Lecture 11 — Scaling in Practice: Case Studies, Optimizers & μP

Language Modeling from Scratch · lecture notes · rebuilt for NVIDIA Blackwell, late 2026

Section 1: Breaking the $O(n^2)$ Scaling-Sweep Bottleneck with WSD and Multi-Step Schedules

Fitting a theoretical power law to a handful of small language-model training runs is straightforward. Choosing a learning-rate schedule, batch size, learning rate, optimizer, and model parametrization that survive a $100\times$ to $1000\times$ extrapolation in compute is not. Recall that classical scaling laws—Kaplan et al. (2020) and Chinchilla (2022)—focused on dense Transformers trained with cosine learning-rate decay and AdamW, asking primarily how to split a compute budget C between parameter count N and training tokens D. Between those 2020–2022 baselines and late-2026 frontier pretraining, scaling practice expanded from fitting (N, D) on dense cosine runs to jointly predicting learning rate, batch size, Mixture-of-Experts (MoE) sparsity, attention architecture, and numerical stability.

To see the full arc of this lecture, consider four tightly coupled engineering questions that follow the progression from 2024 historical recipes (MiniCPM, DeepSeek LLM, Llama 3, Hunyuan-Large, Qwen 2.5) through 2025 MoE and Muon milestones (MiniMax-01, Qwen 3, Kimi K2) to current late-2026 frontier systems (DeepSeek-V4, Kimi K3/K3.1, and Qwen3.8-Next):

1. **Scaling recipes:** Does Chinchilla's scaling recipe actually work on real dense and MoE runs, and can we cut sweep compute instead of retraining from scratch for every token horizon?
2. **Hyperparameter laws:** How do we set learning rate η and batch size B as N, D, and sparsity scale, and when do clean small-scale fits blow up?
3. **Optimizers and Muon:** Which optimizers actually hold their advantage at scale and prevent training blow-ups at frontier compute budgets?
4. **μP theory and 2026 extensions:** How can we parametrize a network so optimal hyperparameters transfer cleanly across width, depth, token count, and MoE routing?

The very first bottleneck appears before you even fit a single scaling curve: measuring how validation loss behaves along the data axis D is prohibitively expensive under a cosine learning-rate schedule.

Chinchilla (2022) demonstrated empirically that for a cosine learning-rate schedule to achieve minimum validation loss at a target token count T, the cosine cycle length must match that exact target horizon T. Consider two stacked plots comparing cosine schedules of different planned lengths—$1.0\times$ steps, $2.0\times$ steps, and $5.0\times$ steps—evaluated at a common vertical reference line marking horizon T_1. In the top plot (learning rate normalized by peak LR on the vertical axis, training steps on the horizontal axis), all three curves warm up steeply to the same maximum learning rate, then decay along different cosine arcs. The $1.0\times$ curve finishes its decay right at T_1, reaching its minimum learning rate there. The $2.0\times$ curve

is only partway through its decay at T_1, and the $5.0\times$ curve is still sitting near its peak learning rate at T_1. In the bottom plot (validation loss on the vertical axis, training tokens on the horizontal axis), all three curves start at the same high loss and drop steeply, but at the vertical evaluation line T_1, they separate cleanly: the $1.0\times$ curve achieves the lowest validation loss right as its cycle completes, the $2.0\times$ curve sits noticeably higher, and the $5.0\times$ curve sits higher still because its learning rate has barely decayed.

Because an unfinished cosine run keeps the learning rate too high at intermediate horizons, you cannot train one long cosine run and extract intermediate checkpoints as compute-optimal data points for a scaling-law fit. Instead, to evaluate k distinct token horizons $D_1, D_2, \ldots, D_k$ for a single model size, you must launch k separate training runs from random initialization. When those horizons are spaced across n increments, the total token cost scales as $\sum_{j=1}^{n} j = \frac{n(n+1)}{2} = O(n^2)$ rather than $O(n)$.

COMMON MISCONCEPTION

You might think that saving checkpoints every $10N$ tokens during a single $60N$-token cosine run lets you plot the optimal validation loss at $10N, 20N, \ldots, 60N$ for free. That breaks because at any intermediate token count $T_1 < 60N$, the cosine schedule has not annealed the learning rate down to its floor; the checkpoint at T_1 has systematically worse validation loss than a model whose cosine cycle ended at T_1, biasing every fitted scaling-law exponent.

Modern scaling recipes break this horizon coupling by decoupling the long bulk of training from the final annealing phase. Two schedules accomplish this in practice: Warmup–Stable–Decay (WSD), popularized by MiniCPM (2024), and multi-step decay, used by DeepSeek LLM (2024).

DEFINITION

Warmup–Stable–Decay (WSD) schedule. A three-phase learning-rate schedule consisting of a short linear warmup, a long constant-learning-rate stable phase that can run indefinitely, and a brief final decay phase (typically around 10% of total training tokens) during which the learning rate anneals rapidly to its minimum.

Visualize the WSD schedule against a cosine reference on a plot of learning rate versus training tokens. A dashed cosine curve labeled $\mathrm{Cosine}(40N)$ rises during warmup and smoothly curves downward from peak LR to its floor at $40N$ tokens. By contrast, WSD rises during the same short warmup window and then follows a flat horizontal trunk—the shared stable phase—extending across the entire token axis ($40N, 60N, 80N$, and beyond). Whenever you want an optimal data point at a target horizon—say $40N$, $60N$, or $80N$—you branch off from an earlier checkpoint on that shared stable trunk and run only the final $\sim 10\%$ decay phase: for example, $\mathrm{WSD}(40N, 4N)$ branches at $36N$ and decays for $4N$ tokens; $\mathrm{WSD}(80N, 8N)$ branches at $72N$ and decays for $8N$ tokens. Because the stable trunk is computed once and reused across every horizon, measuring the entire data axis drops from quadratic cost $O(n^2)$ to linear cost $O(mC)$ in the target compute budget (for a small branching overhead factor m). As an operational bonus, WSD also lets production teams continue pretraining a model past its initial token budget directly from the pre-decay checkpoint without starting over.

DeepSeek LLM (2024) achieved the same horizon-decoupling effect with a discrete **multi-step schedule**: 2,000 warmup steps, followed by a constant learning rate across most of training, dropping to 31.6% of the maximum learning rate at 80% of total tokens and to 10% of the maximum learning rate at 90% of total tokens (with gradient clipping set to 1.0). Notice that $31.6\% \approx 1/\sqrt{10}$, so each drop divides the learning rate by roughly $\sqrt{10}$.

WORKED EXAMPLE

Worked Example: Sweep Cost of Cosine vs. WSD Across Six Token Horizons.

Suppose we want optimal validation-loss measurements for a fixed model of size N at six token horizons: $D \in \{10N, 20N, 30N, 40N, 50N, 60N\}$.

1. **Cosine schedule (retraining from scratch for each horizon):** Each horizon requires its own independent run whose cosine cycle ends at D_k. The total number of tokens processed is:

$$D_{\text{cosine}} = 10N + 20N + 30N + 40N + 50N + 60N = 210N.$$

2. **WSD schedule (shared stable trunk + 10% decay branches):** To evaluate horizon D_k with a 10% decay phase, the model trains on the shared stable trunk for $0.9D_k$ tokens and then branches off for $0.1D_k$ decay tokens. The longest stable prefix needed is $0.9 \times 60N = 54N$, and every shorter horizon's stable prefix ($9N, 18N, 27N, 36N, 45N$) is simply an earlier checkpoint saved along that single $54N$ run. The six separate decay branches cost:

$$D_{\text{decays}} = 1N + 2N + 3N + 4N + 5N + 6N = 21N.$$

   Summing the single shared stable trunk and all six decay branches gives:

$$D_{\text{WSD}} = 54N + 21N = 75N.$$

3. **Interpretation:** Even with only six horizons, WSD reduces total sweep tokens from $210N$ to $75N$—a $2.80\times$ compute reduction. If we evaluated twelve horizons up to $120N$, cosine would require $780N$ tokens while WSD would require $108N + 78N = 186N$ tokens (a $4.19\times$ reduction), because the shared trunk scales only with the maximum horizon rather than the sum of all horizons.

A subtle dynamic appears when you inspect the validation loss curve of a WSD run. Plotting validation loss against tokens processed for $\text{Cosine}(80N)$ alongside WSD shows two distinct regimes. Throughout the stable phase, the WSD loss curve sits noticeably *above* the smoothly falling $\text{Cosine}(80N)$ curve—it looks as though WSD is stalling on a slow plateau. The moment a decay branch begins, however, the validation loss plummets sharply downward: at $40N$, $60N$, and $80N$, each $\sim 10\%$ decay branch drops rapidly from the high stable trunk straight down to match—or slightly beat—the cosine run that trained specifically for that horizon.

How long must the decay phase be to close the gap? MiniCPM tested shorter $2N$ decay phases against $4N$, $6N$, and $8N$ decay lengths (corresponding to 10% of $40N$, $60N$, and $80N$) and found that roughly 10% decay is needed for the loss to fully catch up to cosine. DeepSeek LLM similarly confirmed that multi-step schedules split as 80/10/10, 70/15/15, and 60/20/20 (percent of tokens at 100%, 31.6%, and 10% LR) all match cosine final validation loss.

COMMON MISCONCEPTION

You might think that because a WSD model has higher validation loss than a cosine model halfway through training, the WSD run is underperforming and should be aborted. That breaks because high constant learning rates prevent the optimizer from settling into sharp local basins during the stable phase; the model is still building representations, and the loss drop is deferred until annealing. You should never judge a WSD run mid-plateau—evaluate only after the decay phase completes.

KEY IDEA

- Cosine schedules couple the learning-rate decay trajectory to a single predetermined token horizon T, making multi-horizon scaling sweeps cost $O(n^2)$.
- WSD (shared constant trunk plus $\sim 10\%$ decay branches) and DeepSeek-style multi-step schedules match cosine final loss while cutting data-axis sweep cost to linear $O(mC)$ and allowing continued pretraining without restarts.

Cutting sweep cost along the data axis solves only half of the practical recipe. When you sweep across multiple model sizes N as well as data sizes D, the optimal learning rate and batch size change with scale—unless you control how model parameters and updates scale. We turn next to the first complete end-to-end case study that pairs WSD with parametrization-invariant scaling.

✎ Check Your Understanding

1. **Yes/No with justification:** Can you estimate the compute-optimal validation loss at $20N$ tokens by taking the $20N$ checkpoint from a cosine run configured for $100N$ total tokens?
   Answer: No. Because the $100N$ cosine cycle has barely begun decaying at $20N$ tokens, the learning rate is still far too high and the checkpoint's validation loss is systematically worse than a run whose cosine cycle finished at $20N$.
2. **Quick computation:** Under a WSD schedule with a 10% decay phase, how many total tokens are required to evaluate a single model size N at two horizons, $40N$ and $80N$?
   Answer: The shared stable trunk runs to $0.9 \times 80N = 72N$ tokens (capturing the $36N$ checkpoint along the way), and the two decay branches add $4N + 8N = 12N$ tokens, for a total of $84N$ tokens (compared to $120N$ for two separate cosine runs).

Section 2: Parametrization-Invariant Scaling in Practice — The MiniCPM Case Study

Even with WSD eliminating $O(n^2)$ retraining along the token axis, scaling across model sizes introduces a second combinatorial explosion: if every model width d_m has a different optimal learning rate, then every model in your scaling sweep needs its own learning-rate grid search. MiniCPM (2024)—a historical dense-model case study targeting 1.2B and 2.4B models—published one of the cleanest end-to-end recipes for eliminating per-scale learning-rate sweeps by combining three ingredients: Maximal Update Parametrization (μP), WSD schedules, and Chinchilla scaling fits.

MiniCPM first constructed a seven-model sweep family spanning from 9M to 0.5B parameters while holding the head dimension fixed at $d_h = 64$ and keeping the aspect ratio consistent as model width d_m, feedforward dimension d_{ff}, head count $n_h = d_m/d_h$, and layer count L grew together:

Model Name	N (B)	d_m	d_{ff}	d_h	n_h	L
9M	0.009	320	800	64	5	8
30M	0.036	512	1280	64	8	12
70M	0.066	640	1600	64	10	14
0.1B	0.109	768	1920	64	12	16
0.17B	0.166	896	2240	64	14	18
0.2B	0.241	1024	2560	64	16	20
0.5B	0.499	1344	3360	64	21	24

Notice two disciplined design choices in this table. First, $d_{ff}/d_m = 2.5$ across every single row, and d_m/L stays tightly bounded as depth grows from 8 to 24 layers, ensuring the sweep measures clean scale progression rather than architectural drift. Second, the extrapolation gap from the largest sweep model (0.5B) to the final target model (2.4B) is only about $5\times$ in parameters, keeping extrapolation risk modest.

To avoid re-tuning the learning rate for each of those seven widths and for the final 2.4B target, MiniCPM applied five explicit μP scaling operations while holding four base hyperparameters fixed across all widths: `scale_emb = 12`, `scale_depth = 1.4`, `init_std = 0.1`, and base `lr = 0.01`. Letting d_{base} denote the reference proxy width, MiniCPM's exact rules are:

Operation	Exact Rule in MiniCPM
Embedding Output Scaling	Multiply embedding output by scale_emb
Residual Scaling	Scale each layer's residual increment by $\text{scale_depth}/\sqrt{L}$
Tensor Initialization	Set 2D tensor init standard deviation to $\text{init_std}/\sqrt{d_m/d_{\text{base}}}$; other parameters to 0.1
LR Scaling of Tensors	Scale 2D tensor LR by $1/(d_m/d_{\text{base}})$ relative to base LR
LM Head Scaling	Scale output logits by $1/(d_m/d_{\text{base}})$

We will derive the exact spectral theory behind why initialization scales as $1/\sqrt{d_m/d_{\text{base}}}$, AdamW learning rate scales as $1/(d_m/d_{\text{base}})$, and output logits scale as $1/(d_m/d_{\text{base}})$ when we get to Section 8. For now, observe the empirical payoff: across 0.04B, 0.1B, 0.3B, 0.5B, and 2.1B models, plotting validation loss versus base learning rate produces curves whose minima all align tightly around $\eta \approx 10^{-2}$. Minimal learning-rate drift eliminates per-scale learning-rate grid searches.

Even with learning rate locked by μP, two scaling quantities still must be determined empirically: the **optimal batch size** and the **optimal token-to-parameter ratio** $D_{\text{opt}}/N_{\text{opt}}$.

First, for batch size, MiniCPM swept pairs (B, D) across the 9M, 30M, and 170M models, traced the minimum-loss ridge at each token budget, and fit a Kaplan-style power law relating optimal batch size BS (in tokens) to the cross-entropy loss L:

$$\log(\text{BS}) = -6.24\log(L) + 20.91 \iff \text{BS} \approx \frac{1.21 \times 10^9}{L^{6.24}}$$

Let us unpack this equation term by term. On the left-hand side, $\log(\text{BS})$ is the natural logarithm of the optimal batch size in tokens; on the right-hand side, $\log(L)$ is the natural logarithm of the validation loss. Exponentiating both sides gives $e^{20.91} \approx 1.21 \times 10^9$ in the numerator and $L^{6.24}$ in the denominator. Conceptually, this equation expresses how gradient noise evolves as training progresses: early in training when L is high, individual token gradients carry a strong shared signal, so small batches suffice; as L falls, gradients become noisier relative to the remaining reducible error, so the optimal batch size grows rapidly—polynomially with exponent -6.24—as loss drops.

Second, to determine how to allocate compute $C = 6ND$ between parameters N and tokens D, MiniCPM applied Chinchilla Methods 1 (lower envelope) and 3 (joint parametric fit) using **byte-normalized loss** across four corpora (Code, English, Chinese, and Ultratext). Recall the Chinchilla parametric loss function:

$$L(N, D) = C_N N^{-\alpha} + C_D D^{-\beta} + L_0, \qquad \text{subject to } C = 6ND$$

Here L_0 is the irreducible entropy floor of the data distribution (in byte-normalized units), $C_N N^{-\alpha}$ is the finite-capacity error from having N parameters, and $C_D D^{-\beta}$ is the finite-data optimization error from training on D tokens. Substituting $D = \frac{C}{6N}$ into $L(N, D)$ turns the constrained optimization into

a single-variable minimization over N:

$$L(N) = C_N N^{-\alpha} + C_D \left(\frac{C}{6}\right)^{-\beta} N^{\beta} + L_0$$

Setting $\frac{dL}{dN} = 0$ gives:

$$-\alpha C_N N^{-\alpha-1} + \beta C_D \left(\frac{C}{6}\right)^{-\beta} N^{\beta-1} = 0 \implies \alpha C_N N^{-\alpha} = \beta C_D D^{-\beta}$$

Solving this optimality condition alongside $ND = C/6$ yields the optimal parameter-to-token ratio in the exact form reported by MiniCPM:

$$\frac{N_{\text{opt}}}{D_{\text{opt}}} = K^2 \left(\frac{C}{6}\right)^{\eta}$$

where $\eta = \frac{\beta-\alpha}{\alpha+\beta}$ governs whether the optimal ratio tilts toward parameters ($\eta > 0$) or tokens ($\eta < 0$) as compute grows, and $K^2 = \left(\frac{\alpha C_N}{\beta C_D}\right)^{\frac{2}{\alpha+\beta}}$ sets the baseline scale of the ratio when $\alpha \approx \beta$. When $\alpha = \beta$, $\eta = 0$, meaning the optimal data-to-parameter ratio $D_{\text{opt}}/N_{\text{opt}} = 1/K^2$ stays constant across compute budgets.

On the Ultratext corpus, MiniCPM's joint fit gave:

$$L = 0.0754\, N^{-0.30} + 0.292\, D^{-0.30} + 0.25$$

with $K^2 = 0.01$ and $\eta \approx 0$, yielding an optimal data-to-parameter ratio of:

$$\left.\frac{D_{\text{opt}}}{N_{\text{opt}}}\right|_{C=10^{21}} = 95.6$$

WORKED EXAMPLE

Worked Example: Checking MiniCPM's Batch-Size Law and Token-to-Parameter Ratio.

1. **Optimal batch size as loss drops:** Suppose a model's cross-entropy loss drops from $L = 2.5$ early in training to $L = 1.8$ later in training. Plugging both values into $\text{BS} \approx 1.21 \times 10^9 / L^{6.24}$:
   - At $L = 2.5$: $2.5^{6.24} \approx 303.7$, so $\text{BS} \approx \frac{1.21\times10^9}{303.7} \approx 3.98 \times 10^6$ tokens (≈ 4.0M tokens).
   - At $L = 1.8$: $1.8^{6.24} \approx 39.17$, so $\text{BS} \approx \frac{1.21\times10^9}{39.17} \approx 30.89 \times 10^6$ tokens (≈ 30.9M tokens). A 28% reduction in loss causes the predicted optimal batch size to jump by nearly $8\times$.
2. **Compute-optimal (N, D) from the Ultratext fit:** Starting from $\alpha C_N N^{-\alpha} = \beta C_D D^{-\beta}$ with the rounded Ultratext fit $C_N = 0.0754$, $C_D = 0.292$, and $\alpha = \beta = 0.30$, the exponents

cancel in the prefactors:

$$0.0754\,N^{-0.30} = 0.292\,D^{-0.30} \implies \left(\frac{D_{\text{opt}}}{N_{\text{opt}}}\right)^{0.30} = \frac{0.292}{0.0754} \approx 3.873$$

Raising both sides to the $1/0.30 \approx 3.333$ power gives $D_{\text{opt}}/N_{\text{opt}} \approx 91.2$ (while the unrounded Ultratext fit with $K^2 = 0.01$ and slight exponent difference yields 95.6 at $C = 10^{21}$ FLOPs). Using $D_{\text{opt}}/N_{\text{opt}} = 95.6$ at $C = 6ND = 10^{21}$ FLOPs:

$$6N_{\text{opt}}(95.6\,N_{\text{opt}}) = 10^{21} \implies N_{\text{opt}} = \sqrt{\frac{10^{21}}{573.6}} \approx 1.32 \times 10^{9} \text{ params}, \quad D_{\text{opt}} \approx 126\text{I}$$

Notice that the byte-normalized loss $L_0 = 0.25$ in the Ultratext fit uses different units from the token-level cross-entropy L in the batch-size formula above—do not plug byte-normalized loss directly into the token-level batch-size equation.

Both the lower-envelope fit (Method 1) and the joint fit (Method 3) in MiniCPM yielded compute-optimal data-to-parameter ratios around $95.6 : 1$—nearly five times higher than Chinchilla's famous $\sim 20 : 1$ ratio.

COMMON MISCONCEPTION

You might think that Chinchilla's $20 : 1$ token-to-parameter ratio is a universal physical constant of Transformers. That breaks because $D_{\text{opt}}/N_{\text{opt}} = (\beta C_D/\alpha C_N)^{1/\alpha}$ (when $\alpha = \beta$) depends sensitively on the dataset quality, tokenizer/byte-normalization, architecture, and optimization schedule—all of which shift the constants C_N and C_D and can easily move the compute-optimal ratio from $20 : 1$ to $40 : 1$ or nearly $100 : 1$.

KEY IDEA

- MiniCPM eliminates per-scale learning-rate sweeps by pairing WSD with five μP scaling rules, keeping the optimal base learning rate stable near $\eta \approx 10^{-2}$ from 0.04B to 2.1B parameters.
- Optimal batch size grows steeply as loss falls ($\text{BS} \propto L^{-6.24}$), and empirical Chinchilla fits on modern data and schedules yield far higher token-to-parameter ratios (such as $95.6 : 1$) than the original $20 : 1$ baseline.

MiniCPM shows how cleanly scaling works when you commit to μP from the start. Yet many major production labs—including DeepSeek LLM and Llama 3 in 2024—trained dense frontier-scale models *without* μP. How do you choose learning rate, batch size, model size, and even predict downstream benchmark accuracy when you stay in standard parametrization? That is the exact problem addressed by direct empirical fitting.

✎ Check Your Understanding

1. **Fill-in-the-blank:** In MiniCPM's μP implementation, when model width d_m doubles relative to d_{base}, the initialization standard deviation of 2D weight tensors is scaled by _____, while their learning rate is scaled by _____.
   Answer: $1/\sqrt{2}$ (from $1/\sqrt{d_m/d_{\text{base}}}$) for the initialization standard deviation, and $1/2$ (from $1/(d_m/d_{\text{base}})$) for the 2D tensor learning rate.
2. **What would happen if...** Suppose in $L(N, D) = C_N N^{-\alpha} + C_D D^{-\beta} + L_0$, the exponents satisfy $\alpha = \beta = 0.30$ and a cleaner dataset cuts C_N in half while leaving C_D unchanged. Does the compute-optimal token-to-parameter ratio $D_{\text{opt}}/N_{\text{opt}}$ increase or decrease?
   Answer: It increases. Since $(D_{\text{opt}}/N_{\text{opt}})^{\alpha} = C_D/C_N$, halving C_N doubles the right-hand side and multiplies $D_{\text{opt}}/N_{\text{opt}}$ by $2^{1/0.30} \approx 10.1\times$.

Section 3: Direct Empirical Fitting and Downstream Prediction — DeepSeek LLM and Llama 3

Not every team wants to rewrite initialization, residual scaling, attention scaling, and per-tensor learning rates to implement μP—especially when inheriting an existing standard-parametrization codebase. If you do not use μP, the optimal learning rate and batch size both drift as model size and compute scale up. Two landmark 2024 dense-model recipes—DeepSeek LLM (7B MHA and 67B GQA on 2T tokens) and Llama 3 (July 2024, 405B on 15T tokens)—demonstrated how to scale without μP by fitting direct empirical scaling laws for hyperparameters, compute allocation, and downstream task performance.

DeepSeek LLM tackled the hyperparameter problem by treating optimal learning rate η_{opt} and optimal batch size B_{opt} as power-law functions of total training compute C. The team ran grid searches over (B, η) at small compute budgets—specifically 10^{17} FLOPs (using a model of 177M FLOPs/token) and 10^{20} FLOPs (using a model of 2.94B FLOPs/token). A critical empirical observation made direct fitting viable: generalization error stays remarkably flat over a **wide basin** of (B, η) combinations, with many pairs landing within 0.25% of the minimum validation loss. Fitting power laws across all near-optimal points yielded:

$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.1250}, \qquad B_{\text{opt}} = 0.2920 \cdot C^{0.3271}$$

Break both equations apart. In each expression, C is the total non-embedding training compute in FLOPs. In the learning-rate law, the negative exponent $-0.1250 = -1/8$ tells you that every $10\times$ increase in compute shrinks the optimal learning rate by a factor of $10^{-0.125} \approx 0.75$—a gentle decay that reflects larger models needing smaller steps in standard parametrization. In the batch-size law, the positive

exponent $+0.3271 \approx 1/3$ tells you that every $10\times$ increase in compute expands the optimal batch size (in tokens) by $10^{0.3271} \approx 2.12\times$. Together, these two equations turn an entire 2D grid search at target scale into a one-line evaluation of C.

WORKED EXAMPLE

Worked Example: Predicting DeepSeek LLM 7B and 67B Hyperparameters.

Let us evaluate DeepSeek LLM's power laws for its two target runs on $D = 2 \times 10^{12}$ tokens:

1. **7B MHA (2T tokens):** Using the standard dense compute approximation $C \approx 6ND = 6 \times (7 \times 10^9) \times (2 \times 10^{12}) = 8.40 \times 10^{22}$ FLOPs:

$$\eta_{\text{opt}} = 0.3118 \cdot (8.40 \times 10^{22})^{-0.1250} \approx 4.25 \times 10^{-4}$$

$$B_{\text{opt}} = 0.2920 \cdot (8.40 \times 10^{22})^{0.3271} \approx 9.20 \times 10^6 \text{ tokens}$$

   This matches DeepSeek LLM's reported predictions ($\eta = 4.2 \times 10^{-4}$, $B = 9.2\text{M}$ tokens).

2. **67B GQA (2T tokens):** DeepSeek LLM measures model scale by non-embedding FLOPs per token M (so $C = MD$). For the 67B model, $M = 4.3 \times 10^{11}$ FLOPs/token, giving $C = (4.3 \times 10^{11}) \times (2 \times 10^{12}) = 8.60 \times 10^{23}$ FLOPs:

$$\eta_{\text{opt}} = 0.3118 \cdot (8.60 \times 10^{23})^{-0.1250} \approx 3.17 \times 10^{-4}$$

$$B_{\text{opt}} = 0.2920 \cdot (8.60 \times 10^{23})^{0.3271} \approx 19.70 \times 10^6 \text{ tokens}$$

   Again, this matches the reported table values ($\eta = 3.2 \times 10^{-4}$, $B = 19.7\text{M}$ tokens).

3. **Interpretation:** Moving from 7B to 67B ($\sim 10\times$ more compute) drops the predicted learning rate by roughly 25% and more than doubles the optimal batch size.

Despite how cleanly those two numbers turned out, there is an important statistical caveat. While DeepSeek LLM's batch-size power law is clean across 8 orders of magnitude in FLOPs, the **learning-rate fit has wide vertical scatter** across the small runs. Extrapolating $\eta_{\text{opt}}(C)$ across three to four orders of magnitude succeeds largely because the loss basin around the optimum is wide and flat—not because the small-scale LR power law is tightly constrained. We will return to this fragility in Section 5 when we compare seven competing hyperparameter laws.

Once $\eta_{\text{opt}}(C)$ and $B_{\text{opt}}(C)$ are set, how did DeepSeek LLM size its models and predict final loss? The team used **Chinchilla Method 2 (IsoFLOPs)** across compute budgets from 10^{17} to 3×10^{20} FLOPs, making two metric refinements over naive parameter counting:

DEFINITION

Non-embedding FLOPs per token (M). The exact floating-point operations per forward-and-backward pass on a single token excluding vocabulary embedding lookups, so total training compute satisfies $C = MD$ cleanly across different attention and MLP shapes.

DEFINITION

Bits-per-byte (BPB). A tokenizer-independent validation loss metric measuring the average number of bits required to encode each raw UTF-8 byte of text, allowing fair loss comparisons across different vocabularies and tokenizers.

Picture the two side-by-side plots of DeepSeek LLM's IsoFLOP study. In panel (a), *IsoFLOP Curves (*10^{17}–3×10^{20}*)*, the vertical axis is validation bits-per-byte and the horizontal axis is $\log M$ (non-embedding FLOPs/token). Five U-shaped parabolas step downward and to the right as the fixed compute budget C increases. Each parabola has a clear minimum—representing the optimal model scale $M_{\text{opt}}(C)$ for that compute budget—and a dashed straight line passes cleanly through all five minima. In panel (b), *Loss Extrapolation*, the vertical axis is validation bits-per-byte and the horizontal axis is $\log C = \log(MD)$. Five small-run optima at $C \leq 3 \times 10^{20}$ FLOPs define a smooth downward power-law curve; extrapolating that curve over three full orders of magnitude to $C = 4.5 \times 10^{23}$ FLOPs predicts $M = 4.3 \times 10^{11}$ FLOPs/token and $D = C/M = 1.04 \times 10^{12}$ tokens (the 67B scale). When DeepSeek LLM actually trained the 7B and 67B models, their final validation losses landed directly on the extrapolated curve fit from runs at most 3×10^{20} FLOPs.

Pretraining validation loss, however, is not the quantity users actually care about—downstream benchmark accuracy is. Llama 3 (July 2024, historical dense baseline of 405B parameters on 15T tokens) extended the IsoFLOP workflow into a **three-stage compute-to-downstream prediction pipeline**, illustrated as three sequential stages feeding left to right:

1. **Stage 1 — IsoFLOPs (6×10^{18} to 10^{22} FLOPs):** Just like DeepSeek LLM, Llama 3 fit quadratic IsoFLOP parabolas over compute budgets spanning 6×10^{18} to 10^{22} FLOPs. Tracing the parabola minima produced a compute-optimal token-to-parameter ratio of approximately $39 : 1$ ($D : N$)—nearly double Chinchilla's $20 : 1$ and well below MiniCPM's $95.6 : 1$.
2. **Stage 2 — Compute $\rightarrow$ Task Negative Log-Likelihood (NLL):** Rather than stopping at general pretraining loss, Stage 2 plots the negative log-likelihood per character (NLL/char) of the correct answer on a specific downstream benchmark against $\log C$. Across the small compute-optimal scaling-law models, this relationship is a clean downward-sloping straight line, allowing a linear extrapolation in $\log C$ out to the target 405B compute budget.
3. **Stage 3 — Task NLL $\rightarrow$ Benchmark Accuracy:** Accuracy (such as multiple-choice or coding pass rate) is bounded between random-chance floor and 100%, so it cannot be a linear function of NLL. Stage 3 plots benchmark accuracy on the vertical axis against task NLL on the horizontal axis and fits a **sigmoid curve**. Because small scaling-law models alone often sit near the low-accuracy flat tail of the sigmoid, Llama 3 combined those small models with older **Llama 2 (2023) checkpoints** to anchor the middle and upper rise of the sigmoid. Plugging Stage 2's extrapolated NLL for 405B into Stage 3's sigmoid accurately predicted Llama 3 405B's downstream benchmark accuracy *before* committing to the full 405B / 15T-token run.

COMMON MISCONCEPTION

You might think that you can extrapolate downstream benchmark accuracy directly as a power law of training FLOPs C. That breaks because accuracy is a nonlinear, thresholded metric—models often sit flat near random guessing for two orders of magnitude in compute before rising sharply ("emergent" jumps) and then saturating near 100%. Decomposing the prediction into a smooth power law for $\text{Compute} \to \text{Task NLL}$ followed by a sigmoidal map $\text{Task NLL} \to \text{Accuracy}$ removes the apparent discontinuity.

KEY IDEA

- Direct empirical fitting without μP relies on IsoFLOPs (Method 2) over non-embedding FLOPs/token M and bits-per-byte, paired with power-law fits $\eta_{\text{opt}}(C)$ and $B_{\text{opt}}(C)$ whose extrapolation works largely because of a wide flat loss basin.
- Llama 3 bridges pretraining scaling laws to downstream capabilities through a two-step link: a linear fit in $\log C$ predicting task NLL per character, composed with a sigmoid (anchored by both scaling-law runs and prior-generation checkpoints) predicting benchmark accuracy.

All three case studies so far—MiniCPM, DeepSeek LLM, and Llama 3—were **dense** Transformers, where every token activates every parameter ($C \approx 6ND$). By 2025 and into late 2026, virtually every frontier model switched to **sparse Mixture-of-Experts (MoE)** paired with sparse or hybrid attention architectures. How do scaling laws change when total parameters decouple from active parameters per token, and when attention itself becomes a variable to sweep?

✎ Check Your Understanding

1. **Yes/No with justification:** In DeepSeek LLM's hyperparameter scaling study, was the small-scale learning-rate power law $\eta_{\text{opt}}(C)$ as tightly concentrated around its trend line as the batch-size power law $B_{\text{opt}}(C)$?
   Answer: No. While the batch-size law was clean across 8 orders of magnitude, the learning-rate fit had wide vertical scatter on small runs; extrapolation succeeded mainly because generalization error stays flat within 0.25% across a wide basin of (B, η) pairs.
2. **Fill-in-the-blank:** In Llama 3's three-stage downstream prediction pipeline, Stage 2 maps $\log C$ linearly to _____, and Stage 3 maps that quantity to benchmark accuracy using a _____ function anchored with small scaling runs and Llama 2 checkpoints.
   Answer: task negative log-likelihood per character (NLL/char); sigmoid.

Section 4: Extending Scaling Laws to Sparse MoEs, Architectures, and Synthesizing Recipes

In a dense Transformer, parameter count N controls both model capacity (memory) and per-token compute ($M \approx 6N$). In a sparse Mixture-of-Experts (MoE) model, each token routes through only a small subset of experts, splitting N into **total parameters** N_{total} and **activated parameters** N_a. Because training FLOPs scale with $N_a D$ rather than $N_{\text{total}} D$, dense scaling ratios like $20:1$ or $39:1$ no longer describe the compute-optimal trade-off. Furthermore, modern frontier models also vary the attention mechanism itself (comparing full softmax attention against linear, sparse, and hybrid variants). Scaling studies from 2024–2025 extended the empirical toolkit along three new axes: active-parameter IsoFLOPs, expert sparsity laws, and architecture selection sweeps.

Active-Parameter IsoFLOPs: Hunyuan-Large (2024)

Hunyuan-Large (2024) adapted Chinchilla Method 2 (IsoFLOPs) to sparse MoEs by sweeping compute budgets from 5×10^{18} to 9.5×10^{19} FLOPs and plotting validation loss against **activated parameters** N_a. Tracing the minima of the MoE IsoFLOP parabolas yielded a compute-optimal ratio of $96:1$ **(tokens to active parameters)**—far higher than dense Chinchilla's $20:1$. Extrapolating this law to the target compute budget predicted an optimal active size of 58.1B parameters (and the final model was trained with 52B activated parameters).

Why does an MoE prefer a much higher token-to-active-parameter ratio than a dense model? Because each active FLOP is backed by a much larger pool of total expert parameters; to fill that much larger total capacity with enough training signal per expert, compute-optimal allocation must send far more tokens per active parameter than a dense model requires.

Sparsity Scaling Laws: Kimi K2 (July 2025)

Holding activated parameters fixed, how many total experts should an MoE have? Kimi K2 (July 2025, an earlier milestone on the path to late-2026 frontier MoEs) formalized a **sparsity scaling law** by fixing the active routing budget at 8 active routed experts plus 1 shared expert and sweeping the total expert count.

> **DEFINITION**
>
> **MoE sparsity.** The ratio of total routed experts to active routed experts per token:
>
> $$\text{sparsity} = \frac{\text{total experts}}{\text{active experts}}.$$

Kimi K2 found that at constant training FLOPs, **higher sparsity consistently lowers validation loss**. Quantitatively, to reach a target validation loss of 1.5, using **sparsity** 48 reduced required training FLOPs by:

- $1.69\times$ compared to sparsity 8,
- $1.39\times$ compared to sparsity 16, and

- $1.15\times$ compared to sparsity 32.

Notice the diminishing returns alongside rising system overhead: going from sparsity $8 \to 16 \to 32 \to 48$ yields progressively smaller FLOP savings while increasing total parameter memory and all-to-all routing communication during training and serving. Kimi K2 therefore chose **sparsity** 48 (8 active out of 384 routed experts) as the practical trade-off between validation loss and serving/training overhead. In the same sweep, Kimi K2 also measured attention scaling, finding that doubling the number of attention heads cut validation loss by 0.5% **to** 1.2%.

WORKED EXAMPLE

Worked Example: Computing Sparsity and FLOP Savings in Kimi K2 and Kimi K3.

1. **FLOP reduction at validation loss 1.5 (Kimi K2):** Suppose a sparsity-48 MoE reaches validation loss 1.5 in $C_{48} = 1.00 \times 10^{22}$ FLOPs. Using Kimi K2's measured FLOP ratios:
   - Sparsity 32 requires $1.15 \times C_{48} = 1.15 \times 10^{22}$ FLOPs.
   - Sparsity 16 requires $1.39 \times C_{48} = 1.39 \times 10^{22}$ FLOPs.
   - Sparsity 8 requires $1.69 \times C_{48} = 1.69 \times 10^{22}$ FLOPs. Switching from sparsity 8 (64 experts, 8 active) to sparsity 48 (384 experts, 8 active) saves 0.69×10^{22} FLOPs—a 40.8% reduction in total training compute to hit the exact same loss.
2. **Connecting to late-2026 frontier sparsity:** Compare Kimi K2 (July 2025) to the current late-2026 frontier model Kimi K3 / K3.1 (July 2026), which has 896 routed experts with 16 active (plus 2 shared). Its sparsity is:

$$\text{sparsity}_{\text{Kimi K3}} = \frac{896}{16} = 56.$$

   Similarly, DeepSeek-V4.1-Flash (Sept 2026) uses 384 routed experts with 6 active ($384/6 = 64$), and Qwen3.8-Flash-Next (Aug 2026) uses 512 routed experts with 10 active ($512/10 = 51.2$). Late-2026 frontier models pushed sparsity slightly past Kimi K2's 48 into the 51–64 range as routing infrastructure matured.

Multi-Stage Hyperparameter Laws & Architecture Selection: Qwen 2.5/3 and MiniMax-01

Two additional 2024–2025 studies broadened scaling laws into full system design tools:

- **Qwen 2.5 (2024) and Qwen 3 (May 2025):** Rather than fitting a single scalar compute law, the Qwen team fit direct empirical scaling laws for η_{opt} and B_{opt} jointly as functions of N, D, architecture (dense vs. MoE), and pretraining stage. Qwen 2.5 swept dense models (44M–14B) and MoE models (44M–1B active) across 0.8B–600B tokens, explicitly using the fitted laws to configure MoE models to match the quality of 14B and 72B dense baselines. Qwen 3 extended this exact hyperparameter-scaling framework across three pretraining stages totaling 36T tokens.
- **MiniMax-01 (2025) — Architecture Scaling:** MiniMax-01 applied **Chinchilla Method 1** (lower envelope of loss vs. compute) across 70M to 7B models to compare three candidate attention

architectures *before* committing to a full-scale run: **Softmax Attention**, **Lightning Attention**, and **Hybrid-Lightning Attention**. By deriving the compute-optimal model size and token count *per architecture*, MiniMax-01 established that scaling laws are now standard for **architecture selection**, not just (N, D) sizing. Current late-2026 frontier models—such as **MiniMax-M3** (June 2026, blockwise top-k sparse attention cutting per-token attention compute by $28.4\times$ at 1M context on a 109B model), **Qwen3.8-Flash-Next** (Aug 2026, 3:1 Gated DeltaNet and Qwen Sparse Attention with 512 experts), and **DeepSeek-V4** (2026, hybrid CSA and HCA sparse attention)—all pair sparse or hybrid attention with sparse MoE selected through such sweeps.

Synthesizing the Two Dominant Scaling Recipes

Stepping back across Sections 1–4, real-world pretraining recipes consolidate into two dominant schools of practice, summarized across five design axes:

Axis	Direct Empirical Fitting (DeepSeek LLM, Qwen 2.5/3, Llama 3, Hunyuan)	Parametrization-Invariant (μP) (MiniCPM, Cerebras-GPT, 2026 MoE μP)
Architectural hyperparameters	Assume most Transformer hyperparameters are scale-invariant	Rescale initialization, LR, and multipliers by width so dynamics stay invariant
LR & Batch	Grid search at small scale; fit power laws η_{opt}, B_{opt}	Tune base LR once on a tiny proxy; still fit batch size vs. L or D
(N, D) **sizing**	IsoFLOPs (Method 2) on dense or active MoE parameters	Joint fit (Method 3) or lower envelope (Method 1)
Schedule	Multi-step / WSD to reuse stable trunk	WSD ($\sim 10\%$ decay) for linear-cost data sweeps
Extra axes	MoE sparsity laws (Kimi K2, July 2025); attention architecture sweeps (MiniMax-01, 2025)	MoE sparsity laws (Kimi K2, July 2025); attention architecture sweeps (MiniMax-01, 2025)

COMMON MISCONCEPTION

You might think that using μP eliminates the need for any empirical scaling fits at all. That breaks because classical μP stabilizes learning-rate transfer across width, but you must still empirically determine the optimal batch size (as a function of loss L or tokens D), the compute-optimal (N, D) split, MoE sparsity, and the best attention architecture.

KEY IDEA

- Sparse MoE scaling shifts compute-optimal allocation to much higher token-to-active-parameter ratios (such as Hunyuan-Large's $96:1$) and introduces sparsity ($\text{total}/\text{active}$ experts) and attention architecture as first-class scaling axes.
- Practical scaling recipes divide into two camps: direct empirical fitting of hyperparameters and IsoFLOPs versus μP-based invariant scaling, both relying on WSD or multi-step schedules to control sweep cost.

If you follow the direct empirical fitting route—or fit batch sizes even under μP—a natural question arises immediately: what is the *right* functional form for η_{opt} and B_{opt}? Should they depend on N, D, C, or L? And what happens when a clean-looking empirical law fails during a $100\times$ extrapolation? We examine competing hyperparameter laws and their failure modes next.

✎ Check Your Understanding

1. **Yes/No with justification:** Holding active parameters fixed at constant training FLOPs, does increasing MoE sparsity from 16 to 48 improve validation loss, and should you keep increasing sparsity arbitrarily?
   Answer: Yes, increasing sparsity from 16 to 48 consistently lowers validation loss (cutting FLOPs by $1.39\times$ to reach loss 1.5 in Kimi K2), but you cannot increase it arbitrarily because FLOP savings show diminishing returns while memory and routing/serving overheads grow.
2. **Quick computation:** An MoE layer has 512 total routed experts and routes each token to 8 active routed experts (plus 1 shared expert). What is its sparsity according to Kimi K2's definition?
   Answer: $\text{sparsity} = \frac{\text{total experts}}{\text{active experts}} = \frac{512}{8} = 64.$

Section 5: Competing Hyperparameter Scaling Laws, Step Law, and Extrapolation Failure

In Sections 2 and 3, we saw two completely different formulas for the same quantities: MiniCPM predicted batch size as a function of loss L ($\text{BS} \propto L^{-6.24}$), while DeepSeek LLM predicted both batch size and learning rate as functions of total compute C ($\eta_{\text{opt}} \propto C^{-0.125}$, $B_{\text{opt}} \propto C^{0.3271}$). Those are not the only proposals in the literature. Across major pretraining papers, at least **seven competing hyperparameter laws** disagree on the fundamental predictors—whether optimal learning rate and batch size are governed by parameter count N, token count D, compute $C = 6ND$, or validation loss L:

Law	Data Recipe Tested	Sparsity Tested	Learning Rate (η_{opt})	Batch Size (B_{opt})	Rel. Loss Err.
Kaplan / OpenAI (2020)	✗	✗	$3.239 \times 10^{-3} - 1.395 \times 10^{-4} \log N$	$2 \times 10^{8} L^{-4.76}$	9.51‰
Microsoft Law	✗	✗	$1.3192 \times 10^{-5} N^{-0.23} D^{-0.32}$	—	9.25‰
DeepSeek LLM (2024)	✗	✗	$0.3118\, C^{-0.1250}$	$0.2920\, C^{0.3271}$	9.26‰
Porian Law	✗	✗	$3.7\, N^{-0.36}$	$0.7576\, N^{0.703}$	3.71‰
MiniCPM (2024)	✗	✗	— (fixed via μP)	$1.21 \times 10^{9} / L^{6.24}$	—
MeiTuan Law	✗	✓	$\lambda L^{-\alpha}$	$\lambda_B L^{-\alpha_B^{-1}}$	—
Step Law (StepFun)	✓	✓	$1.79\, N^{-0.713} D^{0.307}$	$0.58\, D^{0.571}$	0.94‰

(Note: All coefficients above come from the primary papers—Kaplan et al. 2020, DeepSeek LLM 2024, MiniCPM 2024—correcting transcription typos present in the Step Law comparison table.)

Look closely at the structural contradictions in this table. Kaplan's learning-rate law is logarithmic in N (and goes negative at huge N if extrapolated blindly!). Porian's laws depend purely on N—ignoring how many tokens D you train on—which immediately fails when you overtrain a small model on trillions of tokens. DeepSeek LLM collapses N and D into their product $C = 6ND$, forcing η_{opt} and B_{opt} to depend symmetrically on N and D even when D/N varies by $10\times$. Microsoft's law separates N and D with *negative* exponents on both ($N^{-0.23} D^{-0.32}$), whereas **Step Law** separates N and D with opposite signs on η_{opt} and drops N completely from B_{opt}, cutting relative validation-loss error down to $0.94‰$ (less than one part in a thousand)—roughly $4\times$ to $10\times$ lower than Porian ($3.71‰$) and DeepSeek/Microsoft/Kaplan ($\sim 9.3‰$–$9.5‰$).

Let us unpack Step Law's two equations carefully:

$$\eta_{\text{opt}} = 1.79\, N^{-0.713}\, D^{0.307}, \qquad B_{\text{opt}} = 0.58\, D^{0.571}$$

Break these apart term by term:

- In $B_{\text{opt}} = 0.58\, D^{0.571}$, D is total training tokens and B_{opt} is optimal batch size in tokens. Notice that N does not appear at all: across model sizes spanning $N \in [59\text{M}, 1\text{B}]$, StepFun found that optimal batch size collapses onto a single power-law line in D alone. Because the exponent 0.571 is slightly above $1/2$, doubling the token budget increases the optimal batch size by $2^{0.571} \approx 1.49\times$.
- In $\eta_{\text{opt}} = 1.79\, N^{-0.713}\, D^{0.307}$, the two factors act in opposite directions. Scaling up model size N at fixed data D sharply *decreases* the optimal learning rate ($N^{-0.713}$). Scaling up training tokens D at fixed model size N *increases* the optimal learning rate ($D^{0.307}$) under cosine decay—partly because

larger D also increases B_{opt} via $D^{0.571}$, and larger batches tolerate larger learning rates. Together, the two formulas map any dense or MoE training specification (N, D) to a single predicted point $(\eta_{\text{opt}}, B_{\text{opt}})$ inside the loss landscape.

Visualize the pretraining loss contour plot over $(\log \eta, \log B)$ from the Step Law study. On a plane with $\log(\text{Learning Rate})$ on the horizontal axis and $\log(\text{Batch Size})$ on the vertical axis, the loss contours form nested, tilted ellipses—smooth and **convex** around a single global minimum marked by a red $\times$. The inner ring encloses points within $+0.25\%$ of minimum loss; the outer ring encloses points within $+1.0\%$. The tilt of the ellipses reflects the positive coupling between learning rate and batch size (larger batches shift the optimal LR slightly higher). Right next to the red $\times$ sits Step Law's prediction (gold $\star$), well inside the inner $+0.25\%$ ring. By contrast, DeepSeek LLM's prediction (teal ▲) sits up and to the left near the $+0.25\%$–$+1.0\%$ boundary, and Porian's prediction (purple ■) sits up and to the right—both offset from the true global minimum because they cannot separate N and D independently.

WORKED EXAMPLE

Worked Example: Comparing Step Law vs. DeepSeek LLM Predictions.

Consider a target dense run with $N = 7 \times 10^9$ parameters trained on $D = 2 \times 10^{12}$ tokens, so $C = 6ND = 8.40 \times 10^{22}$ FLOPs (a heavily overtrained regime with $D/N \approx 286$).

1. **Step Law prediction:**
   - First compute optimal batch size from D alone:

$$B_{\text{opt}} = 0.58 \cdot (2 \times 10^{12})^{0.571} = 0.58 \cdot (1.057 \times 10^7) \approx 6.13 \times 10^6 \text{ tokens.}$$

   - Next compute optimal learning rate from N and D:

$$(7 \times 10^9)^{-0.713} \approx 9.56 \times 10^{-8}, \qquad (2 \times 10^{12})^{0.307} \approx 5.976 \times 10^3$$

$$\eta_{\text{opt}} = 1.79 \cdot (9.56 \times 10^{-8}) \cdot (5.976 \times 10^3) \approx 1.02 \times 10^{-3}.$$

2. **DeepSeek LLM prediction (from Section 3):**
   - At $C = 8.40 \times 10^{22}$ FLOPs, DeepSeek predicts $B_{\text{opt}} \approx 9.20 \times 10^6$ tokens and $\eta_{\text{opt}} \approx 4.25 \times 10^{-4}$.
3. **Interpretation:** Because $D/N = 286$ is far above Chinchilla-optimal $20:1$, DeepSeek's compute-only formula conflates the huge token count D with a larger model size, pushing its predicted learning rate downward (4.25×10^{-4}) and its batch size upward (9.20M). Step Law separates the two effects: the moderate $N = 7\text{B}$ combined with the large $D = 2\text{T}$ keeps the predicted learning rate more than $2\times$ higher (1.02×10^{-3}) and the batch size tighter (6.13M).

Step Law rests on three empirical observations—the third of which carries a critical warning:

1. **Convexity:** Pretraining loss slices in $\log \eta$ and $\log B$ are smooth and convex, making global optima cleanly identifiable without noisy local traps.
2. **Batch size depends on D:** Across $N \in [59\text{M}, 1\text{B}]$, $B_{\text{opt}} = 0.58\, D^{0.571}$ holds independent of N.
3. **LR vs. D schedule caveat:** Step Law's positive data exponent ($D^{0.307}$) was fit under **cosine decay**. An InternLM scaling study (Zhou et al., 2026) demonstrated that this D-exponent is **fragile under WSD**—swapping cosine for WSD shifts how η_{opt} scales with D. Because WSD holds peak LR constant for 90% of training rather than immediately annealing it, raising η with D interacts differently with long stable phases.

Within its cosine regime, Step Law showed impressive generalization without re-fitting across two axes that earlier laws never tested:

- **Transfer across MoE sparsity:** Evaluated on $\sim 2.1\text{B}$-total MoE models ($D = 8.0\text{B}$ tokens) across three active-parameter ratios N_a/N—0.58 (low-to-medium sparsity, $N_a \approx 1.24\text{B}$), 0.27 ($N_a \approx 592\text{M}$), and 0.09 (high sparsity, $N_a \approx 188\text{M}$)—Step Law landed inside the flat near-optimal basin across all three sparsity regimes.
- **Transfer across data recipes:** Tested without re-fitting on three distinct training corpora—a **bilingual corpus**, a **code-integration corpus**, and a **code-dominant corpus**—Step Law consistently tracked the global minimum much closer than single-variable compute (C) or parameter-only (N) laws.

Even with a $0.94‰$ relative validation-loss error on dense sweeps, however, empirical hyperparameter laws suffer from a fundamental blind spot: **empirical curves predict loss, not numerical stability.**

When Clean Scaling Laws Blow Up: The Delphi Study

Consider the **Delphi scaling study** (Held), whose results are captured on a log-compute plot of Paloma Macro Loss (vertical axis) against training compute in FLOPs from 10^{19} to 10^{23} (horizontal axis), divided by a vertical dashed boundary at 3×10^{20} FLOPs. To the left of the boundary (3×10^{18} to 3×10^{20} FLOPs), six IsoFLOP-optimal runs lie right on top of a textbook straight power-law line. Every small-scale diagnostic looks flawless.

Now cross the dashed boundary into the extrapolation regime (10^{21} to 10^{23} FLOPs) while scaling under **Cautious AdamC paired with $\sqrt{B}$ learning-rate scaling**. Instead of continuing down the dashed power-law line, the actual runs silently drift above it and then collapse:

- At 10^{21} FLOPs: validation loss is 0.8% **worse** than predicted.
- At 10^{22} FLOPs: validation loss is 2.5% **worse** than predicted.
- At 10^{23} FLOPs: the run **diverged completely** (marked by a red $\times$ well above the trend line).

Fixing this failure did not come from refitting the power-law exponent; it required tighter parametrization, revised learning-rate scaling, and optimizer changes.

COMMON MISCONCEPTION

You might think that if your IsoFLOP parabolas from 3×10^{18} to 3×10^{20} FLOPs align on a razor-sharp straight line in log-log space, extrapolating $100\times$ to 10^{23} FLOPs is guaranteed to track that line. That breaks because small-scale runs can remain stably convergent even when activation norms, attention logits, or optimizer step scales are quietly growing with width and depth—drifting by 0.8% at 10^{21}, 2.5% at 10^{22}, and diverging catastrophically at 10^{23}. Clean small-scale parabolas never guarantee large-scale stability.

KEY IDEA

- Separating N and D in Step Law ($\eta_{\text{opt}} = 1.79N^{-0.713}D^{0.307}$, $B_{\text{opt}} = 0.58D^{0.571}$) cuts relative loss error to $0.94‰$ and transfers across MoE sparsity and data recipes under cosine decay, though the D-exponent shifts under WSD.
- Clean small-scale scaling curves predict loss inside the convergent regime, not stability at $100\times$–$1000\times$ larger compute—as the Delphi blow-up at 10^{23} FLOPs demonstrates, surviving extrapolation depends critically on optimizer choice and parametrization.

The Delphi failure shifts our focus squarely to **optimizers**: if Adam-family variants can silently drift and diverge at 10^{23} FLOPs, how do we fairly evaluate alternative pretraining optimizers across scale, and what optimizer replaced vanilla AdamW at the late-2026 frontier?

✎ Check Your Understanding

1. **What would happen if…** According to Step Law, if you keep model size N fixed and quadruple the training token budget $D \to 4D$ under a cosine schedule, how does the optimal batch size B_{opt} change, and why might the η_{opt} prediction fail if you switch to a WSD schedule?
   Answer: Optimal batch size scales by $4^{0.571} \approx 2.21\times$ (since $B_{\text{opt}} = 0.58D^{0.571}$). The η_{opt} prediction can fail under WSD because the positive $D^{0.307}$ exponent was fit under cosine decay and shifts when swapping to WSD (Zhou et al., 2026).
2. **Yes/No with justification:** If a scaling study shows textbook IsoFLOP parabolas with near-zero fit error up to 3×10^{20} FLOPs, does that ensure the same optimizer and LR rule will converge cleanly at 10^{23} FLOPs?
   Answer: No. As demonstrated in the Delphi study, clean small-scale fits do not guarantee large-scale stability; subtle scaling drift grew from 0.8% at 10^{21} to 2.5% at 10^{22} and caused complete divergence at 10^{23} FLOPs.

Section 6: Optimizer Benchmarking Pitfalls and Matrix Orthogonalization with Muon

Choosing a pretraining optimizer for a multi-trillion-token run is notoriously error-prone because optimizer papers frequently claim $1.5\times$ to $2\times$ speedups that vanish when deployed at frontier scale. Before introducing the matrix optimizer that took over frontier pretraining in 2025–2026, we need to understand the two experimental confounders documented by Wen et al. (*Fantastic Pretraining Optimizers and Where to Find Them*) that invalidate naive optimizer comparisons.

Pitfall 1: Untuned Baselines (Learning Rate and Weight Decay Confounders)

Apparent gains from a new optimizer often disappear the moment the AdamW baseline is tuned with the same care as the challenger:

1. **Learning rate confounder (130M model on C4/EN):** Using a common "default" AdamW learning rate of $\eta = 6 \times 10^{-4}$, training takes roughly $10{,}000$ steps to reach a validation loss of ~ 3.59. Simply raising AdamW's learning rate to $\eta = 8 \times 10^{-3}$ reaches that exact same loss in roughly $5{,}000$ steps—a $2\times$ **speedup** from tuning the baseline alone. That single adjustment completely eliminates the previously claimed speedup of Mars and Nesterov AdamW at the same compute budget.
2. **Weight decay confounder (Lion vs. AdamW):** Optimal weight decay differs sharply across optimizer families. In Wen et al.'s sweeps, AdamW's optimal weight decay sits near $\mathrm{wd} \approx 0.1$, whereas Lion's optimal weight decay sits near $\mathrm{wd} \approx 0.6$. Comparing two optimizers under shared hyperparameters systematically handicaps whichever optimizer is sitting further from its own optimum.

The engineering rule is unambiguous: different optimizers require independent hyperparameter sweeps—and because their update rules scale differently with matrix dimensions, they follow different scaling laws as well.

Pitfall 2: Scale Dependence Across Model Size

Even when every optimizer is carefully tuned at small scale, speedups measured on a 130M model do not stay constant as you scale up. Consider Wen et al.'s plot of **Token Speedup vs. AdamW** (vertical axis, with a horizontal dashed red baseline at $1.0\times$ representing tuned AdamW) across four model sizes on the horizontal axis—**130M, 300M, 520M, and 1.2B**—all trained at $8\times$ Chinchilla token-to-parameter ratio:

- **SOAP** (olive curve): starts at $1.40\times$ speedup at 130M, drops to $1.24\times$ at 300M and $1.29\times$ at 520M, and falls to $1.10\times$ at 1.2B.
- **Muon** (orange curve): starts at $1.38\times$ at 130M, drops to $1.26\times$ at 300M and $1.25\times$ at 520M, and falls to $1.10\times$ at 1.2B.
- **NAdamW** (dashed purple curve): starts at $1.18\times$ at 130M, drops to $1.06\times$ at 300M, and sits at $1.10\times$ at 520M and 1.2B.

Two structural conclusions emerge from this figure. First, **matrix-based optimizers** (Muon, SOAP, Kron) consistently outperform scalar elementwise optimizers (AdamW, Mars, NAdamW) across $1\times$ to $8\times$ Chinchilla ratios. Second, **token speedup shrinks substantially with model size**—Muon and SOAP drop

from roughly $1.4\times$ at 130M down to roughly $1.1\times$ at 1.2B. You must therefore evaluate any candidate optimizer across *both* model scale and token-to-parameter ratio before extrapolating compute savings.

COMMON MISCONCEPTION

You might think that a $1.4\times$ token speedup measured for Muon or SOAP on a 130M proxy model means you will save 40% of your compute budget on a 100B+ frontier run. That breaks because relative token speedups compress as model width and scale increase—dropping from $\sim 1.4\times$ at 130M to $\sim 1.1\times$ by 1.2B—so small-scale speedup ratios cannot be extrapolated flatly to frontier scale.

Muon: Orthogonalized Momentum for Matrix Parameters

If SOAP and Muon achieve nearly identical token speedups across scale, why did Muon—and not SOAP or Shampoo—become the standard optimizer for 2025–2026 frontier pretraining? The first part of the answer lies in how Muon works and how little per-step overhead it adds; the second part (stability) follows in Section 7.

Recall how standard elementwise optimizers like AdamW update a 2D weight matrix $\theta \in \mathbb{R}^{m\times n}$: they treat each of the mn matrix entries independently, normalizing each coordinate by an exponential moving average of its squared gradient. That elementwise view ignores the matrix structure of a linear layer—specifically, its singular values and singular vectors. In practice, gradient and momentum matrices in Transformers are often heavily ill-conditioned, with a few giant singular values dominating the update while dozens or hundreds of useful orthogonal feature directions receive tiny updates.

Muon replaces elementwise second-moment normalization on 2D hidden weight matrices with **matrix orthogonalization** of the momentum buffer. (Non-matrix and boundary parameters—embeddings, the LM head, and 1D parameters such as biases or norm parameters—still use AdamW.)

DEFINITION

Muon (MomentUm Orthogonalized by Newton–Schulz). A matrix optimizer for 2D hidden weight parameters that accumulates gradient momentum $B_t = \mu B_{t-1} + G_t$ and replaces B_t with an approximate polar orthogonalization $O_t = \mathrm{NewtonSchulz5}(B_t) \approx UV^\top$ before applying the update $\theta_t = \theta_{t-1} - \eta O_t$.

The complete algorithm for a 2D weight matrix θ with learning rate η and momentum coefficient μ is five lines:

```
Algorithm: Muon (2D matrix weights)
Require: learning rate η, momentum μ
  B_0 ← 0
  for t = 1, 2, ... do
    G_t ← ∇_θ L_t(θ_{t-1})
    B_t ← μ B_{t-1} + G_t
    O_t ← NewtonSchulz5(B_t)
    θ_t ← θ_{t-1} − η O_t
  end for
```

Examine the core mathematical operation, $\mathrm{NewtonSchulz5}(B_t)$. Recall from linear algebra that any real $m \times n$ matrix B_t admits a singular value decomposition (SVD):

$$B_t = USV^\top$$

where $U \in \mathbb{R}^{m \times r}$ has orthonormal columns (left singular vectors), $V \in \mathbb{R}^{n \times r}$ has orthonormal columns (right singular vectors), and $S = \mathrm{diag}(s_1, s_2, \ldots, s_r)$ holds the positive singular values. Muon wants the **nearest semi-orthogonal matrix** (the polar factor) to B_t, which replaces every nonzero singular value s_i by 1:

$$\mathrm{NewtonSchulz5}(B_t) \approx UV^\top$$

Computing an exact SVD at every optimizer step for every weight matrix in a large Transformer would be far too slow. Instead, `NewtonSchulz5` runs just **five matrix-multiply polynomial iterations** (after normalizing B_t so its singular values lie in $(0, 1]$) that map every nonzero singular value quickly toward 1 while leaving the singular vectors U and V untouched. Conceptually, Muon equalizes the update magnitude across all active orthogonal feature directions of a linear layer—boosting under-updated small-singular-value directions and capping dominant directions so the spectral norm of O_t stays bounded near 1.

WORKED EXAMPLE

Worked Example: Plain Momentum vs. Muon on an Ill-Conditioned 2×2 Momentum Matrix.

Suppose at step t a 2×2 weight matrix has momentum buffer:

$$B_t = \frac{1}{\sqrt{2}} \begin{bmatrix} 10 & -0.1 \\ 10 & 0.1 \end{bmatrix}$$

and learning rate $\eta = 0.01$.

1. **Compute the SVD $B_t = USV^\top$:** Check the two columns of B_t. Their inner product is $\frac{1}{2}(10(-0.1) + 10(0.1)) = 0$, so they are already orthogonal. The norm of column 1 is $\sqrt{50 + 50} = 10$, and the norm of column 2 is $\sqrt{0.005 + 0.005} = 0.1$. Therefore:

$$U = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}, \qquad S = \begin{bmatrix}10 & 0\\ 0 & 0.1\end{bmatrix}, \qquad V = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}.$$

The condition number of the momentum matrix is $s_1/s_2 = 10/0.1 = 100$.

2. **Compare plain momentum against Muon:**
   - **Plain momentum ($\Delta\theta = -\eta B_t$):** Moves by $0.01 \times 10 = 0.10$ along u_1 and by only $0.01 \times 0.1 = 0.001$ along u_2. The second feature direction learns $100\times$ slower than the first, and the update's spectral norm is $\|\Delta\theta\|_* = 0.10$.
   - **Muon ($\Delta\theta = -\eta O_t$ with $O_t = UV^\top$):** Replacing S with I_2 yields:

     $$O_t = UV^\top = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}.$$

     Every singular value of O_t is 1, so $\Delta\theta = -\eta O_t$ moves by exactly $\eta = 0.01$ along *both* orthogonal directions u_1 and u_2, with exact spectral norm $\|\Delta\theta\|_* = \eta = 0.01$.
3. **Interpretation:** Muon simultaneously caps the oversized dominant singular direction (preventing an overly large step along u_1) and boosts the suppressed second direction by $10\times$ relative to B_t ($100\times$ relative to u_1), while keeping the spectral norm of the matrix update tightly controlled. (We will see in Section 8 that controlling the spectral norm $\|\Delta W_l\|_*$ is also the exact theoretical heart of μP.)

Because `NewtonSchulz5` requires only five dense matrix multiplications and stores only a single momentum buffer B_t per matrix (rather than maintaining and inverting preconditioner matrices like Shampoo or SOAP), its per-step wall-clock overhead is remarkably small. In the NanoGPT speedrun benchmark comparing per-step wall-clock time:

- **Adam:** 139 ms/step
- **Muon:** 142 ms/step (only $\sim 2\%$ overhead over Adam, achieving the **fastest wall-clock convergence**)
- **DistributedShampoo:** 154–179 ms/step
- **SOAP:** 301 ms/step (more than $2\times$ slower per step than Muon)

Thus, while SOAP and Muon show similar *token* speedups in Wen et al.'s plot, SOAP's heavy per-step cost erases much of its wall-clock advantage, whereas Muon runs at almost the same step time as Adam.

KEY IDEA

- Valid optimizer benchmarking requires tuning each optimizer's learning rate and weight decay independently and measuring speedups across both model scale and token-to-parameter ratio, because token speedups compress with scale (from $\sim 1.4\times$ at 130M to $\sim 1.1\times$ at 1.2B for Muon and SOAP).
- Muon applies five Newton–Schulz matrix-multiply iterations to approximate the polar factor $UV^\top$ of each 2D momentum matrix, equalizing all nonzero singular values of the update at only $\sim 2\%$ per-step overhead over Adam (142 ms vs. 139 ms).

Still, if Muon's token speedup shrinks toward $1.1\times$ by 1.2B parameters, raw token efficiency alone does not explain why every major late-2026 frontier lab adopted Muon for trillion-parameter models. What actually drove Muon's takeover of frontier pretraining—and how do you run whole-matrix Newton–Schulz when a single weight matrix is sharded across many ranks? We answer both questions next.

✎ Check Your Understanding

1. **Fill-in-the-blank:** Given a 2D momentum matrix with singular value decomposition $B_t = USV^\top$, Muon's `NewtonSchulz5` step approximates the matrix _____, which has the effect of _____.
   Answer: $UV^\top$; equalizing all nonzero singular values of the update to 1.
2. **Yes/No with justification:** In Wen et al.'s 130M study, did Mars and Nesterov AdamW retain a $2\times$ speedup over AdamW once AdamW's learning rate was swept properly?
   Answer: No. Raising AdamW's learning rate from the default 6×10^{-4} to 8×10^{-3} gave AdamW itself a $2\times$ speedup (10,000 steps down to 5,000 steps to reach loss 3.59), eliminating the claimed gap.

Section 7: Frontier Training Stability and Distributed Muon

At trillion-parameter scale, a single unrecoverable loss spike or divergence can waste millions of dollars of training compute—the exact nightmare illustrated by the Delphi study at 10^{23} FLOPs in Section 5. Even when Muon's raw token speedup over tuned AdamW shrinks toward $1.1\times$ at larger sizes, **training stability** became the decisive reason Muon replaced vanilla AdamW across late-2026 frontier pretraining.

Solving Exploding Attention Logits: QK-Clip (MuonClip)

When Moonshot AI scaled Muon to 1T parameters in **Kimi K2** (July 2025, historical milestone for trillion-scale Muon), they encountered a concrete failure mode: **exploding attention logits**. Because Muon boosts small singular directions and takes aggressive spectral-norm steps in the query and key

projection matrices W_Q and W_K, the inner products $q^\top k/\sqrt{d_h}$ can grow unchecked over millions of tokens, saturating the attention softmax and triggering loss spikes.

> **DEFINITION**
> **QK-Clip (MuonClip).** A stability mechanism introduced in Kimi K2 (July 2025) that monitors the maximum attention logit per head during training and rescales the query and key projection weights whenever that maximum exceeds a threshold, bounding attention logits without altering the forward-pass architecture at inference.

Pairing Muon with QK-Clip enabled Kimi K2 to pretrain $1\mathrm{T}$ **total parameters on** $15.5\mathrm{T}$ **tokens with zero loss spikes**—a milestone that unlocked Muon for the entire industry, contrasting sharply with earlier late-2024 dense and MoE milestones (DeepSeek-V3 in Dec 2024 and Llama 3 in July 2024) that had relied on AdamW.

Quantifying Frontier Stability: Qwen3.8-Next (Aug 2026)

A year later, the **Qwen3.8-Next stability study** (Aug 2026) directly stress-tested AdamW against Muon and architectural normalization/residual gating at deliberately elevated learning rates—measuring **loss spikes per 10,000 steps**. Visualize the grouped bar chart from that study, split down the middle into two stress-test regimes:

- **Left group (**$4\times$ **optimal LR):** A towering red bar for **AdamW** hits **183 loss spikes per 10k steps**. Right beside it, a flat green bar for **Muon + gated residuals** sits at **0 loss spikes per 10k steps**.
- **Right group (**$3\times$ **optimal LR):** A light-blue bar for the **Baseline** normalization records **32.0 loss spikes per 10k steps**, while a short dark-blue bar for **GatedNorm** cuts that rate by exactly $10\times$, down to **3.2 loss spikes per 10k steps**.

Look at what these numbers tell you operationally: Muon paired with gated residual connections and GatedNorm widens the stable learning-rate margin by several factors, turning a regime where AdamW spikes 183 times every 10k steps into a completely spike-free run.

Late-2026 Frontier Adoption

Across all three major open-weight frontier model families in late 2026, **matrix orthogonalization (Muon) paired with residual and normalization gating** replaced vanilla AdamW + pre-norm:

Model (Released)	Scale & MoE / Attention Structure	Optimizer & Stability Choices
DeepSeek-V4 (2026)	Hybrid CSA (compressed sparse attention) + HCA (heavily compressed attention)	Uses **Muon** (explicitly credited with training stability) paired with **mHC** residual connections
Kimi K3 / K3.1 (July 2026)	2.8T total / 104B active; 896 routed experts (16 active, 2 shared), Stable LatentMoE; 3 Kimi Delta Attention : 1 gated MLA	**Per-Head Muon** + **Attention Residuals**
Qwen3.8-Flash-Next (Aug 2026)	125B total / 6B active; 512 experts (10 active + 1 shared); $3:1$ Gated DeltaNet + Qwen Sparse Attention (QSA)	Backed by Qwen3.8-Next's **Muon + gated residual / GatedNorm** stability study

Distributed Muon in Practice

Adopting Muon at trillion-parameter scale creates an immediate distributed-systems problem. Recall how standard data-parallel optimizer sharding (such as ZeRO / FSDP) works for AdamW: because AdamW is purely **elementwise**, you can slice every parameter matrix and its optimizer states arbitrarily across P ranks, and each rank updates its local 1D slice independently with zero extra communication during the optimizer step.

Newton–Schulz breaks elementwise sharding because computing $O_t = \text{NewtonSchulz5}(B_t)$ requires matrix multiplications over the **entire 2D matrix** B_t. If B_t is sliced across four ranks, no single rank holds the full rows and columns needed to orthogonalize B_t locally.

> **COMMON MISCONCEPTION**
> You might think you can avoid extra communication in distributed Muon by running Newton–Schulz independently on each rank's local shard of a parameter matrix. That breaks because orthogonalizing sub-blocks of a matrix does not produce the polar factor $UV^\top$ of the full matrix—singular vectors couple rows and columns globally across the entire weight matrix, so blockwise orthogonalization alters the optimizer dynamics and loses Muon's spectral properties.

The standard distributed solution is **owner-rank assignment**. Trace the three-stage dataflow across four ranks (Ranks 0, 1, 2, and 3) holding shards of two matrices W_1 and W_2:

1. **Top row (sharded states):** Ranks 0, 1, 2, and 3 each hold slices of the momentum states for both W_1 and W_2.
2. **Middle stage (gather + parallel whole-matrix Newton–Schulz):** Full 2D momentum matrices are gathered onto distinct **owner ranks**—for example, all shards of B_1 gather onto the owner rank assigned to W_1, while simultaneously all shards of B_2 gather onto a different owner rank assigned to W_2. Because a Transformer has hundreds of distinct 2D parameter matrices, different owner ranks

run full `NewtonSchulz5` computations **in parallel across different matrices**, keeping every rank busy.

3. **Bottom stage (scatter / broadcast back):** Once each owner rank finishes computing $O_1 = \mathrm{NewtonSchulz5}(B_1)$ and $O_2 = \mathrm{NewtonSchulz5}(B_2)$, the orthogonalized updates are scattered or broadcast back to the sharded parameter layout to apply $\theta_t \leftarrow \theta_{t-1} - \eta O_t$.

> **WORKED EXAMPLE**
> **Worked Example: Owner-Rank Load Balancing for Distributed Muon.**
>
> Suppose a model has 256 distinct 2D hidden weight matrices (all of equal size $d \times d$) distributed across $P = 64$ data-parallel sharding ranks.
>
> 1. **Assignment:** Assign $256/64 = 4$ complete matrices to each rank as its owned set.
> 2. **Execution per optimizer step:**
>   - During the gather phase, each rank receives the remaining $63/64$ shards of its 4 owned matrices while sending out its local shards of the other 252 matrices (a balanced all-to-all / reduce-scatter + all-gather communication pattern).
>   - During the compute phase, all 64 ranks execute `NewtonSchulz5` simultaneously on their 4 complete $d \times d$ matrices—achieving 100% parallel load balance across ranks without ever splitting a single Newton–Schulz call across ranks.
>   - During the scatter phase, each rank returns the updated shards of its 4 matrices to their home shards (or applies the update directly if parameter ownership matches).
> 3. **Interpretation:** Assigning whole matrices to distinct owner ranks amortizes Newton–Schulz compute evenly across the cluster, turning a matrix-global operation into parallel local matrix multiplications.

How expensive is this communication and compute in practice? NVIDIA reports (vendor-reported) that distributed Muon runs within a **few percent of AdamW throughput on GB300**. To reduce communication and orthogonalization overhead even further on larger clusters or bandwidth-constrained interconnects, recent compressed distributed variants—**Dion2/Dion3** and **MuonBP**—orthogonalize only part of each matrix (such as low-rank or selected subspaces) or synchronize orthogonalized updates less frequently across steps.

KEY IDEA

- Muon won late-2026 frontier pretraining primarily because of **stability**, not just small-scale token speedup: paired with QK-Clip (Kimi K2, zero spikes over 1T/15.5T) and gated residuals/GatedNorm (Qwen3.8-Next, 0 vs. 183 spikes per 10k steps at $4\times$ LR), it powers DeepSeek-V4, Kimi K3/K3.1, and Qwen3.8-Flash-Next.
- Distributed Muon solves the whole-matrix requirement of Newton–Schulz by assigning each 2D parameter matrix to an **owner rank** so orthogonalization runs in parallel across matrices, achieving throughput within a few percent of AdamW (vendor-reported on GB300), with compressed variants (Dion2/Dion3, MuonBP) trimming overhead further.

We now have cheap sweeps (WSD), empirical hyperparameter laws (Step Law), and a stable frontier optimizer (Muon). Yet one foundational question we flagged back in Section 2 remains unanswered: *why* do optimal learning rates drift downward as width grows in standard parametrization, and what exact mathematical rules guarantee width-invariant feature learning? We derive Maximal Update Parametrization (μP) from first principles next.

✎ Check Your Understanding

1. **Quick recall / comparison:** In the Qwen3.8-Next stability study (Aug 2026), how did loss spikes per 10k steps compare between AdamW and Muon + gated residuals at $4\times$ optimal LR, and what factor reduction did GatedNorm provide at $3\times$ optimal LR?
   Answer: At $4\times$ optimal LR, AdamW had 183 spikes per 10k steps while Muon + gated residuals had 0. At $3\times$ optimal LR, GatedNorm cut spikes by $10\times$ (from 32.0 down to 3.2 per 10k steps).
2. **What would happen if…** Why can distributed AdamW update sharded parameter tensors purely locally on each rank, whereas distributed Muon requires gathering full 2D matrices onto owner ranks?
   Answer: AdamW's update is elementwise, so each scalar entry depends only on its own gradient and momentum moments; Muon's Newton–Schulz iteration computes matrix products over the entire 2D momentum matrix to approximate its global polar factor $UV^\top$.

Section 8: Maximal Update Parametrization (μP) — Spectral Derivation from First Principles

Compare two side-by-side plots of validation loss versus $\log_2(\text{Learning Rate})$ as model width scales from $w = 128$ up to $w = 8192$. On the left, under **Standard Parametrization (SP)**, three U-shaped loss curves shift steadily to the left as width grows: the optimal learning rate for $w = 128$ lies far to the right, and by $w = 8192$ the optimum has drifted orders of magnitude lower while the old $w = 128$

optimum has blown up off the top of the chart. On the right, under **μP (Width-Invariant)**, the three U-shaped curves stack vertically on top of one another, dropping lower in loss as width increases while sharing a single vertical dashed line at the exact same optimal learning rate.

This stability does not merely save grid-search compute—it also makes scaling trajectories themselves much cleaner. Consider the empirical evidence from **Cerebras-GPT**, which trained dense models from 0.1B to 13B parameters under the Chinchilla recipe. Plotting the **percentage loss deviation from the fitted scaling law** against training FLOPs ($10^{18} \to 10^{22}$), the standard-parametrization runs zig-zag erratically by roughly $\pm 1\%$ above and below the zero-deviation line (for example, SP 6.7B spikes to $+1\%$ worse than the power law, while SP 13B drops below 0%). By contrast, tuning hyperparameters once on a tiny 40M **μP proxy** and transferring zero-shot up to 2.7B produces a smooth, flat trajectory consistently *below* the baseline scaling curve.

How does μP achieve width invariance? Look first at the empirical summary table for scaling a network's width from base width M to target width M' by a factor $r = M'/M$ under AdamW:

Hyperparameter	Base Model M	Target Model $M' \sim rM$
AdamW LR (matrix-like)	l	l/r
AdamW LR (others)	l	l
Init variance (matrix-like)	σ^2 (slide notation σ)	σ^2/r (σ/r)
Init variance (others)	σ^2 (σ)	σ^2 (σ)
Multiplier (output / LM head)	τ	τ/r
Multiplier (others)	τ	τ

Here, **"matrix-like"** means 2D tensors whose *both* dimensions grow with width (such as QKV, attention output, and MLP projections); embeddings count as "others" because their vocabulary dimension is fixed rather than growing with width (and note that the LM head output multiplier τ/r absorbs the extra fan-in scaling when mapping width M back down to fixed vocabulary size).

Rather than memorizing this table as a bag of ad-hoc tricks, we can derive every single rule cleanly from two elementary spectral conditions due to Yang, Simon, and Bernstein.

The Spectral Conditions for Feature Learning (A1 and A2)

Consider a layer l of width n_l with pre-activation vector $h_l \in \mathbb{R}^{n_l}$. If every coordinate of h_l has typical magnitude $\Theta(1)$—neither vanishing toward zero nor exploding to infinity as width $n_l \to \infty$—then its Euclidean norm is $\|h_l\|_2 = \sqrt{\sum_{i=1}^{n_l} (h_{l,i})^2} = \Theta(\sqrt{n_l})$. The exact same reasoning applies to the *change* in features Δh_l produced by one gradient step.

DEFINITION

Maximal Update Parametrization (μP) spectral conditions. A parametrization of a width-n_l network satisfies μP if two conditions hold across every layer l as width grows:

- **Condition A1 (Initialization):** Each coordinate of h_l is $\Theta(1) \iff \|h_l\|_2 = \Theta(\sqrt{n_l})$.
- **Condition A2 (One gradient step):** Each coordinate of Δh_l is $\Theta(1) \iff \|\Delta h_l\|_2 = \Theta(\sqrt{n_l})$.

Visualize the forward flow from **Layer $l-1$** to **Layer l**. Layer $l-1$ enters with $\|h_{l-1}\|_2 = \Theta(\sqrt{n_{l-1}})$ and update $\|\Delta h_{l-1}\|_2 = \Theta(\sqrt{n_{l-1}})$. An arrow labeled $W_l + \Delta W_l$ carries activations into Layer l, where we require $\|h_l\|_2 = \Theta(\sqrt{n_l})$ and $\|\Delta h_l\|_2 = \Theta(\sqrt{n_l})$. As we are about to prove, preserving both norms requires **both W_l and ΔW_l to have spectral norm $\Theta(\sqrt{n_l/n_{l-1}})$**.

What goes wrong if Condition A1 or A2 fails?

- **Too small** ($\|\Delta h_l\|_2 \ll \sqrt{n_l}$): The network falls into the **lazy / kernel regime**, where internal features barely move from initialization as $n_l \to \infty$ and the network stops learning useful representations.
- **Too large** ($\|h_l\|_2 \gg \sqrt{n_l}$ or $\|\Delta h_l\|_2 \gg \sqrt{n_l}$): Activations and output logits blow up as width increases, destabilizing training.

μP is called *maximal* update parametrization because $\Delta h_l = \Theta(1)$ per coordinate is the *largest* feature update size that does not explode as width goes to infinity.

Deriving Condition A1 (Initialization Standard Deviation σ_l)

Model the network as a deep linear network $h_l = W_l h_{l-1}$, where $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$ has i.i.d. Gaussian entries $W_{l,ij} \sim \mathcal{N}(0, \sigma^2)$ (written compactly as $W_l \sim \mathcal{N}(0, \sigma^2 I_{n_l \times n_{l-1}})$). Recall that the **spectral norm** $\|W_l\|_*$ (the largest singular value of W_l) is the operator norm $\max_{x \neq 0} \|W_l x\|_2 / \|x\|_2$.

1. **Random matrix concentration:** Recall from random matrix theory (the Marchenko–Pastur / Gordon–Slepian bound for an $n_l \times n_{l-1}$ Gaussian matrix with entry standard deviation σ) that the spectral norm concentrates sharply around:

$$\|W_l\|_* \to \sigma\left(\sqrt{n_{l-1}} + \sqrt{n_l}\right)$$

2. **Inductive forward norm condition:** Assume inductively that the previous layer satisfies A1, so $\|h_{l-1}\|_2 = \Theta(\sqrt{n_{l-1}})$. Bounding the forward propagation via the worst-case spectral relation $\|h_l\|_2 \approx \|W_l\|_* \|h_{l-1}\|_2$, achieving $\|h_l\|_2 = \Theta(\sqrt{n_l})$ requires:

$$\|W_l\|_* \cdot \sqrt{n_{l-1}} = \Theta(\sqrt{n_l}) \implies \|W_l\|_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$$

3. **Solving for σ:** Equating the two expressions for $\|W_l\|_*$:

$$\sigma\left(\sqrt{n_{l-1}}+\sqrt{n_l}\right)=\Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)\implies\sigma=\frac{\sqrt{n_l}}{\sqrt{n_{l-1}}\left(\sqrt{n_l}+\sqrt{n_{l-1}}\right)}$$

Now simplify the denominator cleanly by noticing that for any two positive numbers a, b, $a+b=\Theta(\max(a,b))$:

- If $n_l \geq n_{l-1}$ (fan-out $\geq$ fan-in), then $\sqrt{n_l}+\sqrt{n_{l-1}}=\Theta(\sqrt{n_l})$, so the $\sqrt{n_l}$ factors cancel and $\sigma=\Theta(1/\sqrt{n_{l-1}})$.
- If $n_l < n_{l-1}$ (fan-out $<$ fan-in), then $\sqrt{n_l}+\sqrt{n_{l-1}}=\Theta(\sqrt{n_{l-1}})$, so $\sigma=\Theta(\sqrt{n_l}/n_{l-1})=\Theta\left(\frac{1}{\sqrt{n_{l-1}}}\sqrt{\frac{n_l}{n_{l-1}}}\right)$.

Combining both cases into a single formula yields the μP initialization rule:

$$\sigma=\frac{\sqrt{n_l}}{\sqrt{n_{l-1}}\left(\sqrt{n_l}+\sqrt{n_{l-1}}\right)}=\Theta\left(\frac{1}{\sqrt{n_{l-1}}}\min\left(1,\sqrt{\frac{n_l}{n_{l-1}}}\right)\right)$$

Notice what this worst-case spectral derivation reveals: whenever fan-out equals or exceeds fan-in ($n_l \geq n_{l-1}$, such as embeddings or square hidden layers), $\sigma=\Theta(1/\sqrt{n_{l-1}})$ matches standard Xavier/He initialization. Whenever fan-out is *smaller* than fan-in ($n_l < n_{l-1}$)—most critically at the **unembedding / LM head**, where $n_l=n_{\text{vocab}}$ is fixed while fan-in $n_{l-1}=M$ grows—σ must shrink further, as $\Theta(1/n_{l-1})$ (so variance $\sigma^2=\Theta(1/M^2)$)!

Deriving Condition A2, Part 1 (Weight Update Spectral Norm $\|\Delta W_l\|_*$)

Now take one SGD step with layer learning rate η_l on a single input example with loss ℓ. By the chain rule on $h_l=W_l h_{l-1}$, the gradient $\nabla_{W_l}\ell=(\nabla_{h_l}\ell)h_{l-1}^\top$ is a **rank-1 outer product**, so the weight update is also rank-1:

$$\Delta W_l=-\eta_l\left(\nabla_{h_l}\ell\right)h_{l-1}^\top$$

Check a crucial linear-algebra fact about this outer product. For any rank-1 matrix $uv^\top$, its sole nonzero singular value is $\|u\|_2\|v\|_2$, its sole right singular vector is $v/\|v\|_2$, and applying $uv^\top$ to v itself achieves the spectral-norm bound *with exact equality*: $\|(uv^\top)v\|_2=\|u\|_2\|v\|_2^2=(\|u\|_2\|v\|_2)\|v\|_2=\|uv^\top\|_*\|v\|_2$. Setting $u=-\eta_l\nabla_{h_l}\ell$ and $v=h_{l-1}$ immediately gives exact alignment:

$$\|\Delta W_l h_{l-1}\|_2=\|\Delta W_l\|_*\|h_{l-1}\|_2$$

After this step, the weight becomes $W_l+\Delta W_l$ and the incoming activation becomes $h_{l-1}+\Delta h_{l-1}$. Expanding $(W_l+\Delta W_l)(h_{l-1}+\Delta h_{l-1})-W_l h_{l-1}$ gives the exact activation shift:

$$\Delta h_l=W_l\Delta h_{l-1}+\Delta W_l h_{l-1}+\Delta W_l\Delta h_{l-1}=W_l\Delta h_{l-1}+\Delta W_l\left(h_{l-1}+\Delta h_{l-1}\right)$$

Evaluate the norm of each of the three terms, assuming the leading terms do not cancel:

1. **Propagated prior shift ($W_l \Delta h_{l-1}$):** By induction, $\|\Delta h_{l-1}\|_2 = \Theta(\sqrt{n_{l-1}})$, and by Condition A1, $\|W_l\|_* = \Theta(\sqrt{n_l/n_{l-1}})$. Their product automatically has norm $\Theta(\sqrt{n_l})$.
2. **New update on old features ($\Delta W_l h_{l-1}$):** Because of the exact rank-1 alignment proved above, $\|\Delta W_l h_{l-1}\|_2 = \|\Delta W_l\|_* \|h_{l-1}\|_2 = \|\Delta W_l\|_* \Theta(\sqrt{n_{l-1}})$. For this term to be $\Theta(\sqrt{n_l})$, we must enforce:

$$\|\Delta W_l\|_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$$

3. **Second-order cross term ($\Delta W_l \Delta h_{l-1}$):** Bounded by $\|\Delta W_l\|_* \|\Delta h_{l-1}\|_2 = O\big(\|\Delta W_l\|_* \sqrt{n_{l-1}}\big)$, which automatically matches $\Theta(\sqrt{n_l})$ once condition (2) holds.

This yields the **spectral update condition** of μP:

$$\|\Delta W_l\|_* = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right)$$

Deriving Condition A2, Part 2 (Learning Rate η_l for SGD and Adam)

We now know the exact spectral norm required for ΔW_l, but how do we translate $\|\Delta W_l\|_* = \Theta(\sqrt{n_l/n_{l-1}})$ into a layer learning rate η_l? The answer depends on the optimizer.

Case 1: SGD

Require that one gradient step changes the scalar loss by a width-independent amount $\Delta \ell = \Theta(1)$. By first-order Taylor expansion, $\Delta \ell \approx \langle \Delta W_l, \nabla_{W_l} \ell \rangle$ (Frobenius inner product). Because $\Delta W_l = -\eta_l \nabla_{W_l} \ell$ is parallel to $\nabla_{W_l} \ell$ and both matrices have **rank 1**, the Frobenius norm and the spectral norm coincide ($\|uv^\top\|_F = \|uv^\top\|_* = \|u\|_2 \|v\|_2$):

$$\Delta \ell \approx \Theta(\langle \Delta W_l, \nabla_{W_l} \ell \rangle) = \Theta(\|\Delta W_l\|_F \|\nabla_{W_l} \ell\|_F) = \Theta(\|\Delta W_l\|_* \|\nabla_{W_l} \ell\|_*)$$

Substitute $\Delta \ell = \Theta(1)$ and $\|\Delta W_l\|_* = \Theta(\sqrt{n_l/n_{l-1}})$ to solve for the gradient's spectral norm:

$$\|\nabla_{W_l} \ell\|_* = \Theta\left(\sqrt{\frac{n_{l-1}}{n_l}}\right)$$

Finally, since $\Delta W_l = -\eta_l \nabla_{W_l} \ell$, taking spectral norms gives $\|\Delta W_l\|_* = \eta_l \|\nabla_{W_l} \ell\|_*$, so:

$$\eta_l^{\text{SGD}} = \frac{\|\Delta W_l\|_*}{\|\nabla_{W_l} \ell\|_*} = \Theta\left(\frac{\sqrt{n_l/n_{l-1}}}{\sqrt{n_{l-1}/n_l}}\right) = \Theta\left(\frac{n_l}{n_{l-1}}\right)$$

Case 2: Adam / AdamW

Adam does not step proportionally to the raw gradient magnitude; it divides each entry of the gradient (or momentum) by its root-mean-square magnitude, making every entry of the update ΔW_l sign-like with typical magnitude $\Theta(\eta_l)$. For a structured (aligned, low-rank-correlated) $n_l \times n_{l-1}$ matrix whose

entries all have magnitude $\Theta(\eta_l)$, its Frobenius norm is $\sqrt{n_l n_{l-1} \cdot \eta_l^2} = \Theta(\eta_l \sqrt{n_l n_{l-1}})$, and because the sign of a rank-1 outer product $\operatorname{sign}(uv^\top) = \operatorname{sign}(u)\operatorname{sign}(v)^\top$ is *still* rank-1, its spectral norm equals its Frobenius norm:

$$\|\Delta W_l\|_* = \Theta(\eta_l \sqrt{n_l n_{l-1}})$$

Matching this to the spectral update condition $\|\Delta W_l\|_* = \Theta(\sqrt{n_l / n_{l-1}})$:

$$\eta_l \sqrt{n_l n_{l-1}} = \Theta\left(\sqrt{\frac{n_l}{n_{l-1}}}\right) \implies \eta_l^{\text{Adam}} = \Theta\left(\frac{1}{n_{l-1}}\right)$$

Look how cleanly the $\sqrt{n_l}$ cancels out for Adam! Every single rule from MiniCPM (Section 2) and the width-scaling table at the top of this section just fell out of two lines of linear algebra.

Side-by-Side Summary: μP vs. Standard Parametrization

Rule	Standard Parametrization (SP)	μP
Init stdev σ_l	$\Theta\left(\frac{1}{\sqrt{n_{l-1}}}\right)$	$\Theta\left(\frac{1}{\sqrt{n_{l-1}}} \min\left(1, \sqrt{\frac{n_l}{n_{l-1}}}\right)\right)$
SGD LR η_l	$\Theta(1)$	$\Theta\left(\frac{n_l}{n_{l-1}}\right)$
Adam LR η_l	$\Theta(1)$	$\Theta\left(\frac{1}{n_{l-1}}\right)$

Two concrete differences separate μP from Standard Parametrization:

1. **Adam learning rate** shrinks as $1/n_{l-1}$ for hidden and output matrices instead of staying constant as width grows.
2. **Initialization standard deviation** shrinks *further* below $1/\sqrt{n_{l-1}}$ whenever fan-out $n_l <$ fan-in n_{l-1}—most importantly at the **unembedding / LM head**, where vocabulary size $n_l = n_{\text{vocab}}$ is fixed while n_{l-1} grows with model width.

COMMON MISCONCEPTION

You might think that standard He/Xavier initialization ($\sigma_l = 1/\sqrt{n_{l-1}}$) already keeps every layer's activations $\Theta(1)$ per coordinate at initialization, so changing initialization is unnecessary for μP. That breaks at the unembedding / LM head (and after the first gradient step): at the LM head, $n_{l-1} = M$ grows while $n_l = V$ is fixed. Under worst-case spectral alignment (which occurs immediately once features correlate with output weights during training), a weight matrix with $\sigma = 1/\sqrt{M}$ has spectral norm $\Theta(1)$, mapping an input vector of norm $\Theta(\sqrt{M})$ to logits of norm $\Theta(\sqrt{M})$—blowing up the logits as width grows unless $\sigma = \Theta(1/M)$ (or an explicit $1/r$ logit multiplier is applied).

WORKED EXAMPLE

Worked Example: Computing μP Scaling for Embedding, Hidden, and LM Head Layers ($M = 128 \to 2048$).

Suppose vocabulary size is fixed at $V = 32{,}768$ (treated as $\Theta(1)$ with respect to width scaling), base proxy width is $P = 128$, and target width is $M = 2048$, giving width scale factor $r = M/P = 16$. Evaluate σ_l and η_l^{Adam} for all three matrix types:

1. **Embedding ($W^E \in \mathbb{R}^{M \times V}$, treated as lookup from one-hot input of effective fan-in $n_{l-1} = 1$, fan-out $n_l = M$):**
   - Since $n_l = M \geq n_{l-1} = 1$, $\min(1, \sqrt{M/1}) = 1$, so $\sigma_E = \Theta(1/\sqrt{1}) = \Theta(1)$ (variance $\Theta(1)$, unchanged as r grows).
   - $\eta_E^{\text{Adam}} = \Theta(1/n_{l-1}) = \Theta(1)$ (unchanged as r grows).
2. **Hidden square layer ($W_h \in \mathbb{R}^{M \times M}$, $n_{l-1} = M$, $n_l = M$):**
   - $\min(1, \sqrt{M/M}) = 1$, so $\sigma_h = \Theta(1/\sqrt{M})$ (variance scales as $1/M$, dropping by $16\times$ from 128 to 2048).
   - $\eta_h^{\text{Adam}} = \Theta(1/M)$, so the AdamW learning rate drops by $r = 16\times$ (e.g., from α to $\alpha/16$).
3. **Unembedding / LM head ($W^U \in \mathbb{R}^{V \times M}$, fan-in $n_{l-1} = M$, fixed fan-out $n_l = V$):**
   - Treating $V = \Theta(1)$ relative to growing M, $\min(1, \sqrt{V/M}) = \Theta(1/\sqrt{M})$, so:

     $$\sigma_U = \Theta\left(\frac{1}{\sqrt{M}} \cdot \frac{1}{\sqrt{M}}\right) = \Theta\left(\frac{1}{M}\right) \implies \text{init variance } \sigma_U^2 = \Theta\left(\frac{1}{M^2}\right).$$

     From $M = 128$ to $M = 2048$, the LM head initialization variance drops by $r^2 = 256\times$ (or equivalently, keep variance $\propto 1/M$ and multiply output logits by $1/r = 1/16$, as in MiniCPM).
   - $\eta_U^{\text{Adam}} = \Theta(1/M)$, dropping by $r = 16\times$.
4. **Interpretation:** Every single scaling factor—1 for embeddings, $1/M$ for hidden variance and LR, and $1/M^2$ for LM head variance with $1/M$ LR—is fixed uniquely by the spectral condition $\|W_l\|_* = \|\Delta W_l\|_* = \Theta(\sqrt{n_l/n_{l-1}})$.

KEY IDEA

- μP keeps both features h_l and feature updates Δh_l at $\Theta(1)$ per coordinate (Euclidean norm $\Theta(\sqrt{n_l})$), preventing both the lazy/kernel regime and activation blow-up.
- Enforcing $\|W_l\|_* = \|\Delta W_l\|_* = \Theta(\sqrt{n_l/n_{l-1}})$ directly derives $\sigma_l = \Theta\left(\frac{1}{\sqrt{n_{l-1}}} \min(1, \sqrt{n_l/n_{l-1}})\right)$, SGD $\eta_l = \Theta(n_l/n_{l-1})$, and Adam $\eta_l = \Theta(1/n_{l-1})$.

The derivation above used a clean deep linear network. Real language models are full Transformers with multi-head attention, SwiGLU MLPs, RMSNorm gains, projection biases, decoupled weight decay, and—in late 2026—Muon and sparse MoE layers. How do these spectral rules translate to every parameter matrix

in a Transformer, where does μP actually break in practice, and how did late-2026 research fix those failures?

> **✎ Check Your Understanding**
>
> 1. **Quick computation:** For a rectangular hidden layer with fan-in $n_{l-1} = 4M$ (such as an MLP down-projection) and fan-out $n_l = M$, what are the μP scalings (in Θ-notation with respect to M) for initialization standard deviation σ_l, SGD learning rate, and Adam learning rate?
> **Answer:** Since $n_l/n_{l-1} = 1/4$ is constant in M, $\sigma_l = \Theta(1/\sqrt{M})$ (with an extra constant factor $\sqrt{1/4} = 1/2$ from $\min(1, \sqrt{n_l/n_{l-1}})$), SGD $\eta_l = \Theta(n_l/n_{l-1}) = \Theta(1)$, and Adam $\eta_l = \Theta(1/n_{l-1}) = \Theta(1/M)$.
>
> 2. **Fill-in-the-blank:** Under Condition A2, if feature updates $\|\Delta h_l\|_2$ shrink much faster than $\Theta(\sqrt{n_l})$ as width grows, training collapses into the _____ regime; if they grow faster than $\Theta(\sqrt{n_l})$, training suffers _____.
> **Answer:** lazy / kernel; activation and logit blow-up (divergence).

Section 9: μP in Real Transformers — Empirical Transfer, Failure Modes, and Late-2026 Extensions

Applying the linear-network derivation from Section 8 to an actual Transformer requires checking every weight matrix—plus one crucial nonlinear interaction inside self-attention. Lingle (*A Large-Scale Exploration of μ-Transfer*) laid out the complete Transformer μP specification for a model of residual width M, base proxy width P, MLP inner width F (typically $F = 4M$), H attention heads of dimension D (with $HD = M$), and base AdamW learning rate α:

Parameter	Role	Init Var (Θ)	Adam LR (Θ)	Init Var (Exact)	Adam LR (Exact)
W^E	Embedding	1	1	1	α
W^{AQ}, W^{AK}, W^{AV}	Attention QKV	$1/M$	$1/M$	$1/M$	$\alpha P/M$
W^{AO}	Attention output	$1/(HD)$	$1/(HD)$	$1/M$	$\alpha P/M$
W^{FI}	MLP input	$1/M$	$1/M$	$1/M$	$\alpha P/M$
W^{FO}	MLP output	$1/F$	$1/F$	$0.25/M$	$\alpha P/M$
W^U	**LM head (unembed)**	$1/M^2$	$1/M$	$1/M^2$	$\alpha P/M$

Notice how every single entry matches our Section 8 formulas: embeddings have fan-in 1 (variance 1, LR α); every internal 2D matrix has fan-in proportional to M (variance $1/M$, except W^{FO} with fan-in $F = 4M$ giving exact init variance $1/(4M) = 0.25/M$, and exact Adam LR $\alpha P/M$); and the unembedding matrix W^U has init variance $1/M^2$ with Adam LR $\alpha P/M$.

One additional change is required inside scaled dot-product attention: **attention logit scaling uses $\tau^{-1} = 1/D$ instead of the standard $1/\sqrt{D}$**. Why? At random initialization, query vector $q \in \mathbb{R}^D$ and key vector $k \in \mathbb{R}^D$ have independent $\Theta(1)$ coordinates, so their inner product $q^\top k = \sum_{j=1}^{D} q_j k_j$ has standard deviation $\Theta(\sqrt{D})$—which is why standard attention divides by $\sqrt{D}$. Once feature learning begins under μP, however, q and k become correlated across coordinates, causing $q^\top k$ to scale as $\Theta(D)$ rather than $\Theta(\sqrt{D})$. Dividing by D instead of $\sqrt{D}$ keeps attention logits $\Theta(1)$ during feature learning. Empirically, Lingle found that even when head dimension D is held fixed across widths, using $\tau^{-1} = 1/D$ has a major positive impact on both hyperparameter transfer and final model quality.

Does μP Transfer in Practice? Lingle's Replication

Lingle directly tested standard parametrization against baseline μP across widths 128, 512, and 2048 (a $16\times$ width span) over a grid of base learning rates from 2^{-10} to 2^{-2}.

Standard Parametrization (SP) — validation loss (optimum drifts left and blows up):

Width	2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}
128	3.841	3.757	**3.706**	3.879	4.030
512	3.013	**2.967**	2.987	3.383	*7.403*
2048	**2.738**	2.902	*7.247*	*7.477*	*7.314*

Look at what happens in SP: the optimal learning rate marches leftward at every width increase ($2^{-6} \to 2^{-8} \to 2^{-10}$), and by width 2048, the old width-128 optimum (2^{-6}) completely diverges to loss 7.247.

Baseline μP — validation loss (locked at 2^{-6} across $16\times$ width):

Width	2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}
128	3.846	3.743	**3.695**	3.884	4.143
512	3.114	2.993	**2.953**	3.221	3.506
2048	2.711	2.553	**2.511**	2.563	3.244

Two striking results stand out here. First, the optimal learning rate stays locked on 2^{-6} across the entire $16\times$ width span with zero blow-ups even at 2^{-2}. Second, at width 2048, μP does not merely transfer cleanly—it achieves a dramatically better final validation loss (2.511) than SP's best tuned run (2.738)!

Scaling further from a **2M-parameter proxy** ($M = 128$) all the way to a **10B-parameter model** ($M = 8192$)—a $5000\times$ **jump in parameter count**—μP remained locked at the exact same base learning rate 2^{-6}:

Params	Width	2^{-8}	2^{-6}	2^{-4}
2M	128	3.791	**3.766**	3.814
40M	512	3.016	**2.983**	3.004
600M	2048	2.513	**2.459**	2.466
10B	**8192**	2.238	**2.167**	2.169

What μP Is Robust To — and What Breaks It

Lingle's ablation suite mapped out precisely which modern Transformer components preserve μP transfer and which ones quietly break it:

- **Robust (transfer holds ✓):**
  - Modern activations: **SwiGLU** and **squared ReLU**
  - **Large and small batch sizes**
  - Initialization tweaks (such as zero-initializing attention output projections)
  - **Projection biases**
- **Failures in Lingle's ablations (transfer breaks ✗):**

Component	Best LR ($128 \to 512 \to 2048$)	Mechanism & Practical Takeaway
RMSNorm gains (vector / scalar)	Vector: $2^{-4} \to 2^{-4} \to 2^{-8}$ Scalar: $2^{-4} \to 2^{-4} \to 2^{-6}$	Applying a $\Theta(1)$ learning rate to learnable norm gains breaks width invariance; trainable gains also slightly hurt the largest μP models and can simply be dropped.
Lion optimizer	$2^{-10} \to 2^{-8} \to 2^{-8}$	Lion's sign-based update rules violate Adam μP scaling assumptions.
Decoupled weight decay (0.1)	$2^{-8} \to 2^{-6} \to 2^{-6}$	**Biggest practical failure:** the decay step $-\eta_t \lambda \theta_{t-1}$ shrinks weights unevenly across widths unless the weight-decay coefficient λ scales as well.

Examine *why* decoupled weight decay is the biggest practical failure. In AdamW, a parameter matrix updates as:

$$\theta_t = \theta_{t-1} - \eta_t \lambda \theta_{t-1} - \eta_t \hat{u}_t = (1 - \eta_t \lambda)\theta_{t-1} - \eta_t \hat{u}_t$$

Under μP, the hidden-layer learning rate scales down as $\eta_t = \alpha P/M \propto 1/M$ so that the gradient term $\eta_t \hat{u}_t$ maintains spectral norm $\Theta(1)$. If you hold the decoupled weight-decay coefficient λ fixed (for example, $\lambda = 0.1$) across widths, then the per-step multiplicative decay rate $\eta_t \lambda = (\alpha P/M)\lambda$ shrinks by M/P! Wide models receive dramatically weaker weight decay than narrow proxy models, shifting the effective regularization strength and breaking optimal-LR transfer.

WORKED EXAMPLE

Worked Example: Diagnosing Decoupled Weight-Decay Mismatch in Transformer μP.

Consider transferring a μP Transformer from proxy width $P = 128$ to target width $M = 2048$ ($r = M/P = 16$), with MLP width $F = 4M$, base AdamW LR $\alpha = 2^{-6} = 0.015625$, and fixed decoupled weight decay $\lambda = 0.1$.

1. **Compute exact μP rules at $M = 2048$:**
   - Embedding W^E: init var $= 1$, $\eta_E = \alpha = 0.015625$.
   - Attention QKV / W^{AO} / W^{FI}: init var $= 1/2048 \approx 4.88 \times 10^{-4}$, $\eta = \alpha(128/2048) = 2^{-10} \approx 9.77 \times 10^{-4}$.
   - MLP output W^{FO}: init var $= 0.25/2048 \approx 1.22 \times 10^{-4}$, $\eta = 2^{-10} \approx 9.77 \times 10^{-4}$.
   - LM head W^U: init var $= 1/2048^2 \approx 2.38 \times 10^{-7}$, $\eta = 2^{-10} \approx 9.77 \times 10^{-4}$.
2. **Compare per-step weight decay fraction $\eta\lambda$ on hidden matrices:**
   - At proxy width $P = 128$: $\eta_{\text{proxy}}\lambda = 2^{-6} \times 0.1 \approx 1.56 \times 10^{-3}$ (0.156% weight shrinkage per step).
   - At target width $M = 2048$: $\eta_{\text{target}}\lambda = 2^{-10} \times 0.1 \approx 9.77 \times 10^{-5}$ (0.0098% weight shrinkage per step).
3. **Interpretation:** Keeping $\lambda = 0.1$ fixed cuts the per-step weight-decay rate on hidden matrices by $16\times$ at width 2048 compared to the width-128 proxy. To preserve identical per-step multiplicative decay $(1 - \eta\lambda)$, an independent (width-scaled) weight decay rule is required—which explains why Lingle saw best LR drift from 2^{-8} to 2^{-6} as soon as decoupled weight decay 0.1 was turned on.

COMMON MISCONCEPTION

You might think that because AdamW decouples weight decay λ from the adaptive gradient variance denominator, holding $\lambda = 0.1$ constant across widths automatically keeps regularization strength width-invariant under μP. That breaks because AdamW still multiplies λ by the layer learning rate η_t: since μP scales $\eta_t \propto 1/M$, holding λ constant weakens weight decay by a factor of M/P on hidden matrices and shifts the optimal learning rate.

Late-2026 μP Extensions: Muon, Depth, Horizon, and Sparse MoE

Classical μP targeted dense AdamW models with weak or zero weight decay and transferred primarily along the **width** axis. Late-2026 research closed the gap to real frontier pretraining—which combines **Muon**, **depth scaling**, **multi-trillion-token horizons**, **Multi-Head Latent Attention (MLA)**, and **sparse MoEs**—through three major extensions:

Method / Study	Key Extension
Qiu et al. (μP for Muon)	Pairs μP with $1/\text{width}$ **weight decay** for Muon—directly addressing the decoupled weight-decay transfer failure on matrix optimizers.
HyperP	Extends zero-shot hyperparameter transfer so **one learning rate transfers across width, depth, token count, and MoE granularity** simultaneously.
"Let's Scale Step by Step"	Adapts μP to an **MLA + Muon MoE** architecture, extrapolating the optimal learning rate out to $\sim 10\text{T}$ **tokens** for a 155B **total** / 17B**-active** model.

With these extensions, μP evolved from a theoretical dense-width recipe into a practical engineering toolkit covering width, depth, token horizon, MoE routing granularity, and orthogonalized matrix optimization.

Unified Recap: Scaling Language Models in Practice (Late 2026)

Bringing together all four parts of this lecture—schedules, hyperparameter laws, optimizers, and parametrization—each core scaling challenge now has a well-tested late-2026 practical solution:

Scaling Challenge	Late-2026 Practical Solution
$O(n^2)$ **Chinchilla sweep cost**	Use **WSD** ($\sim 10\%$ decay) or multi-step schedules to branch multiple token horizons from one shared stable trunk in $O(n)$ cost.
LR, batch & MoE sizing	Fit empirical laws (**Step Law**: $B \propto D^{0.571}$, $\eta \propto N^{-0.713} D^{0.307}$; IsoFLOPs; **Kimi K2 sparsity laws**), keeping WSD horizon-exponent caveats in mind.
Optimizer scale & stability	Tune baselines independently across scale and Chinchilla ratio; adopt **Muon** with **QK-Clip / gated residuals / GatedNorm** and owner-rank or compressed distributed Muon (**Dion2/3, MuonBP**).
Hyperparameter transfer	Use **μP** (controlling $\|W\|_*$ and $\|\Delta W\|_*$, using $\tau^{-1} = 1/D$, and dropping RMSNorm gains) plus 2026 extensions: $1/\text{width}$ **weight decay for Muon**, **HyperP**, and **MLA + Muon MoE μP**.

KEY IDEA

- In real Transformers, μP locks optimal learning rate across $5000\times$ parameter growth ($2\text{M} \to 10\text{B}$) and beats standard parametrization in final loss, provided you use $1/D$ attention logit scaling, drop trainable RMSNorm gains, and account for decoupled weight decay.
- Predictable late-2026 frontier scaling rests on three interlocking pillars: **cheap sweeps (WSD) + invariant dynamics (μP and its Muon/MoE extensions) + stable optimization (Muon with logit/residual gating)**.

✎ Check Your Understanding

1. **Yes/No with justification:** Under Lingle's Transformer μP specification, should you keep trainable RMSNorm gains and use standard $1/\sqrt{D}$ attention logit scaling?
   Answer: No to both. Trainable RMSNorm gains with $\Theta(1)$ LR break width-invariant transfer (and slightly hurt the largest μP models, so they can be dropped), and μP requires $\tau^{-1} = 1/D$ attention logit scaling rather than $1/\sqrt{D}$ to preserve transfer and quality during feature learning.
2. **What would happen if…** You scale a Transformer under AdamW μP from width $P = 256$ to $M = 2048$ while keeping decoupled weight decay fixed at $\lambda = 0.1$. By what factor does the per-step weight decay shrinkage $\eta_t\lambda$ on hidden matrices change, and how does this affect transfer?
   Answer: Because $\eta_t = \alpha P/M$ shrinks by $2048/256 = 8\times$, the per-step shrinkage $\eta_t\lambda$ drops by $8\times$ on hidden matrices, weakening regularization unevenly across widths and breaking optimal learning-rate transfer.