamma 0.250 vs 0.237
Wang et al. 2026b	same, S~97%	same	own	to 1e22	saving 19.6% [0, 51] at 1e22 (extrap.)
Schwethelm et al. 2026	recurrence r = 2, 4, 8	depth 20 fixed	own, 1.56-1.69	4.6e17-2.2e19	trails 0.03-0.12 nats; r ^{0.46}
Schwethelm et al. 2026	r = 4 check run	~20x top of grid	own	~4e20	trails by 0.061 nats
Prairie et al. 2026	stable looping	parameters, data fixed	-	to 1.3B	+ 2.99 CORE vs same-size baseline
Liew and Kato 2025	model reuse / growth	first-stage tokens varied	-	-	second-stage exponent falls with first-stage tokens
Kaddour et al. 2023	layer stacking	fully decayed baseline	-	BERT, T5 budgets	gains vanish
Qiu et al. 2025	matrix optimizers	transferred hyperparameters	-	190M-1.4B	~1.4x, constant
Wen et al. 2025	matrix optimizers	tuned per scale at 0.1-0.5B; 1.2B from fitted hyperparameter law	-	0.1B-1.2B	1.4x falling to 1.1x
Bordelon et al. 2024b	feature learning (theory)	solvable model	-	-	nearly doubles exponent on hard tasks only

study in the tables fits per-arm floors on a compute axis, with per-arm recipes, over a depth-growing ladder, and tests the result with a held-out run far beyond the window.

Algorithm 1 states the classification this paper argues a reported scaling advantage should pass through before it is called an exponent change.

Applied to the anchor study's published results, the procedure stops at step 3 for the operator - its own fixed-depth control, run on the two-pass pair Untied-2 against Deep Vanilla rather than on Operator-1 against Vanilla,

returns a constant - so the operator's label is "ladder-coupled", conditional on its recipe. For growth, steps 3 and 4 are not reported, so the label is "undetermined" on present evidence, with a held-out point (step 5) that is consistent with both candidate readings. For SMELT, step 4 with per-arm floors gives a steeper exponent together with a higher floor, with no reported uncertainty on either difference; the label is "undetermined". For Schwethelm et al., the label is CONSTANT at the top of the grid - a penalty that narrows and then flattens - on a fixed-depth ladder. None of these labels says the anchor is wrong. They say what it has shown.

Table 2. Eleven designs mapped onto the five indices of Section 5. "Grows" means the ladder's depth rises with compute.

Study	Axis	Floor	Recipe	Ladder	Window
Hestness et al. 2017	data	not separated	-	best size per shard	~2.6 decades data
Kaplan et al. 2020 (UT)	N and C	none	shared	-	-
Tay et al. 2022	FLOPs and N	none	shared	fixed steps	4-5 sizes
Clark et al. 2022	N, experts	none	shared	tokens fixed	5 orders N
Krajewski et al. 2024	compute	fitted	tuned	tokens tuned	129M-3.7B
Chen et al. 2026	FLOPs, executed	shared	per arm	ratio 128, grows	2 decades
Wang et al. 2026b	FLOPs	per arm	shared (baseline's optimizer and schedule; shape re-matched)	sparsity grid	4 sizes
Schwethelm et al. 2026	FLOPs	per arm	muP transfer	depth fixed	1.7 decades
Prairie et al. 2026	FLOPs at fixed N	-	-	parameters fixed	to 1.3B
Qiu et al. 2025	compute	-	transferred	width and depth	190M-1.4B
Wen et al. 2025	compute	-	tuned 0.1-0.5B; fitted law at 1.2B	1-8x Chinchilla data	0.1B-1.2B

Algorithm 1. Classifying a reported scaling advantage of variant B over baseline A

Input: ladders for A and B; per-point compute (executed FLOPs), loss, shape, tokens step 1, axis: if compute is not counted on executed FLOPs for both arms, restate on that axis; report parameter-axis results separately. step 2, recipe: if B or A uses the other's hyperparameters, or one rule fitted on small models only, refit per arm at ≥ 2 scales; record the multiplier under each recipe. step 3, ladder: record how depth, width and tokens grow with compute. If depth grows, rerun one arm pair at fixed executed depth. If the trend vanishes at fixed depth, label "ladder-coupled". step 4, floor: fit $L = E + A * C^{(-g)}$ per arm with E free; bootstrap over runs and seeds. Also fit with E shared. if $g_B - g_A$ excludes 0 and $E_B - E_A$ includes 0: candidate EXPONENT elif $E_B - E_A$ excludes 0 and $g_B - g_A$ includes 0: candidate FLOOR elif both include 0 and multiplier is flat: CONSTANT else: UNDETERMINED (window too narrow to separate g from E) step 5, window: predict the multiplier at a held-out budget $\geq 10\times$ the largest fitted one under each candidate; train it; keep the candidates whose interval contains the measurement. step 6, report: label in {CONSTANT, EXPONENT, FLOOR, LADDER-COUPLED, UNDETERMINED}, with the recipe and ladder it holds for.

13. The strongest case for the exponent reading

The case for reading the anchor's result as a real exponent change is strong enough to state at full force. First, the multipliers rise for every variant that includes growth or the operator, and on FineWeb the rise is interpolated between trained models, not extrapolated; the FineWeb-Edu ladders show the same direction. Second, the held-out 7.4B run beat its forecast rather than missing it, which is what one would expect if the variant's fitted law were, if anything, conservative. Third, the floor objection cuts both ways: if the true story were Case 3, a lower floor, the advantage at very large budgets would be larger than the exponent reading predicts, not smaller. Fourth, the mechanism is plausible and measured: effective depth rises with the operator and with growth, and the curse of depth is documented across model families (Sun et al., 2025; Csordas et al., 2025). Fifth, the anchor fitted a recipe for each arm, which is more than most architecture studies do, and

checked that adding output-multiplier or weight-decay scaling to Vanilla's learning-rate rule changes Vanilla's own multiplier by at most about 3 percent.

Each point holds, and together they establish that the operator and growth buy an advantage that grows across the measured window and probably beyond it. What they do not establish is the specific claim in the title - that the exponent changed - as distinct from the floor changing or the baseline's ladder carrying a defect that grows with depth. The third point is the important one for practice: a practitioner who cares only about whether the advantage grows does not need to distinguish Case 2 from Case 3. A practitioner who wants to know how large it will be at 10^{25} does, because the two cases give different numbers, and a theorist who wants to know whether architecture can change exponents does, because only Case 2 answers that question.

14. Studies that would settle the open part

Four controls would separate the readings. The first three need no more compute than the anchor study already used; the fourth needs about eight times its largest run.

The first is floor-free comparison. Fit per-arm floors on the anchor's ladders, as Wang et al. (2026b) and Schwethelm et al. (2026) did, and bootstrap them over seeds and runs, which neither did. If the exponent gap survives with floors free and the floor gap includes zero, the exponent reading is supported; if the reverse, the floor reading is. If both are undetermined, the window is too short and the claim should be stated as a rising multiplier.

The second is a depth-repaired baseline. Rerun Vanilla with a depth-aware normalisation of the kind Sun et al. (2025) propose, or with the anchor's own operator removed but depth-scaled residual multipliers fitted by a complete parameterisation such as that of Mlodozieniec et al. (2025), and compare Operator-1 against it. If the operator's exponent gain is a correction for a baseline defect that grows with depth, this gap should shrink toward a constant.

The third is a ladder swap for growth. Repeat for the growth arms the fixed-depth, width-only control that turned the operator's exponent gain in the two-pass pair (Untied-2 against Deep Vanilla) into a constant. If growth's advantage over Untied-2 stays rising on a fixed-depth ladder, growth carries an effect beyond depth rescue.

The fourth is a second held-out point, with a matching Vanilla run. The 7.4B run is at eight times the largest fitted budget, and has no Vanilla partner. A pair of runs at about 10^{22} FLOPs would separate Case 2 from Case 3 more sharply than any amount of refitting inside the window, because the two predict multipliers that diverge with distance. Stating the predicted multiplier under each case before training would make the result hard to explain away.

Beyond the anchor, the routing and optimizer precedents suggest a standing protocol for any architectural scaling claim: report the multiplier under the variant's recipe and under the baseline's, under a depth-growing and a fixed-depth ladder, with floors shared and free, and with one held-out pair an order of magnitude beyond the window. None of the studies in Table 2 does all four.

15. What is not known

Whether any architectural intervention changes the compute exponent of language-model pretraining, as distinct from its floor or its interaction with a particular ladder, is not known. Whether the anchor's operator would show an exponent gain against a baseline without the curse of depth is not known. Whether growth's advantage depends on depth growing along the ladder is

not known. No study in Table 1 reports an uncertainty on the difference between two architectures' fitted floors. Whether the 2.7 times multiplier projected at 10^{25} FLOPs is reachable is not known; it is a model-dependent extrapolation over five decades from fits over two. Whether looped mixture-of-experts models keep their advantage beyond the fitted window is not known; at the first extrapolated budget the published interval includes no advantage at the 95 and 97 percent sparsity settings, and at 85 percent it is 23.5 percent [8, 42]. And whether exponents measured on single-epoch FineWeb transfer to the multi-epoch, data-constrained regime where the anchor finds a different, regularisation-driven benefit from looping is not known.

16. Limitations

This analysis synthesises published text and tables. The derived exponent ratios in Section 4 assume pure power laws over a shared floor, which is the assumption the paper questions, and they are stated to show the scale of the effect a decision must resolve, not as estimates. Where the anchor states fitted exponents in its text (Appendix C.1: 0.1168 against 0.1113 on FineWeb for Untied-Grow and Vanilla), the analysis uses them; for the other arms it relies on the multipliers the anchor reports in its text. The 2026 looped-model studies are preprints and could change. The paper treats the anchor's appendix ablations as decisive for the ladder argument; they are single comparisons, and the anchor's authors read the fixed-depth result as support for their mechanism rather than as a qualification of their exponent claim, a reading this paper argues is compatible with its own. Finally, the five indices are not exhaustive: batch size, which the anchor holds fixed and names as out of scope, and data mixture, which Mayilvahanan et al. (2025) show sets loss-to-loss trends, are further candidates.

17. Conclusion

The working rule that architecture moves constants and not exponents was an induction from the interventions that happened to be tested, and the theory of scaling exponents does not forbid architecture from mattering. The September 2026 evidence that a boundary operator and depth growth produce compute multipliers that rise with scale is measured and careful. But a rising multiplier over two decades is what an exponent change, a floor change, and a depth defect that grows along the ladder all produce, and the anchor's own ablations show the boundary operator's exponent gain vanishing under the baseline's recipe and, in the two-pass pair, becoming a constant at fixed depth. The looped-model studies that reach the opposite verdict measured weight sharing at fixed depth, which every study agrees costs a constant. Routing and optimizers have already shown that the sign of a scale trend can flip with the protocol. A steeper slope, on one ladder, against one floor, is evidence that

something grows with scale. It is not yet a new law, and the four controls in Section 14 would say which one it is.

References

- Abnar, S., Shah, H., Busbridge, D., Ali, A. M. E., Susskind, J., & Thilak, V. (2025). Parameters vs FLOPs: Scaling Laws for Optimal Sparsity for Mixture-of-Experts Language Models. arXiv:2501.12370. doi:10.48550/arXiv.2501.12370.
- Alabdulmohsin, I., Neyshabur, B., & Zhai, X. (2022). Revisiting Neural Scaling Laws in Language and Vision. NeurIPS 2022. arXiv:2209.06640. doi:10.48550/arXiv.2209.06640.
- Bae, S., Fisch, A., Harutyunyan, H., Ji, Z., Kim, S., & Schuster, T. (2024). Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. ICLR 2025. arXiv:2410.20672. doi:10.48550/arXiv.2410.20672.
- Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., et al. (2025). Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. arXiv:2507.10524. doi:10.48550/arXiv.2507.10524.
- Bahri, Y., Dyer, E., Kaplan, J., Lee, J., & Sharma, U. (2024). Explaining Neural Scaling Laws. PNAS 121(27). arXiv:2102.06701. doi:10.48550/arXiv.2102.06701.
- Bansal, Y., Ghorbani, B., Garg, A., Zhang, B., Krikun, M., Cherry, C., Neyshabur, B., & Firat, O. (2022). Data Scaling Laws in NMT: The Effect of Noise and Architecture. arXiv:2202.01994. doi:10.48550/arXiv.2202.01994.
- Beck, M., Poeppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., & Hochreiter, S. (2024). xLSTM: Extended Long Short-Term Memory. arXiv:2405.04517. doi:10.48550/arXiv.2405.04517.
- Besiroglu, T., Erdil, E., Barnett, M., & You, J. (2024). Chinchilla Scaling: A replication attempt. arXiv:2404.10102. doi:10.48550/arXiv.2404.10102.
- Bordelon, B., Atanasov, A., & Pehlevan, C. (2024a). A Dynamical Model of Neural Scaling Laws. ICML 2024. arXiv:2402.01092. doi:10.48550/arXiv.2402.01092.
- Bordelon, B., Atanasov, A., & Pehlevan, C. (2024b). How Feature Learning Can Improve Neural Scaling Laws. ICLR 2025. arXiv:2409.17858. doi:10.48550/arXiv.2409.17858.
- Brandfonbrener, D., Anand, N., Vyas, N., Malach, E., & Kakade, S. (2024). Loss-to-Loss Prediction: Scaling Laws for All Datasets. arXiv:2411.12925. doi:10.48550/arXiv.2411.12925.
- Caballero, E., Gupta, K., Rish, I., & Krueger, D. (2022). Broken Neural Scaling Laws. ICLR 2023. arXiv:2210.14891. doi:10.48550/arXiv.2210.14891.
- Chen, Z., Vegesna, A., Dahal, S., & Wilson, A. G. (2026). How Model Growth, Recursion, and Boundary Operators Influence Scaling Exponents. arXiv:2609.19107. doi:10.48550/arXiv.2609.19107.
- Choshen, L., Zhang, Y., & Andreas, J. (2024). A Hitchhiker's Guide to Scaling Law Estimation. ICML 2025. arXiv:2410.11840. doi:10.48550/arXiv.2410.11840.
- Clark, A., de las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., et al. (2022). Unified Scaling Laws for Routed Language Models. arXiv:2202.01169. doi:10.48550/arXiv.2202.01169.
- Csordas, R., Manning, C. D., & Potts, C. (2025). Do Language Models Use Their Depth Efficiently? NeurIPS 2025. arXiv:2505.13898. doi:10.48550/arXiv.2505.13898.
- DeepSeek-AI (2024). DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. arXiv:2401.02954. doi:10.48550/arXiv.2401.02954.
- Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2019). Universal Transformers. ICLR 2019. arXiv:1807.03819. doi:10.48550/arXiv.1807.03819.
- Du, W., Luo, T., Qiu, Z., Huang, Z., Shen, Y., Cheng, R., Guo, Y., & Fu, J. (2024). Stacking Your Transformers: A Closer Look at Model Growth for Efficient LLM Pre-Training. NeurIPS 2024. arXiv:2405.15319. doi:10.48550/arXiv.2405.15319.
- Everett, K., Xiao, L., Wortsman, M., Alemi, A. A., Novak, R., Liu, P. J., et al. (2024). Scaling Exponents Across Parameterizations and Optimizers. ICML 2024. arXiv:2407.05872. doi:10.48550/arXiv.2407.05872.
- Gadre, S. Y., Smyrnis, G., Shankar, V., Gururangan, S., Wortsman, M., Shao, R., et al. (2024). Language models scale reliably with over-training and on downstream tasks. arXiv:2403.08540. doi:10.48550/arXiv.2403.08540.
- Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., & Goldstein, T. (2025). Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171. doi:10.48550/arXiv.2502.05171.
- Gromov, A., Tirumala, K., Shapourian, H., Gloriosio, P., & Roberts, D. A. (2024). The Unreasonable Ineffectiveness of the Deeper Layers. ICLR 2025. arXiv:2403.17887. doi:10.48550/arXiv.2403.17887.
- Hagele, A., Bakouch, E., Kosson, A., Ben Allal, L., Von Werra, L., & Jaggi, M. (2024). Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations. NeurIPS 2024. arXiv:2405.18392. doi:10.48550/arXiv.2405.18392.
- Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., et al. (2020). Scaling Laws for Autoregressive Generative Modeling. arXiv:2010.14701. doi:10.48550/arXiv.2010.14701.
- Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv:1712.00409. doi:10.48550/arXiv.1712.00409.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., et al. (2022). Training Compute-Optimal Large Language Models. arXiv:2203.15556. doi:10.48550/arXiv.2203.15556.
- Hutter, M. (2021). Learning Curve Theory. arXiv:2102.04074. doi:10.48550/arXiv.2102.04074.
- Ivgi, M., Carmon, Y., & Berant, J. (2022). Scaling Laws Under the Microscope: Predicting Transformer Performance from Small Scale Experiments. Findings of EMNLP 2022. arXiv:2202.06387. doi:10.48550/arXiv.2202.06387.
- Kaddour, J., Key, O., Nawrot, P., Minervini, P., & Kusner, M. J. (2023). No Train No Gain: Revisiting Efficient Training Algorithms For Transformer-based Language Models. NeurIPS 2023. arXiv:2307.06440. doi:10.48550/arXiv.2307.06440.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling Laws for Neural Language Models. arXiv:2001.08361. doi:10.48550/arXiv.2001.08361.
- Kim, K., Kotha, S., Liang, P., & Hashimoto, T. (2025). Pre-training under infinite compute. arXiv:2509.14786. doi:10.48550/arXiv.2509.14786.
- Kimi Team, Chen, G., Zhang, Y., Su, J., Xu, W., Pan, S., et al. (2026). Attention Residuals. arXiv:2603.15031. doi:10.48550/arXiv.2603.15031.
- Krajewski, J., Ludziejewski, J., Adamczewski, K., Pioro, M., Krutul, M., Antoniak, S., et al. (2024). Scaling Laws for Fine-Grained Mixture of Experts. arXiv:2402.07871. doi:10.48550/arXiv.2402.07871.
- Levine, Y., Wies, N., Sharir, O., Bata, H., & Shashua, A. (2020). The Depth-to-Width Interplay in Self-Attention. NeurIPS 2020. arXiv:2006.12467. doi:10.48550/arXiv.2006.12467.
- Li, M., Kudugunta, S., & Zettlemoyer, L. (2025). (Mis)Fitting: A Survey of Scaling Laws. ICLR 2025. arXiv:2502.18969. doi:10.48550/arXiv.2502.18969.
- Liew, S. P., & Kato, T. (2025). Reusing Overtrained Language Models Saturates Scaling. arXiv:2510.06548. doi:10.48550/arXiv.2510.06548.
- Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., et al. (2025). Muon is Scalable for LLM Training. arXiv:2502.16982. doi:10.48550/arXiv.2502.16982.

- Liu, L., Liu, X., Gao, J., Chen, W., & Han, J. (2020). Understanding the Difficulty of Training Transformers. EMNLP 2020. arXiv:2004.08249. doi:10.48550/arXiv.2004.08249.
- Liu, Z., Zhao, C., Iandola, F., Lai, C., Tian, Y., Fedorov, I., et al. (2024). MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. ICML 2024. arXiv:2402.14905. doi:10.48550/arXiv.2402.14905.
- Lourie, N., Hu, M. Y., & Cho, K. (2025). Scaling Laws Are Unreliable for Downstream Tasks: A Reality Check. Findings of EMNLP 2025. arXiv:2507.00885. doi:10.48550/arXiv.2507.00885.
- Lovelace, J., Belardi, C., Kundurthy, S., Sudhakar, S., & Weinberger, K. Q. (2026). Prescriptive Scaling Laws for Data Constrained Training. arXiv:2605.01640. doi:10.48550/arXiv.2605.01640.
- Maloney, A., Roberts, D. A., & Sully, J. (2022). A Solvable Model of Neural Scaling Laws. arXiv:2210.16859. doi:10.48550/arXiv.2210.16859.
- Mayilvahanan, P., Wiedemer, T., Mallick, S., Bethge, M., & Brendel, W. (2025). LLMs on the Line: Data Determines Loss-to-Loss Scaling Laws. ICML 2025. arXiv:2502.12120. doi:10.48550/arXiv.2502.12120.
- Michaud, E. J., Liu, Z., Girit, U., & Tegmark, M. (2023). The Quantization Scaling Model of Neural Scaling. NeurIPS 2023. arXiv:2303.13506. doi:10.48550/arXiv.2303.13506.
- Mlodozieniec, B., Ablin, P., Bethune, L., Busbridge, D., Klein, M., Ramapuram, J., & Cuturi, M. (2025). Completed Hyperparameter Transfer across Modules, Width, Depth, Batch and Duration. arXiv:2512.22382. doi:10.48550/arXiv.2512.22382.
- Muennighoff, N., Rush, A. M., Barak, B., Le Scao, T., Piktus, A., Tazi, N., Pyysalo, S., Wolf, T., & Raffel, C. (2023). Scaling Data-Constrained Language Models. arXiv:2305.16264. doi:10.48550/arXiv.2305.16264.
- Narang, S., Chung, H. W., Tay, Y., Fedus, W., Fevry, T., Matena, M., et al. (2021). Do Transformer Modifications Transfer Across Implementations and Applications? EMNLP 2021. arXiv:2102.11972. doi:10.48550/arXiv.2102.11972.
- Pagliardini, M., Mohtashami, A., Fleuret, F., & Jaggi, M. (2024). DenseFormer: Enhancing Information Flow in Transformers via Depth Weighted Averaging. arXiv:2402.02622. doi:10.48550/arXiv.2402.02622.
- Paquette, E., Paquette, C., Xiao, L., & Pennington, J. (2024). 4 + 3 Phases of Compute-Optimal Neural Scaling Laws. arXiv:2405.15074. doi:10.48550/arXiv.2405.15074.
- Pearce, T., & Song, J. (2024). Reconciling Kaplan and Chinchilla Scaling Laws. TMLR 2024. arXiv:2406.12907. doi:10.48550/arXiv.2406.12907.
- Poli, M., Thomas, A. W., Nguyen, E., Ponnusamy, P., Deiseroth, B., Kersting, K., et al. (2024). Mechanistic Design and Scaling of Hybrid Architectures. arXiv:2403.17844. doi:10.48550/arXiv.2403.17844.
- Porian, T., Wortsman, M., Jitsev, J., Schmidt, L., & Carmon, Y. (2024). Resolving Discrepancies in Compute-Optimal Scaling of Language Models. NeurIPS 2024. arXiv:2406.19146. doi:10.48550/arXiv.2406.19146.
- Prairie, H., Novack, Z., Berg-Kirkpatrick, T., & Fu, D. Y. (2026). Parcae: Scaling Laws For Stable Looped Language Models. arXiv:2604.12946. doi:10.48550/arXiv.2604.12946.
- Qiu, S., Chen, Z., Phan, H., Lei, Q., & Wilson, A. G. (2025). Hyperparameter Transfer Enables Consistent Gains of Matrix-Preconditioned Optimizers Across Scales. NeurIPS 2025. arXiv:2512.05620. doi:10.48550/arXiv.2512.05620.
- Saunshi, N., Dikkala, N., Li, Z., Kumar, S., & Reddi, S. J. (2025). Reasoning with Latent Thoughts: On the Power of Looped Transformers. ICLR 2025. arXiv:2502.17416. doi:10.48550/arXiv.2502.17416.
- Schaeffer, R., Schoelkopf, H., Miranda, B., Mukobi, G., Madan, V., Ibrahim, A., Bradley, H., Biderman, S., & Koyejo, S. (2024). Why Has Predicting Downstream Capabilities of Frontier AI Models with Scale Remained Elusive? arXiv:2406.04391. doi:10.48550/arXiv.2406.04391.
- Schwethelm, K., Rueckert, D., & Kaissis, G. (2026). How Much Is One Recurrence Worth? Iso-Depth Scaling Laws for Looped Language Models. arXiv:2604.21106. doi:10.48550/arXiv.2604.21106.
- Sharma, U., & Kaplan, J. (2020). A Neural Scaling Law from the Dimension of the Data Manifold. arXiv:2004.10802. doi:10.48550/arXiv.2004.10802.
- Shen, X., Li, D., Leng, R., Qin, Z., Sun, W., & Zhong, Y. (2024). Scaling Laws for Linear Complexity Language Models. arXiv:2406.16690. doi:10.48550/arXiv.2406.16690.
- Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., & Morcos, A. S. (2022). Beyond neural scaling laws: beating power law scaling via data pruning. NeurIPS 2022. arXiv:2206.14486. doi:10.48550/arXiv.2206.14486.
- Sun, W., Song, X., Li, P., Yin, L., Zheng, Y., & Liu, S. (2025). The Curse of Depth in Large Language Models. NeurIPS 2025. arXiv:2502.05795. doi:10.48550/arXiv.2502.05795.
- Tay, Y., Dehghani, M., Rao, J., Fedus, W., Abnar, S., Chung, H. W., Narang, S., Yogatama, D., Vaswani, A., & Metzler, D. (2021). Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers. ICLR 2022. arXiv:2109.10686. doi:10.48550/arXiv.2109.10686.
- Tay, Y., Dehghani, M., Abnar, S., Chung, H. W., Fedus, W., Rao, J., Narang, S., Tran, V. Q., Yogatama, D., & Metzler, D. (2022). Scaling Laws vs Model Architectures: How does Inductive Bias Influence Scaling? arXiv:2207.10551. doi:10.48550/arXiv.2207.10551.
- Waleffe, R., Byeon, W., Riach, D., Norick, B., Korthikanti, V., Dao, T., Gu, A., et al. (2024). An Empirical Study of Mamba-based Language Models. arXiv:2406.07887. doi:10.48550/arXiv.2406.07887.
- Wang, G., Liu, C., Wang, C., Zhou, C., Sun, Y., Wu, Y., Zhen, S., Scimeca, L., & Abbasi Yadkori, Y. (2026a). HRM-Text: Efficient Pretraining Beyond Scaling. arXiv:2605.20613. doi:10.48550/arXiv.2605.20613.
- Wang, S., Zhang, G., Luo, K., Wu, Y., Liu, S., Liu, J., Huang, W., Yan, S., & Li, J. (2026b). SMELT: Scaling Laws for Compute-Matched MoE Looped Transformers. arXiv:2609.01343. doi:10.48550/arXiv.2609.01343.
- Wen, K., Hall, D., Ma, T., & Liang, P. (2025). Fantastic Pretraining Optimizers and Where to Find Them. arXiv:2509.02046. doi:10.48550/arXiv.2509.02046.
- Yang, G., Hu, E. J., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., & Gao, J. (2022). Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. arXiv:2203.03466. doi:10.48550/arXiv.2203.03466.
- Zhu, D., Huang, H., Huang, Z., Zeng, Y., Mao, Y., Wu, B., Min, Q., & Zhou, X. (2024). Hyper-Connections. arXiv:2409.19606. doi:10.48550/arXiv.2409.19606.