Lecture 9: Scaling Laws — Basics

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

Section 1: The Scaling Design Problem and Foundations of Data Scaling

Suppose your engineering team has 10,000 B200s allocated for one month to train a frontier open-source language model. Your distributed training stack is running smoothly and your pretraining corpus is assembled. The central engineering question still remains: which model should you train? That single decision decomposes into dozens of coupled choices: dense versus sparse Mixture of Experts (MoE), how many layers, attention heads, and active experts to configure, how to trade off parameter count N against training tokens D, and which optimizer, batch size, learning rate, and data mixture to select.

Historically, practitioners tuned these design choices and hyperparameters directly on the target model. At frontier scale, direct trial-and-error tuning is prohibitively expensive—a single failed run consumes millions of dollars of compute and weeks of calendar time. Simply copying architectural and training choices from previously published models also breaks down the moment your compute budget, available data pool, or serving traffic target differs from the published model's constraints.

> **DEFINITION**
> **Scaling Law:** A simple, predictive empirical or semi-theoretical mathematical rule—typically fitted to runs on small models spanning several orders of magnitude in compute—that extrapolates model loss, optimal architecture, hyperparameters, or resource allocation to a much larger target scale.

Rather than guessing at target scale, the scaling-law methodology fits predictive curves on small models and extrapolates forward. The entire lecture is organized around the three sequential questions scaling laws answer, which form a three-stage pipeline:

1. **Data Scaling** asks how much data to use and which data to include—covering dataset size versus test loss, why power laws arise, data mixtures and distribution shift, data repetition, and adaptive filtering.
2. **Model Engineering** builds on data scaling to ask which model design and hyperparameters to choose —covering architecture and optimizer selection, depth, width, and MoE sparsity, critical batch size, and learning-rate transfer from μP through CompleteP and HyperP.
3. **Compute Allocation** resolves the overarching budget trade-off of whether to invest compute in a bigger model or more tokens—covering joint (N, D) loss surfaces, the Kaplan versus Chinchilla debate, IsoFLOP curves and FLOP accounting, and inference-optimal overtraining.

From Classical Sample Complexity to Empirical Data Scaling

Before neural scaling laws existed, statistical learning theory studied how prediction error decreases as a function of sample size m or n for decades. Consider two foundational classical results.

First, for a finite hypothesis class H containing k candidate functions, a standard uniform-convergence generalization bound guarantees that, with probability at least $1-\delta$ over a training set of m independent samples, the empirical risk minimizer $\hat{h}$ satisfies:

$$\epsilon(\hat{h}) \leq \left(\min_{h \in H} \epsilon(h)\right) + 2\sqrt{\frac{1}{m}\log\frac{2k}{\delta}}$$

Breaking this inequality apart term by term clarifies what it predicts:

- On the left-hand side, $\epsilon(\hat{h})$ is the true expected error (population risk) of the learned hypothesis $\hat{h}$.
- The first term on the right-hand side, $\min_{h \in H} \epsilon(h)$, is the best achievable error within the hypothesis class H (the approximation error floor).
- Inside the square root, $\log(2k/\delta)$ measures the complexity of searching over k candidate models at confidence level $1-\delta$, while $1/m$ scales that penalty inversely with the number of training examples m.
- Taking the square root yields an excess-error decay proportional to $m^{-1/2}$.

As a whole, this bound states that the test error of the chosen model exceeds the best model in the class by at most an $O(m^{-1/2})$ statistical estimation penalty.

Second, in classical nonparametric statistics—where the target function is not restricted to a finite parametric family—consider smooth density estimation (Hall, 1989). For a class $\mathcal{P}(\beta, L)$ of probability densities with smoothness order β and Lipschitz/derivative bound L, defining the convergence rate $\psi_n = n^{-\frac{\beta}{2\beta+1}}$, the worst-case mean squared error of a nonparametric estimator $\hat{p}_n$ at a point x_0 satisfies:

$$\sup_{p \in \mathcal{P}(\beta, L)} \mathbb{E}_p\left[(\hat{p}_n(x_0) - p(x_0))^2\right] \leq C\psi_n^2$$

Here, the innermost expression $(\hat{p}_n(x_0) - p(x_0))^2$ is the squared estimation error at x_0, $\mathbb{E}_p[\cdot]$ takes its expectation over n samples drawn from density p, and $\sup_{p \in \mathcal{P}(\beta,L)}$ takes the worst-case density in the smoothness class. Because $\psi_n^2 = n^{-\frac{2\beta}{2\beta+1}}$, smoother functions (larger β) approach the parametric n^{-1} rate, while rougher functions decay more slowly.

> **COMMON MISCONCEPTION**
> You might think that plugging a neural network's parameter count or hypothesis capacity into these classical learning-theory bounds gives a usable scaling prediction for language model pretraining. That breaks because classical sample complexity yields **worst-case upper bounds** over adversarial distributions, not the exact, realized loss curves observed on real datasets and neural architectures. An upper bound that is off by orders of magnitude—or predicts the wrong exponent—cannot tell an engineer whether to train a 70B model or a 280B model.

Because worst-case bounds could not predict realized error, researchers turned to empirical curve fitting decades before large language models appeared:

Paper	Empirical Finding	Functional Form / Takeaway
Cortes et al. (1993)	Earliest empirical scaling-law study: fit learning curves on small data subsets to predict full-dataset error	$\mathcal{E}_{\text{test}} = a + b/l^{\alpha}$, $\mathcal{E}_{\text{train}} = a - c/l^{\beta}$; avoids wasting compute training poor classifiers on full data
Banko & Brill (2001)	On confusion-set disambiguation, accuracy grew log-linearly from 0.1M to $1{,}000\text{M}$ words across Winnow, Perceptron, Naive Bayes, and Memory-Based learners	None of the learners asymptoted; corpus growth outweighed algorithm choice
Kolachina et al. (2012)	Systematically compared parametric curve families for downstream machine translation BLEU score versus training set size	Tested Exp_3, Exp_4, ExpP_3, Pow_3 $(c - ax^{-\alpha})$, Pow_4, and ILog_2; power laws fit reliably

Across all three early studies, empirical power laws linked data volume directly to downstream accuracy long before modern Transformers.

Three Scaling Regimes in Deep Learning (Hestness et al., 2017)

Hestness et al. (2017) conducted the earliest large-scale neural scaling study spanning machine translation, language modeling, and speech recognition. Visualizing generalization error against training dataset size on **log-log axes** reveals a characteristic S-shaped composite frontier partitioned into three distinct regimes:

1. **Small Data Region (left third of the log-log plot):** The learning curve begins flat along a horizontal upper asymptote labeled **Best-Guess Error**. With too few training samples, the model cannot extract generalizable patterns and performs no better than random or frequency-based guessing.
2. **Power-Law Region (middle region):** Once the dataset crosses a minimum threshold, the curve bends downward into a clean straight line with negative slope $\beta_g < 0$. Each multiplicative increase in data yields a consistent multiplicative reduction in excess error.
3. **Irreducible Error Region (right third of the plot):** At very large dataset sizes, the curve bends back toward a horizontal lower asymptote labeled **Irreducible Error**, corresponding to the intrinsic uncertainty (Bayes entropy) of the task itself.

Mathematically, Hestness et al. summarize this composite frontier with a shifted power law:

$$\varepsilon(m) = \alpha\, m^{\beta_g} + \gamma$$

Breaking apart every symbol:

- m is the training dataset size (in samples, words, or tokens).
- $\varepsilon(m)$ is the generalization error achieved at dataset size m when model capacity is appropriately scaled.
- $\gamma \geq 0$ is the **irreducible error floor** (Bayes entropy): even an infinitely large model trained on infinite data cannot eliminate inherent ambiguity in predicting the next word or noisy label.

- $\alpha > 0$ is a multiplicative scale factor (which determines the vertical offset of the curve in log space).
- $\beta_g < 0$ is the **scaling exponent**, giving the constant linear slope of the excess error $\varepsilon(m) - \gamma$ on log-log axes.

Notice an essential distinction between individual models and the composite frontier: any single fixed-size model will eventually flatten out once its parameter capacity saturates on large data. The straight line in the power-law region is the **lower envelope** formed by choosing the best-fit model size at each dataset scale.

> **DEFINITION**
> **Power-Law Region:** The intermediate data or compute regime in which excess generalization error $\varepsilon(m) - \gamma$ decays polynomially as αm^{β_g} ($\beta_g < 0$), appearing as a straight line with slope β_g on log-log axes between the small-data plateau and the irreducible error floor.

From these empirical curves, Hestness et al. (2017) drew three prescient engineering insights that shaped modern large-scale training:

1. **"Emergence" and accuracy cliffs:** Models with poor initialization or suboptimal optimizer settings can remain trapped on the flat best-guess plateau until training data crosses a critical threshold into the power-law region, making performance appear to jump suddenly. Consequently, small-scale experiments fail to predict large-scale behavior if the proxy dataset is too small to exit the small-data plateau.
2. **Scaling by compute:** Because both learning curves and optimal model sizes follow predictable power laws, engineers can project the total compute required to reach a target accuracy before committing to month-long training runs.
3. **Speed equals accuracy:** Techniques such as low-precision arithmetic, quantization, or sparsity can trade away local accuracy (for instance, up to 20% degradation at fixed model size and step count) in exchange for substantially higher compute throughput. When that higher throughput enables training larger models on more data within the same wall-clock budget, movement down the scaling curve recovers—and exceeds—the accuracy initially sacrificed.

Scale-Free Power Laws in Autoregressive Transformers (Kaplan et al., 2020)

Kaplan et al. (2020) extended this picture to autoregressive Transformer language models and demonstrated clean log-log power laws independently across three axes—compute $C_{\min}$, dataset size D, and non-embedding parameter count N—whenever a run is not bottlenecked by the other two resources.

A schematic three-panel log-log figure summarizes their results (with Test Loss on the vertical log axis in all three panels):

- **Panel 1 — Compute (PF-days, non-embedding):** Individual training runs for models of different sizes appear as curved trajectories that drop steeply and then peel upward and flatten as each fixed

model saturates. A dashed straight line forms the lower **envelope** tangent to these runs, obeying:

$$L(C_{\min}) = \left(\frac{C_{\min}}{2.3 \times 10^8}\right)^{-0.050}$$

- **Panel 2 — Dataset Size (tokens D):** When model capacity is large enough not to bottleneck learning, test loss lies tightly on a straight line across orders of magnitude in tokens:

$$L(D) = \left(\frac{D}{5.4 \times 10^{13}}\right)^{-0.095}$$

- **Panel 3 — Parameters (non-embedding N):** When data is abundant enough not to bottleneck learning, test loss lies tightly on a straight line across orders of magnitude in parameters:

$$L(N) = \left(\frac{N}{8.8 \times 10^{13}}\right)^{-0.076}$$

Notice the common structure $L(X) = (X/X_c)^{-\alpha_X}$. Dividing X by a characteristic scale constant X_c makes the ratio dimensionless and absorbs the multiplicative constant $\alpha = X_c^{\alpha_X}$ cleanly into the denominator. Because the slope $-\alpha_X$ is constant across orders of magnitude, the relationship is **scale-free**: multiplying X by any factor k multiplies the predicted loss by the same factor $k^{-\alpha_X}$, regardless of whether X starts at 10^7 or 10^{11}. Moreover, predictable scaling curves are not restricted to cross-entropy loss; smooth sigmoid accuracy curves as a function of $\log(\text{compute})$ also appear on downstream tasks such as Word Unscramble and Persian QA, as well as on aggregate capability indices.

WORKED EXAMPLE

Worked Example: Extrapolating Data Scaling with Kaplan's Power Law

Suppose an unbottlenecked autoregressive Transformer obeys Kaplan et al.'s data scaling law $L(D) = (D/5.4 \times 10^{13})^{-0.095}$. Compute the predicted test loss at $D_1 = 10^9$ tokens, $D_2 = 10^{10}$ tokens, and $D_3 = 10^{11}$ tokens, and quantify the fractional loss reduction per $10\times$ increase in data.

1. At $D_1 = 10^9$ tokens:

$$\frac{D_1}{5.4 \times 10^{13}} = \frac{10^9}{5.4 \times 10^{13}} \approx 1.8519 \times 10^{-5}$$

$$L(10^9) = (1.8519 \times 10^{-5})^{-0.095} = \exp\big(-0.095 \ln(1.8519 \times 10^{-5})\big) \approx \exp(1.0352) \approx$$

2. At $D_2 = 10^{10}$ tokens ($10\times$ more data):

$$L(10^{10}) = \left(\frac{10^{10}}{5.4 \times 10^{13}}\right)^{-0.095} = 10^{-0.095} \cdot L(10^9) \approx 0.8035 \times 2.816 \approx 2.262$$

3. At $D_3 = 10^{11}$ tokens (another $10\times$ more data):

$$L(10^{11}) = 0.8035 \times 2.262 \approx 1.818$$

Interpretation: Every $10\times$ increase in training tokens reduces test loss by the exact same multiplicative factor of $10^{-0.095} \approx 0.8035$ (a 19.65% drop), illustrating scale-free decay. However, note the danger of blind extrapolation: because this pure power law omits an explicit irreducible floor γ, pushing $D \to \infty$ would falsely drive $L \to 0$. We'll need explicit irreducible floors when we analyze joint (N, D) loss surfaces in Section 6.

KEY IDEA

1. Scaling laws replace prohibitively expensive trial-and-error tuning on large models by fitting predictive power laws on small models—provided those small runs exit the flat small-data plateau and remain above the irreducible Bayes entropy floor.
2. When unbottlenecked, autoregressive Transformer test loss exhibits scale-free power-law decay across compute, dataset size, and parameter count, and throughput gains from lower precision or sparsity can more than pay back local accuracy losses by moving further down the scaling curve.

We now have overwhelming empirical evidence that neural test loss follows straight lines on log-log axes across many orders of magnitude. Yet this observation immediately raises two fundamental questions: why should learning curves take the form of power laws in the first place, and why are the observed neural exponents (-0.095 for language modeling, -0.13 for machine translation, -0.30 for speech) so dramatically flatter than classical $1/n$ rates?

✎ Check Your Understanding

1. **Yes/No with justification:** If a candidate model architecture shows flat, near-random validation loss across two small dataset sizes, does that prove the architecture cannot scale well on a much larger pretraining corpus?
   Answer: No. According to Hestness et al.'s three scaling regimes, both small datasets may still lie inside the flat small-data (best-guess) region; small-scale tests only predict large-scale scaling once the training set crosses the threshold into the power-law region.
2. **Quick computation:** Under the shifted power law $\varepsilon(m) = \alpha m^{\beta_g} + \gamma$ with $\beta_g = -0.10$, by what factor does the *excess* error $\varepsilon(m) - \gamma$ shrink when you increase dataset size m by a factor of $1{,}000$?
   Answer: It shrinks by a factor of $1000^{-0.10} = (10^3)^{-0.10} = 10^{-0.30} \approx 0.501$, meaning a $1{,}000\times$ increase in data cuts excess error roughly in half.

Section 2: Why Power Laws Arise and the Mystery of Shallow Exponents

Expecting test error to decrease monotonically as we add more training data is intuitive. Expecting that decrease to trace a **straight line on a log-log plot** is much less obvious. To see why power laws arise naturally in statistical estimation—and why deep learning exhibits such shallow slopes—we start with the simplest possible estimation problem and progressively increase the flexibility of the function class.

Toy Example: Parametric Mean Estimation

Suppose we observe n independent, identically distributed scalar samples drawn from a Gaussian distribution with unknown mean μ and known variance σ^2:

$$x_1, \dots, x_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mu, \sigma^2)$$

We estimate the unknown mean using the sample average $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Because $\hat{\mu}$ is unbiased ($\mathbb{E}[\hat{\mu}] = \mu$) and the samples are independent, recall that the variance of a sum of independent random variables equals the sum of their variances, scaling quadratically by $1/n^2$:

$$\mathbb{E}\big[(\hat{\mu} - \mu)^2\big] = \operatorname{Var}\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n^2}\sum_{i=1}^{n} \sigma^2 = \frac{\sigma^2}{n}$$

Taking the logarithm of both sides transforms this polynomial decay into an exact linear equation in $\log n$:

$$\log(\text{Error}) = -\log n + 2\log\sigma$$

More generally, **any polynomial convergence rate** of the form $\text{Error} \propto n^{-\alpha}$ yields $\log(\text{Error}) = -\alpha \log n + C$, which is a straight line with slope $-\alpha$ in log-log coordinates. Power-law scaling is therefore not a mysterious artifact unique to neural networks; it is the geometric signature of polynomial statistical convergence plotted in logarithmic coordinates.

The Exponent Mystery: Why Are Neural Slopes So Shallow?

That simple derivation immediately exposes a quantitative puzzle. Classical parametric estimators—such as Gaussian mean estimation and fixed-dimensional linear regression—converge at the $1/n$ rate, producing a steep line of slope -1.0 in log-log space. Plotting classical $1/n$ decay alongside observed neural scaling curves on the same log-log axes (with $\log(\text{Dataset Size})$ horizontally and $\log(\text{Error} \,/\, \text{Loss})$ vertically) reveals a striking contrast:

- **Classical parametric estimation ($1/n$):** plunges steeply downward with slope -1.0.
- **Speech recognition (Hestness et al., 2017):** decays as $m^{-0.30}$, more than three times shallower than $1/n$.
- **Machine translation (Hestness et al., 2017):** decays as $m^{-0.13}$, nearly eight times shallower than $1/n$.

- **Language modeling (Kaplan et al., 2020):** decays as $D^{-0.095}$, more than ten times shallower than $1/n$.

Why does neural network error decay an order of magnitude more slowly than $1/n$?

> **COMMON MISCONCEPTION**
> You might think a slope of -0.095 means neural networks are statistically broken or wasting 90% of every gradient update compared to classical estimators. That breaks because $1/n$ convergence only holds when the true target function is known to lie inside a small, fixed-dimensional parametric family (such as a single scalar mean μ). When an estimator must learn an arbitrary smooth function over a high-dimensional domain—as a neural network does—information theory imposes a strictly slower, dimension-dependent convergence rate.

Nonparametric Learning and Dimension Dependence

Neural networks behave less like single-parameter estimators and more like **nonparametric** estimators capable of approximating arbitrary smooth functions. Consider a toy two-dimensional nonparametric regression problem: inputs $x_1, \dots, x_n$ are drawn uniformly on the 2D unit square $[0,1]^2$, and noisy observations satisfy $y_i = f(x_i) + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0,1)$ for some smooth target function f.

A simple nonparametric estimator partitions the unit square $[0,1]^2$ into a uniform grid of square cells and predicts the sample average of y_i inside each cell. To visualize this construction, picture the unit square $[0,1]^2$ subdivided into a grid of small square cells of side length $h = n^{-1/4}$, with one highlighted cell containing several scattered sample points:

1. **Number of cells:** Each cell has area $h^2 = (n^{-1/4})^2 = n^{-1/2}$. Dividing the total area 1 by $n^{-1/2}$ yields $\sqrt{n}$ total boxes.
2. **Samples per cell:** Because the n points are uniform over the square, each cell receives approximately $n/\sqrt{n} = \sqrt{n}$ samples.
3. **Variance error inside a cell:** Averaging $\sqrt{n}$ noisy labels with unit variance yields a squared variance error of approximately $1/\sqrt{n}$.
4. **Smoothness bias inside a cell:** Because the estimator approximates $f(x)$ as constant across a cell of width $h = n^{-1/4}$, for a Lipschitz-smooth function the approximation bias is $O(h)$, so the squared bias is $O(h^2) = O((n^{-1/4})^2) = O(1/\sqrt{n})$.

Combining both error sources gives:

$$\text{Error} \approx \frac{1}{\sqrt{n}} + (\text{smoothness bias terms}) \propto n^{-1/2}$$

Here we chose $h = n^{-1/4}$ specifically to balance the variance term $(nh^2)^{-1}$ against the squared smoothness bias h^2: making boxes larger lowers variance but increases bias, while making boxes smaller lowers bias but leaves too few samples per box. In d dimensions, a grid of boxes of side length h contains

h^{-d} total boxes, so the number of boxes grows exponentially with d. To keep enough samples per box to control variance while still shrinking box width, the convergence rate degrades inversely with dimension d:

$$\text{Error} \propto n^{-1/d} \implies y = -\frac{1}{d}x + C$$

where $y = \log(\text{Error})$ and $x = \log n$. As dimension d grows, the log-log slope $-1/d$ becomes progressively shallower.

WORKED EXAMPLE

Worked Example: Balancing Grid Resolution in 2D Nonparametric Estimation

Suppose we collect $n = 10{,}000$ noisy observations uniformly on $[0, 1]^2$ and apply the cell-averaging estimator with cell side length $h = n^{-1/4}$.

1. **Cell side length:** $h = (10{,}000)^{-1/4} = (10^4)^{-1/4} = 10^{-1} = 0.1$.
2. **Total number of cells:** The 2D unit square is split into $1/h = 10$ intervals per axis, yielding $10 \times 10 = 100 = \sqrt{10{,}000}$ cells.
3. **Expected samples per cell:** $10{,}000/100 = 100$ samples per cell.
4. **Variance error:** Averaging 100 samples with noise variance $\sigma^2 = 1$ yields variance $1/100 = 0.01$.
5. **Squared bias:** With Lipschitz constant $L_f = 1$, f varies by at most $O(h) = 0.1$ across a cell, giving squared bias $O(h^2) = 0.01$.
6. **What if we kept 1D parametric intuition and ignored dimension?** Notice that if we instead tried to use $n = 10{,}000$ samples to shrink variance by $100\times$ (requiring $10{,}000$ samples *per cell* at the same resolution), we would need $100 \times 10{,}000 = 1{,}000{,}000$ total samples—and even then the 0.01 bias would stop decreasing unless we shrank the boxes further. Because resolution must shrink along *both* axes simultaneously, quadrupling the sample count only halves the error ($n^{-1/2}$) rather than quartering it (n^{-1}).

Interpretation: Even in just $d = 2$ dimensions, nonparametric smoothness estimation already flattens the log-log slope from -1.0 to -0.5. In higher dimensions, filling the input space forces exponentially more data for every factor-of-two reduction in error.

Intrinsic Dimensionality Theory (Bahri et al., 2021)

Real inputs such as images and text live in ambient spaces with thousands or millions of coordinates, yet real data does not fill that ambient space uniformly; it concentrates on a much lower-dimensional structure called the **data manifold**.

DEFINITION

Intrinsic Dimensionality (d): The effective number of independent degrees of freedom needed to describe the low-dimensional manifold on which a real dataset lies, typically far smaller than the raw ambient input dimension (such as pixel count or vocabulary-by-context dimension).

Bahri et al. (2021) formalized the connection between nonparametric rates and neural scaling laws into the **intrinsic dimensionality hypothesis**:

- Neural scaling laws emerge from polynomial learning rates $L \propto 1/n^{\alpha_D}$.
- In the variance-limited and resolution-limited regimes analyzed by Bahri et al., the data scaling exponent α_D is inversely proportional to the intrinsic dimensionality d of the data manifold—specifically predicting $4/\alpha_D \approx d$ in their teacher–student kernel/neural analysis.

An empirical scatter plot from Bahri et al. (2021) tests this prediction directly by placing **Estimated Intrinsic Dimension** d on the horizontal axis and $4/\alpha_D$ on the vertical axis, with two diagonal dashed reference lines for $4/\alpha_D = d$ (steeper upper line) and $2/\alpha_D$ (lower line):

- **Synthetic teacher–student tasks** (plotted as a sequence of magenta points at lower d) fall cleanly and linearly right along the $4/\alpha_D \approx d$ dashed line.
- **Real image benchmarks** (MNIST, SVHN, FashionMNIST, CIFAR-10, and CIFAR-100) follow the general upward trend—higher estimated intrinsic dimension corresponds to smaller α_D and thus larger $4/\alpha_D$—with MNIST near the line and more complex datasets like CIFAR-100 rising toward the upper right, though with noticeable scatter around the theoretical line.

WORKED EXAMPLE

Worked Example: Estimating Implied Intrinsic Dimension from Language Modeling

Using Bahri et al.'s approximate relationship $4/\alpha_D \approx d$, estimate the implied intrinsic dimension corresponding to Kaplan et al.'s language modeling data exponent $\alpha_D = 0.095$, and compare it to Hestness et al.'s speech exponent $\alpha_D = 0.30$.

1. For language modeling ($\alpha_D = 0.095$):

$$d_{\text{LM}} \approx \frac{4}{0.095} \approx 42.1$$

2. For speech recognition ($\alpha_D = 0.30$):

$$d_{\text{speech}} \approx \frac{4}{0.30} \approx 13.3$$

Interpretation: Language modeling exhibits a shallower scaling slope than speech because natural language has a higher effective intrinsic dimensionality than acoustic phoneme sequences—far lower than the raw ambient dimension of token sequences, yet still high enough to make the scaling slope around -0.1.

Treat this theory with appropriate scientific caution: estimators of intrinsic dimension on high-dimensional real datasets are notoriously noisy and sensitive to representation choice. Intrinsic dimensionality theory provides a strong conceptual explanation for why power laws occur and why neural exponents are shallow, rather than an airtight formula that predicts exact exponents from raw data alone.

KEY IDEA

1. Any polynomial convergence rate $\text{Error} \propto n^{-\alpha}$ produces a straight line of slope $-\alpha$ in log-log coordinates; classical parametric models converge at $1/n$ (slope -1), whereas flexible nonparametric estimators pay a dimension-dependent penalty $\text{Error} \propto n^{-1/d}$.
2. Bahri et al.'s intrinsic dimensionality hypothesis links the shallow empirical exponents of deep learning ($\alpha_D \approx 0.095$–0.30) inversely to the intrinsic dimension d of the data manifold, aligning cleanly on synthetic tasks and approximately on real benchmarks.

Up to this point, we have treated a training corpus as a uniform pool characterized solely by a single scalar size n, m, or D. In practice, pretraining corpora combine heterogeneous domains (web text, code, books, scientific papers), exhaust their finite pool of unique tokens and must be repeated across epochs, and pass through aggressive quality filters. We next examine how data composition, repetition, and compute-aware filtering reshape scaling laws—and where naive extrapolation breaks down.

✎ Check Your Understanding

1. **Fill-in-the-blank:** In a log-log plot of generalization error versus dataset size n, a polynomial convergence rate $\text{Error} \propto n^{-\alpha}$ appears as a straight line with slope _____, which becomes progressively shallower as the _____ of the data manifold increases.
   Answer: $-\alpha$; intrinsic dimensionality d.
2. **What would happen if…:** In the 2D unit-square nonparametric estimator, what would happen to the bias and variance terms if you held the box side length fixed at $h = 0.1$ while increasing n from $10{,}000$ to $10{,}000{,}000$?
   Answer: The variance term inside each cell would drop by $1{,}000\times$ (since each cell gets $1{,}000\times$ more samples), but the smoothness bias term $O(h^2)$ would remain frozen at its constant value, causing total error to plateau rather than continuing down the $n^{-1/2}$ power law.

Section 3: Data Composition, Repetition, Filtering, and Where Scaling Laws Break

Treating every token in a pretraining corpus as identical hides three decisive engineering realities: corpora are mixtures of multiple domains, high-quality unique tokens are finite, and quality filters discard tokens in exchange for higher average quality. Scaling laws extend to all three phenomena, and studying them reveals both permanent efficiency multipliers and hard failure modes of naive extrapolation.

Data Composition and Distribution Shift (Hashimoto, 2021)

Suppose your training corpus mixes data from multiple sources with domain proportions represented by a mixture vector q. How does changing q alter the log-log scaling curve of excess error versus training size?

Hashimoto (2021) established a clean structural result illustrated by two companion plots:

- **Left panel (Parallel Shifted Slopes):** Plotting $\log(\text{Excess error})$ on the vertical axis against $\log(\text{Training size})$ on the horizontal axis for three different mixture proportions ($q = 0.00$, $q = 0.22$, and $q = 0.56$) produces three **parallel straight lines**. Moving from $q = 0.00$ (top line) down to $q = 0.22$ (middle line) and $q = 0.56$ (lowest line) shifts the entire curve vertically downward without altering its slope.
- **Right panel (Error Intercept $\log C(q)$):** Plotting the vertical intercept $\log C(q)$ against source proportion q reveals a sharp **U-shaped curve**. At the two extremes ($q \to 0$ or $q \to 1$, corresponding to pure single-source data), $\log C(q)$ spikes upward, indicating poor performance. In the interior, the curve reaches a clear minimum labeled **best mix**, where diverse mixtures minimize the intercept at every dataset scale.

Because composition shifts the log-space **offset** $\log C(q)$ rather than the **slope** $-\alpha$, improving your data mixture acts as a **permanent multiplicative data efficiency gain** across all scales.

> **WORKED EXAMPLE**
>
> **Worked Example: Converting an Offset Improvement into a Data Multiplier**
>
> Suppose excess error obeys $\mathcal{E}(D) = C(q)\, D^{-0.10}$, where a baseline mixture q_1 has $C(q_1) = 1.00$ and a better-balanced mixture q_2 reduces the offset by 10% to $C(q_2) = 0.90$ without changing the slope -0.10. How much more data D_1 does the baseline mixture need to match the loss achieved by mixture q_2 at dataset size D_2?
>
> Set the two errors equal:
>
> $$C(q_1)\, D_1^{-0.10} = C(q_2)\, D_2^{-0.10}$$
>
> $$1.00 \cdot D_1^{-0.10} = 0.90 \cdot D_2^{-0.10}$$

Solve for the ratio D_1/D_2:

$$\left(\frac{D_1}{D_2}\right)^{-0.10} = 0.90 \implies \frac{D_1}{D_2} = 0.90^{-1/0.10} = 0.90^{-10} \approx 2.87$$

Interpretation: Because the scaling exponent is shallow ($\alpha = 0.10$), even a seemingly modest 10% vertical shift in the intercept $\log C(q)$ translates into a $2.87\times$ **permanent data multiplier**. Achieving the same loss with the worse mixture requires nearly triple the tokens at every scale.

Data Mixture Selection in Practice

Knowing that a better mixture shifts $\log C(q)$ downward does not automatically tell you how to pick the optimal mixture before a frontier training run. Two competing practical methodologies have emerged:

- **Approach A: Parametric Mixing Laws (Ye et al., *Data Mixing Laws*):** Uses a three-step extrapolation pipeline: (1) extrapolate loss over **training steps** using small models on seen mixtures, (2) extrapolate over **model sizes** on those seen mixtures, and (3) fit a parametric **data mixing law** across domain weights to predict the loss of *unseen* mixtures at target scale.
- **Approach B: Empirical Small-Scale Evaluation (Magnusson et al., *DataDecide*):** Evaluated 25 pretraining datasets across model scales from 4M to 1B parameters and tested whether simple small-scale rankings predict large-scale rankings. Simply pretraining at **150M parameters** predicts pairwise dataset rankings at **1B parameters with roughly 80% accuracy**, and direct small-scale ranking frequently beats more complex multi-scale parametric fits.

While joint parametric mixing laws are conceptually appealing, direct empirical evaluation at a moderate small scale (such as 150M parameters) is a remarkably strong baseline that is difficult to outperform.

Data-Constrained Scaling Under Repetition (Muennighoff et al.)

Classical scaling laws assume every token seen during training is freshly drawn from an infinite corpus. Once a training budget demands more tokens D than the number of unique tokens U_D available in your corpus, you must repeat data across multiple epochs.

Muennighoff et al. (*Scaling Data-Constrained Language Models*) modeled the diminishing value of repeated tokens. Plotting **Test Loss** against **Tokens / Epochs (log scale)** compares a dashed curve for **fresh data** against a solid curve for **repeated data**, separated by vertical markers into three regimes:

1. **Up to ≈ 4 epochs:** The solid repeated-data curve tracks right on top of the dashed fresh-data curve; repeating tokens up to roughly 4 epochs is almost as effective as training on brand-new data.
2. **Between 4 and 40 epochs:** The solid curve peels upward away from the fresh-data curve, showing steadily diminishing returns per additional repeated token.
3. **Beyond ≈ 40 epochs:** The repeated-data curve flattens almost completely; additional repetitions contribute essentially no further loss reduction.

To capture this exponential decay in the value of repeated epochs, Muennighoff et al. replace raw token count D with an **effective token count** D':

$$D' = U_D + U_D R_D^* \left(1 - e^{-R_D / R_D^*}\right)$$

Decomposing this formula from the inside out:

- U_D is the number of **unique tokens** in the dataset (the first pass, or epoch 1, contributes its full value U_D).
- R_D is the number of **repetitions** (additional epochs beyond the first pass, so total tokens processed are $D = U_D(1 + R_D)$).
- R_D^* is the characteristic **decay constant** (half-life scale of repetition utility, roughly 15 epochs in Muennighoff et al.'s fits).
- Inside the parentheses, the innermost term e^{-R_D / R_D^*} decays exponentially from 1 toward 0 as repetitions R_D grow. Therefore, $1 - e^{-R_D / R_D^*}$ grows smoothly from 0 toward 1:
  - When $R_D \ll R_D^*$, using the Taylor approximation $1 - e^{-x} \approx x$, we get $U_D R_D^* (R_D / R_D^*) = U_D R_D$, so $D' \approx U_D(1 + R_D) = D$—repeated tokens behave like fresh data.
  - As $R_D \to \infty$, $e^{-R_D / R_D^*} \to 0$, so D' saturates at the hard ceiling $U_D(1 + R_D^*)$.

As a single conceptual unit, the equation maps raw tokens seen across repeated epochs onto an equivalent number of fresh tokens that saturates smoothly once repetitions vastly exceed R_D^*.

DEFINITION

Effective Token Count (D'): The equivalent number of fresh, unique tokens that yields the same loss reduction as training on U_D unique tokens repeated for R_D additional epochs, accounting for the exponential decay in utility of repeated data.

Accounting for repetition directly changes how you allocate compute between parameters N and tokens D. At a fixed compute budget of 10^{22} FLOPs with a fixed unique corpus (approximately 25B unique tokens):

- **Naive allocation (ignoring repetition decay):** allocates 8.67B parameters trained for 178B tokens (7.1 epochs), reaching test loss 2.376.
- **Data-constrained allocation (accounting for D'):** shifts optimal model size *down* and tokens *up* to 6.34B **parameters** trained for 242B **tokens (9.7 epochs)**, achieving a better test loss of 2.359.

WORKED EXAMPLE

Worked Example: Computing Effective Tokens Under Repetition

Suppose your unique corpus contains $U_D = 25\text{B}$ tokens and the repetition decay constant is $R_D^* = 15$. Compare the raw tokens processed $D = U_D(1 + R_D)$, effective tokens D', and token efficiency D'/D at 1, 4, 10, 40, and 100 total epochs ($R_D = \text{epochs} - 1$).

1. **1 epoch ($R_D = 0$, $D = 25.0\text{B}$):**

$$D' = 25 + 25(15)(1 - e^{0}) = 25.00\text{B} \quad (\text{efficiency} = 100.0\%)$$

2. **4 epochs ($R_D = 3$, $D = 100.0\text{B}$):**

$$D' = 25 + 375(1 - e^{-3/15}) = 25 + 375(1 - 0.8187) \approx 92.98\text{B} \quad (\text{efficiency} = 93.0\%$$

3. **10 epochs ($R_D = 9$, $D = 250.0\text{B}$):**

$$D' = 25 + 375(1 - e^{-9/15}) = 25 + 375(1 - 0.5488) \approx 194.20\text{B} \quad (\text{efficiency} = 77.7\%$$

4. **40 epochs ($R_D = 39$, $D = 1{,}000.0\text{B}$):**

$$D' = 25 + 375(1 - e^{-39/15}) = 25 + 375(1 - 0.0743) \approx 372.15\text{B} \quad (\text{efficiency} = 37.2$$

5. **100 epochs ($R_D = 99$, $D = 2{,}500.0\text{B}$):**

$$D' = 25 + 375(1 - e^{-99/15}) = 25 + 375(1 - 0.00136) \approx 399.49\text{B} \quad (\text{efficiency} = 16.$$

Interpretation: Through 4 epochs, 93% of repeated tokens act like fresh data. Going from 40 epochs to 100 epochs burns $1{,}500\text{B}$ extra tokens of compute—more than doubling total training FLOPs—yet buys only 27.3B effective tokens as D' approaches its ceiling of $25(1 + 15) = 400\text{B}$.

When Scaling Laws Break: Unbounded Compute (Kim, Kotha, Liang & Hashimoto)

Even the saturating repetition formula above assumes that over-repeating data merely stops helping. Kim, Kotha, Liang & Hashimoto (*Pre-training under infinite compute*) pushed repetition and parameter scaling to extremes on fixed seed corpora and uncovered two fundamental failures of blind extrapolation, shown in a pair of empirical plots:

- **Left panel (Over-repeating Epochs):** Plotting Validation Loss against Epochs from $1 \to 128$ reveals a sharp **U-shaped curve**, not a flat plateau. Validation loss drops cleanly from 1 epoch down to a minimum around ~ 8 epochs, then **climbs back up sharply** through 128 epochs due to severe overfitting. Similarly, scaling model parameters from $150\text{M} \to 1.4\text{B}$ on a fixed seed dataset quickly flattens out.
- **Right panel (Shifting the Asymptote):** Plotting validation loss against seed dataset size ($209\text{M} \to 1.67\text{B}$ tokens) across three distinct training recipes shows three stacked curves, each sitting cleanly below the previous one:
  - **Standard training:** $L(D) = 1.30/D^{0.23} + 1.89$
  - **Regularized training:** $L(D) = 1.03/D^{0.23} + 1.96$
  - **Ensemble of models:** $L(D) = 0.88/D^{0.24} + 1.90$

COMMON MISCONCEPTION

You might think that a fitted data-constrained scaling law represents a universal, physics-like ceiling on how well any model can perform given a fixed dataset of U_D tokens. That breaks because scaling laws are **recipe-specific lower bounds**, not fundamental ceilings: changing the training algorithm via stronger regularization or ensembling lowers the excess-loss coefficient dramatically (from 1.30 down to 1.03 and 0.88) at the exact same seed token budget.

Compute-Aware Data Filtering (Goyal et al.)

Repetition decay also overturns the naive assumption that stricter data quality filtering is always better. Real web corpora consist of heterogeneous quality pools—for instance, quality buckets ranked $E > D > C > A > B > F$, where bucket E is the highest quality but smallest pool, and adding lower buckets progressively expands the unique token pool at the cost of lower average quality.

Goyal et al. (*Scaling Laws for Data Filtering*) plot **Estimated Error** against **Total Training Samples Seen (log scale)** for progressively broader subsets—E only, $E + D$, $E + D + C$, and $E + D + C + A$—partitioned by vertical dividers into three compute regimes:

1. **Small Compute (Aggressive Filtering — E only):** At small sample budgets, the top-bucket E-only curve starts with the lowest error because the model never exhausts bucket E and sees exclusively top-quality examples.
2. **Medium Compute (Mild Filtering — $E + D$):** As total samples seen increase, the E-only pool starts repeating heavily and its curve flattens early. The $E + D$ curve crosses below E-only and becomes optimal.
3. **Large Compute (Broad Filtering — $E + D + C + A$):** At very large sample budgets, even $E + D$ saturates from repetition. Relaxing the filter to include $E + D + C + A$ crosses below all narrower pools—fresh medium-quality data beats stale repeats of top-quality data.

DEFINITION

Quality–Quantity Tradeoff (QQT): The compute-dependent trade-off in data curation between filtering aggressively for high average token quality (optimal at low compute before repetition sets in) and relaxing filter thresholds to preserve fresh unique tokens (optimal at high compute once top-quality buckets saturate).

Data curation thresholds therefore cannot be compute-agnostic: a filter tuned to win on a small 1B-token ablation run can actively cripple a multi-trillion-token frontier run by starving it of unique tokens.

KEY IDEA

1. Data composition shifts the vertical offset $\log C(q)$ of a scaling law rather than its slope, creating a permanent multiplicative data gain; in practice, moderate small-scale empirical ranking (e.g., 150M models) is a strong baseline for picking mixtures.
2. Repeated data retains near-fresh value up to ≈ 4 epochs, saturates by ≈ 40 epochs, and eventually overfits under unbounded compute—which forces data filtering thresholds to loosen as planned compute grows (the Quality–Quantity Tradeoff).

We have now answered Part 1's questions regarding data volume, composition, repetition, and filtering. Even with an optimal dataset in hand, we still have to engineer the model itself: which architecture, optimizer, layer-to-width shape, and MoE sparsity will dominate at target scale rather than merely looking good on tiny prototypes? Part 2 turns to scaling laws for model engineering.

✎ Check Your Understanding

1. **Yes/No with justification:** If aggressive data filtering (keeping only the top 10% quality bucket) produces the lowest validation loss on a small-compute prototype run, should you keep that exact 10% filter when scaling compute up by $100\times$ on the same raw web pool?
   Answer: No. Because of the Quality–Quantity Tradeoff (QQT), scaling compute by $100\times$ will force heavy repetition of the small top-10% bucket, causing its error curve to flatten early; relaxing the filter to include fresh medium-quality buckets will cross below it and achieve lower loss.
2. **Quick conceptual check:** In Muennighoff et al.'s effective token formula $D' = U_D + U_D R_D^*(1 - e^{-R_D/R_D^*})$, what happens to D' in the two limits $R_D = 0$ (no repetition) and $R_D \to \infty$ (infinite repetition)?
   Answer: At $R_D = 0$, $1 - e^0 = 0$, so $D' = U_D$ (all unique tokens count fully). As $R_D \to \infty$, $e^{-R_D/R_D^*} \to 0$, so D' saturates at the finite ceiling $U_D(1 + R_D^*)$.

Section 4: Scaling Laws for Architecture, Optimizers, Shape, and Parameter Decoupling

Part 2 applies scaling laws to **model engineering**: predicting the impact of architecture, optimizer, model shape, sparsity, batch size, and learning rate on frontier-scale models *before* committing to training them. The workflow follows three steps:

1. **Train a sweep of small models** spanning several orders of magnitude in compute.
2. **Fit scaling curves** for each candidate design (such as Transformer versus LSTM, Muon versus AdamW, or varying MoE sparsity S).

3. **Inspect both offset and slope**—selecting the design that achieves the lowest loss at your *target* compute budget, not merely the design that happens to win at the tiniest tested scale.

1. Architecture Choice: Transformers, LSTMs, and Rank Inversions

Kaplan et al. (2020) compared recurrent language models (LSTMs) against Transformers by plotting **Test Loss** against **Non-Embedding Parameters** ($10^5 \to 10^9$) on log-log axes. In that plot, 1-layer LSTMs lie highest with a relatively flat slope, 2-to-4-layer LSTMs sit slightly below them, and the Transformer line sits well below both across every single parameter scale with a cleaner, steeper downward slope. Because Transformers dominate across the entire sweep, engineers did not need to train an LSTM at GPT-3 scale to know Transformers would win.

However, a single-scale comparison is dangerous when comparing closer architectural families. Tay et al. (2022) evaluated 11 architecture families—including ALBERT, DConv, Funnel Transformer, Transformer-GLU, MLP-Mixer, Performer, and sparse Switch Transformer—across training FLOPs. Their key finding is a critical warning: **architecture rankings can invert with scale**. Several variants engineered to look efficient at small scale (such as ALBERT and Performer) plateau or slow down as FLOPs grow, whereas Transformer-GLU and sparse Switch Transformers scale cleanly and overtake them at higher compute.

> **COMMON MISCONCEPTION**
> You might think that if Architecture A beats Architecture B by 0.05 loss at 100M parameters, Architecture A is guaranteed to stay ahead at 100B parameters. That breaks whenever the two architectures have different scaling *slopes* or different saturation points: an architecture with a slightly worse small-scale offset but a steeper negative slope will cross below and decisively beat a flatter rival at target scale.

2. Optimizer Choice: From Adam vs. SGD to Muon

Optimizers can likewise be evaluated via scaling curves. In an early baseline on depth-10 Recurrent Highway Networks, Hestness et al. (2017) plotted **Minimum Validation Loss** against **Training Characters** on log-log axes for SGD versus Adam:

- **SGD:** $L_{\text{SGD}}(m) = 5.37\, m^{-0.094}$
- **Adam:** $L_{\text{Adam}}(m) = 5.25\, m^{-0.095}$

The two lines run essentially parallel with nearly identical slopes (-0.094 versus -0.095), separated by a consistent downward **constant offset** favoring Adam.

> **WORKED EXAMPLE**
> **Worked Example: Quantifying the Offset Advantage of Adam over SGD**
>
> Using Hestness et al.'s fits $L_{\text{SGD}}(m) = 5.37\, m^{-0.094}$ and $L_{\text{Adam}}(m) = 5.25\, m^{-0.095}$, compute the loss of each optimizer at $m = 10^9$ training characters, and determine how many characters SGD requires to match Adam's loss at 10^9 characters.

1. At $m = 10^9$:

$$L_{\text{Adam}}(10^9) = 5.25 \times (10^9)^{-0.095} = 5.25 \times 10^{-0.855} \approx 5.25 \times 0.1396 \approx 0.7331$$

$$L_{\text{SGD}}(10^9) = 5.37 \times (10^9)^{-0.094} = 5.37 \times 10^{-0.846} \approx 5.37 \times 0.1426 \approx 0.7656$$

2. Solve $5.37\, m_{\text{SGD}}^{-0.094} = 0.7331$ for m_{SGD}:

$$m_{\text{SGD}}^{-0.094} = \frac{0.7331}{5.37} \approx 0.13652$$

$$m_{\text{SGD}} = (0.13652)^{-1/0.094} \approx 1.59 \times 10^9 \text{ characters}$$

Interpretation: Just as we saw with data mixtures in Section 3, because the data scaling exponent is shallow (≈ -0.095), a modest vertical offset between optimizers translates into a substantial $1.59\times$ **data/compute multiplier** for Adam over SGD.

In 2026 frontier practice, matrix-orthogonalized optimizers such as **Muon** have pushed optimizer efficiency further beyond AdamW:

- **DeepSeek-V4** (2026) trains with the **Muon** optimizer.
- **Kimi K3 / K3.1** (July 2026) uses **Per-Head Muon**.
- Controlled scaling studies by **Qiu et al.** demonstrate that pairing Muon with μP scaling and $1/\text{width}$ weight decay achieves roughly $1.4\times$ **compute efficiency** over heavily tuned AdamW across the 190M to 1.4B parameter range.

We'll revisit Muon's scaling rules when we cover learning-rate transfer in Section 5, and its FLOP-accounting caveats when we cover MFU measurement in Section 7.

3. Model Shape: Depth, Aspect Ratio, and Heads

Once total non-embedding parameter count N is fixed, how sensitive is pretraining loss to the geometric **shape** of a dense Transformer—specifically its depth n_{layer}, width d_{model}, aspect ratio $d_{\text{model}}/n_{\text{layer}}$, feed-forward ratio, and attention head dimension?

Kaplan et al. (2020) mapped **Loss Increase (%)** against **Aspect Ratio** $d_{\text{model}}/n_{\text{layer}}$ from 10^1 to 10^3 on a log horizontal axis. The resulting curve forms an extraordinarily wide, flat-bottomed valley:

- Across a $40\times$ **range of aspect ratios**, test loss stays within a few percent of the optimum, only rising steeply at extreme skinny (very deep, very narrow) or extreme pancake (very shallow, very wide) shapes.
- Comparing layer counts directly shows a large jump in quality moving from **1 layer to 2 layers**, followed by strong diminishing returns to extra depth below 10^7 parameters once $n_{\text{layer}} \geq 2$.
- Concretely, at matched non-embedding parameter count, a shallow, wide $(n_{\text{layer}}, d_{\text{model}}) = (6, 4288)$ model lands within **3%** of the loss of a deep, narrow $(48, 1600)$ model.

- Feed-forward expansion ratios and attention head dimensions exhibit similarly wide flat basins (where roughly 22% extra compute suffices to offset a 1% loss rise).

For pretraining cross-entropy, total parameter count dominates shape across a broad interior plateau—though we will see an important downstream caveat to this shape invariance at the end of Section 5.

Not All Parameters Are Equal: Embeddings and Sparse MoE

Counting every parameter identically breaks clean scaling laws in two major ways: **vocabulary embeddings** and **sparse Mixture-of-Experts (MoE)** layers.

First, **vocabulary embeddings** sit at the input and output projections and scale with $V \cdot d_{\text{model}}$, regardless of depth. Kaplan et al. observed that including embedding parameters distorts scaling curves across models of different depths, because shallow wide models carry a much higher fraction of embedding parameters than deep narrow models. Excluding embedding parameters collapses all models with $n_{\text{layer}} \geq 2$ onto a single clean power law. (We will see in Section 7, however, that excluding embedding FLOPs and parameters during compute-allocation fits introduced a subtle mathematical bias that drove the Kaplan–Chinchilla discrepancy.)

Second, **Sparse Mixture-of-Experts (MoE)** architectures deliberately break the link between model capacity and per-token compute.

> **DEFINITION**
> **Parameter Decoupling (MoE Sparsity S):** Separating a model's **total parameter count** N (memory and knowledge capacity stored across all experts) from its **active parameter count** N_a (the subset of parameters executed per token, which governs per-token FLOPs), controlled by the sparsity fraction S of inactive experts.

Abnar et al. (2025) mapped how sparsity S transforms compute-optimal scaling at a fixed **IsoFLOP** (constant total training FLOPs) budget using a pair of plots:

- **Left panel — Loss vs. Total Parameters N (log):** Plotting fixed-FLOP U-shaped loss curves for increasing sparsity levels S (from dense $S = 0\%$ up to $S = 95\%+$) shows each successive U-curve shifting **downward** (better loss at the same training FLOPs) and **rightward** (higher optimal total parameter count N^*). A dashed trajectory connecting the valley minima slopes downward and to the right: **higher** $S \implies$ **larger** N^*.
- **Right panel — Loss vs. Active Parameters N_a (log):** Plotting those exact same IsoFLOP runs against active parameters N_a shows each successive U-curve shifting **downward** (better loss) and **leftward** (smaller optimal active parameter count N_a^*). The trajectory connecting the minima slopes downward and to the left: **higher** $S \implies$ **smaller** N_a^*.

In other words, making an MoE sparser lets the model simultaneously store *more* total knowledge (N^* increases), execute *fewer* FLOPs per token (N_a^* decreases, allowing more training tokens at fixed FLOPs), and reach *lower* loss. Abnar et al. (2025) capture this interaction by fitting a full 2D IsoFLOP surface jointly over sparsity S and active parameters N_a.

Parameter Decoupling in 2026 Frontier Models

Current late-2026 frontier open models exploit parameter decoupling aggressively, separating total memory capacity from per-token active compute by an order of magnitude or more:

Model (Released)	Total N	Active N_a	MoE / Parameter Structure
DeepSeek-V4.1-Flash (Sept 2026)	552B backbone	16B decode / 8B prefill	384 routed experts (6 active) + 1 shared; CSA2 sparse attention; 3 MTP layers
Kimi K3 / K3.1 (July 2026)	2.8T	104B	896 routed experts (16 active) + 2 shared (Stable LatentMoE); 3:1 KDA/MLA; MXFP4 QAT
Qwen3.8-Flash-Next (Aug 2026)	125B (+51B n-gram)	6B	512 experts (10 active) + 1 shared; 51B n-gram embedding table in host memory
GLM-5.3 (2026)	78 layers, $d = 6144$	8/256 routed + 1 shared	256 experts (8 active) + 1 shared; MLA + DSA sparse attention (top-k 2048); 1 MTP layer

Notice how far modern architectures have moved beyond a single dense parameter count N: **Kimi K3 / K3.1** activates only 104B out of 2.8T total parameters ($\approx 3.7\%$ active); **DeepSeek-V4.1-Flash** even decouples decode active parameters (16B) from prefill active parameters (8B) out of a 552B backbone; and **Qwen3.8-Flash-Next** pairs a 6B-active / 125B-total MoE with a 51B-parameter n-gram embedding table stored in host memory. Consequently, classical scaling laws written purely in terms of a single monolithic dense N are now historical baselines rather than complete descriptions of frontier design.

KEY IDEA

1. Model engineering via small-scale sweeps must inspect both offset and slope because architecture rankings can invert with scale, while optimizer advances (SGD $\rightarrow$ AdamW $\rightarrow$ Muon) provide persistent multiplicative compute gains.
2. While dense Transformer pretraining loss is remarkably insensitive across a $40\times$ range of aspect ratios, not all parameters behave alike: sparse MoE layers and large embedding tables decouple total capacity N from per-token active compute N_a by more than $10\times$ in 2026 frontier models.

Even if you select the ideal architecture, optimizer, aspect ratio, and MoE sparsity, a frontier training run will still diverge or burn wasted compute if its batch size and learning rate are wrong. How do you choose batch size and transfer learning rates across orders of magnitude in width, depth, tokens, and MoE granularity without tuning them at target scale? Section 5 answers those questions.

✎ Check Your Understanding

1. **What would happen if…:** Holding total training FLOPs fixed on an MoE model, you increase sparsity S from 50% to 95%. According to Abnar et al. (2025), in which direction do optimal total parameters N^* and optimal active parameters N_a^* shift?
   Answer: Optimal total parameters N^* shift **upward** (more total capacity), optimal active parameters N_a^* shift **downward** (fewer FLOPs per token, enabling more tokens), and the minimum achievable loss drops.
2. **Fill-in-the-blank:** When comparing two candidate architectures across compute scales, picking the winner from a single small-scale run is risky because architecture rankings can _____ with scale if one design has a flatter slope or plateaus early.
   Answer: invert.

Section 5: Scaling Hyperparameters — Critical Batch Size, μP Transfer, and Downstream Caveats

Two optimization hyperparameters can make or break a large-scale run: the **batch size** B (which governs how much data parallelism you can extract to shorten wall-clock time) and the **learning rate** (along with initialization scale and output multipliers, which determine whether training converges cleanly as model size grows). Both obey precise scaling laws.

4. Batch Size and the Critical Batch Size (McCandlish et al., 2018)

Increasing batch size B allows you to split each optimizer step across more parallel workers, reducing the number of serial optimization steps S needed to reach a target loss. However, once a batch is already large enough to average out most gradient noise, doubling B again barely improves the gradient estimate—burning twice the compute per step for negligible extra step progress.

McCandlish et al. (2018) quantified this trade-off by sweeping batch size B to reach a fixed target loss, recording the required serial steps S and total examples processed $E = B \cdot S$. Across diverse tasks and architectures, the empirical frontier obeys a clean hyperbolic trade-off:

$$\frac{S}{S_{\min}} - 1 = \left(\frac{E}{E_{\min}} - 1 \right)^{-1}$$

Breaking this equation apart term by term:

- $S_{\min}$ is the theoretical minimum number of serial optimization steps needed to reach the target loss (approached in the infinite-batch limit $B \to \infty$, where every gradient is exact).
- $E_{\min}$ is the theoretical minimum number of total training examples needed to reach the target loss (approached in the tiny-batch limit $B \to 0$, where every single example gets its own parameter

update).

- The ratio $S/S_{\min} \geq 1$ is the step overhead relative to infinite-batch training, and $E/E_{\min} \geq 1$ is the data/compute overhead relative to tiny-batch training.
- Because $(S/S_{\min} - 1)$ and $(E/E_{\min} - 1)$ are reciprocals, pushing step overhead toward zero forces data overhead toward infinity, and vice versa.

Now substitute $E = B \cdot S$ into the hyperbolic equation to solve explicitly for S and E as functions of B. Define the **critical batch size** as the ratio of the two asymptotes:

$$\mathcal{B}_{\text{crit}} = \frac{E_{\min}}{S_{\min}} \approx \frac{\operatorname{tr}(\Sigma)}{\|\nabla L\|^2}$$

Substituting $E = BS$ and $E_{\min} = \mathcal{B}_{\text{crit}} S_{\min}$ into $(S/S_{\min} - 1)(E/E_{\min} - 1) = 1$:

$$\left(\frac{S}{S_{\min}} - 1\right)\left(\frac{B}{\mathcal{B}_{\text{crit}}}\frac{S}{S_{\min}} - 1\right) = 1$$

Let $u = S/S_{\min}$ and $r = B/\mathcal{B}_{\text{crit}}$. Expanding gives $ru^2 - (r+1)u + 1 = 1 \implies u(ru - r - 1) = 0$. Since $u > 0$, we obtain the exact closed-form expressions:

$$S = S_{\min}\left(1 + \frac{\mathcal{B}_{\text{crit}}}{B}\right), \qquad E = B \cdot S = E_{\min}\left(1 + \frac{B}{\mathcal{B}_{\text{crit}}}\right)$$

DEFINITION

Critical Batch Size ($\mathcal{B}_{\text{crit}}$): The batch size equal to $E_{\min}/S_{\min}$—approximated theoretically by the gradient noise scale $\operatorname{tr}(\Sigma)/\|\nabla L\|^2$, the ratio of gradient noise variance to squared true gradient norm—at which training sits directly on the efficiency knee requiring $2S_{\min}$ steps and $2E_{\min}$ examples.

Look at the statistical intuition behind $\mathcal{B}_{\text{crit}} \approx \operatorname{tr}(\Sigma)/\|\nabla L\|^2$: $\operatorname{tr}(\Sigma)$ is the trace of the per-example gradient covariance matrix (the total gradient noise power), while $\|\nabla L\|^2$ is the squared magnitude of the true full-dataset gradient (the signal power). Thus $\mathcal{B}_{\text{crit}}$ is the inverse gradient signal-to-noise ratio.

Plotting **Step Progress** $\varepsilon_{\text{opt}}/\varepsilon_{\max}$ against **Batch Size / Noise Scale** $B/\mathcal{B}_{\text{crit}}$ on a horizontal log axis illustrates two distinct regimes separated by a vertical dashed line at $B/\mathcal{B}_{\text{crit}} = 1$:

- **Left of the knee ($B \ll \mathcal{B}_{\text{crit}}$ — Perfect scaling / linear speedup):** Doubling the batch size cuts serial steps nearly in half while leaving total examples $E \approx E_{\min}$ almost unchanged.
- **At the knee ($B = \mathcal{B}_{\text{crit}}$):** Marked by a point right on the bend, training requires exactly $S = 2S_{\min}$ serial steps and $E = 2E_{\min}$ examples—balancing wall-clock time and compute efficiency.
- **Right of the knee ($B \gg \mathcal{B}_{\text{crit}}$ — Diminishing returns):** Step progress flattens toward its horizontal ceiling; doubling the batch size barely reduces serial steps while nearly doubling total compute E.

Critical Batch Size Grows as Target Loss Drops (Kaplan et al., 2020)

Crucially, $\mathcal{B}_{\text{crit}}$ is not a fixed constant throughout training. As a model trains and its loss L drops closer to a local minimum, the true gradient magnitude $\|\nabla L\|$ shrinks much faster than the stochastic gradient noise $\text{tr}(\Sigma)$. Because the signal-to-noise ratio drops, the noise scale $\text{tr}(\Sigma)/\|\nabla L\|^2$ grows—meaning **larger batches become productive later in training**.

Kaplan et al. (2020) plotted $\mathcal{B}_{\text{crit}}$ (in tokens, from $10^3 \to 10^6$ on a vertical log axis) against **Train Loss** L (decreasing left to right from $10 \to 3$). Empirical curves for both $N = 3\text{M}$ and $N = 85\text{M}$ models climb together along a single dashed power-law trend line, remarkably insensitive to model size and governed primarily by loss L:

$$\mathcal{B}_{\text{crit}}(L) \approx 2.1 \times 10^8 \text{ tokens} \cdot L^{-4.8}$$

Because $E = E_{\text{min}}(1 + B/\mathcal{B}_{\text{crit}})$ and compute is proportional to examples processed, Kaplan et al. relate actual training compute C spent at batch size B to the ideal minimum compute C_{min} (achieved as $B \to 0$) via:

$$C_{\text{min}}(C) \equiv \frac{C}{1 + B/\mathcal{B}_{\text{crit}}(L)}$$

This is the exact quantity C_{min} that appeared in Panel 1 of Kaplan et al.'s power-law plot in Section 1. Minimum compute occurs at $B \ll \mathcal{B}_{\text{crit}}$, whereas larger batches deliberately trade extra compute for fewer serial steps and higher cluster parallelism.

WORKED EXAMPLE

Worked Example: Computing $\mathcal{B}_{\text{crit}}$ and Batch-Size Compute Overhead

Using Kaplan et al.'s fit $\mathcal{B}_{\text{crit}}(L) = 2.1 \times 10^8 \cdot L^{-4.8}$ tokens:

1. Compute $\mathcal{B}_{\text{crit}}$ early in training when $L = 4.0$, and later in training when $L = 2.0$:
   - At $L = 4.0$:

     $\mathcal{B}_{\text{crit}}(4.0) = 2.1 \times 10^8 \times 4.0^{-4.8} \approx 2.1 \times 10^8 \times 0.001289 \approx 2.71 \times 10^5 \text{ tokens } (0.2$
   - At $L = 2.0$:

     $\mathcal{B}_{\text{crit}}(2.0) = 2.1 \times 10^8 \times 2.0^{-4.8} \approx 2.1 \times 10^8 \times 0.03590 \approx 7.54 \times 10^6 \text{ tokens } (7.54$
2. Now suppose you train with a fixed batch size of $B = 2.71 \times 10^5$ tokens at $L = 4.0$ ($B = \mathcal{B}_{\text{crit}}$) versus $B = 1.08 \times 10^6$ tokens ($B = 4\,\mathcal{B}_{\text{crit}}$):
   - At $B = \mathcal{B}_{\text{crit}}$: $S = 2S_{\text{min}}$ and $E = 2E_{\text{min}}$ ($C_{\text{min}} = C/2$, meaning 50% compute efficiency).
   - At $B = 4\,\mathcal{B}_{\text{crit}}$: $S = 1.25\,S_{\text{min}}$ and $E = 5E_{\text{min}}$ ($C_{\text{min}} = C/5$, meaning 80% of compute is lost to diminishing returns for only a 37.5% reduction in serial steps relative to $B = \mathcal{B}_{\text{crit}}$).

> **Interpretation:** Cutting the loss in half from 4.0 to 2.0 increases the useful batch size by $2^{4.8} \approx 27.9\times$ (from 0.27M to 7.54M tokens). This is why large pretraining runs often warm up their batch size over training, avoiding severe early compute waste while exploiting massive parallelism at low loss.

5. Learning Rates and μP Transfer (Yang et al., 2022; Yao et al., 2024)

Tuning the learning rate directly on a 100B+ model is financially impossible. Ideally, you would sweep learning rates on a small proxy model M and copy the winning learning rate straight to the target model M'.

Under **Standard Parametrization (SP)**—where hidden weights use standard fixed initialization scaling and every layer shares one learning rate—this transfer fails. A side-by-side comparison of **Train Loss** versus $\log_2(\text{LR})$ across progressively wider models demonstrates the failure and its cure:

- **Left panel (Standard Param, SP):** Three U-shaped loss curves for increasing model widths shift steadily to the **left** as width grows, marked by an arrow labeled "optimum shifts." The optimal learning rate of a narrow proxy model causes the wider target model to blow up or perform poorly.
- **Right panel (μP, Maximal Update Parametrization):** The three U-shaped loss curves drop vertically downward as width grows, while their minima align along a single vertical dashed line labeled "optimum stable." The optimal base learning rate tuned on the narrowest model transfers directly without re-tuning.

> **DEFINITION**
> **Maximal Update Parametrization (μP):** A set of width-dependent scaling rules for weight initialization variance, per-parameter learning rates, and output multipliers that keeps activation magnitudes and feature updates $O(1)$ across widths, causing the optimal base learning rate to remain invariant as a model is widened.

When scaling model width by a factor r from a proxy model M to a wider model M' ($M \to M'$, where $r = d'_{\text{model}}/d_{\text{model}}$), μP scales three hyperparameter groups differently depending on whether a parameter is **matrix-like** (a 2D hidden projection tensor whose fan-in and fan-out both scale with width) or belongs to **"others"** (vocabulary embeddings and 1D normalization vectors):

Hyperparameter (weight group)	Proxy Model M	Scaled Model M' (width factor r)
AdamW LR (matrix-like)	l	l/r
AdamW LR (others)	l	l
Init variance (matrix-like)	σ	σ/r
Init variance (others)	σ	σ
Multiplier (output / `lm_head`)	τ	τ/r
Multiplier (others)	τ	τ

(Note on notation: in this table, σ denotes the initialization *variance* of the weights, which scales as $1/r$ —equivalently, the initialization standard deviation scales as $1/\sqrt{r}$.)

WORKED EXAMPLE

Worked Example: Transferring Hyperparameters via μP Across an $8\times$ Width Jump

Suppose you tune a small proxy Transformer M of width $d_{\text{model}} = 512$ and find optimal AdamW hyperparameters: base learning rate $l = 4 \times 10^{-3}$, hidden weight initialization variance $\sigma = 0.0016$ (standard deviation 0.04), and output `lm_head` logit multiplier $\tau = 1.0$. You now scale up to a target model M' of width $d'_{\text{model}} = 4096$. Compute every transferred hyperparameter.

1. **Width scaling factor:** $r = 4096/512 = 8$.
2. **AdamW learning rates:**
   - Hidden matrix-like projections (attention Q/K/V/O and FFN matrices): $l' = l/r = (4 \times 10^{-3})/8 = 5 \times 10^{-4}$.
   - Embeddings and 1D LayerNorm/RMSNorm parameters ("others"): unchanged at $l' = 4 \times 10^{-3}$.
3. **Initialization variance:**
   - Hidden matrix-like projections: $\sigma' = \sigma/r = 0.0016/8 = 0.0002$ (standard deviation $\sqrt{0.0002} \approx 0.0141$).
   - Embeddings and 1D norms ("others"): unchanged at $\sigma' = 0.0016$.
4. **Output multiplier:**
   - Output `lm_head` logits are multiplied by $\tau' = \tau/r = 1.0/8 = 0.125$.
   - All other multipliers remain $\tau' = 1.0$.

Interpretation: Without re-running a single learning-rate sweep at width 4096, the base learning rate $l = 4 \times 10^{-3}$ tuned cheaply at width 512 transfers cleanly because the $1/r$ rescalings cancel the width growth in hidden matrix updates and output dot products.

2025–2026 Extensions to μP Transfer

Original μP stabilized hyperparameter transfer across **dense model width** alone. Modern pretraining simultaneously scales depth, training tokens, MoE expert granularity, and non-AdamW optimizers. Four major 2025–2026 developments closed these gaps:

Work	Scaling Axis	Key Result
CompleteP (2025)	Model **depth** (and width)	Extends μP hyperparameter transfer cleanly across depth, closing a major gap in original μP
MoE μP study (2025)	Sparse **MoE**	μP transfer holds across MoE width, **but breaks** when **expert granularity** changes
Qiu et al. (Muon scaling)	**Muon** optimizer	Derived μP plus $1/\text{width}$ **weight decay** for Muon—achieves about $1.4\times$ efficiency over tuned AdamW (190M–1.4B)
HyperP (Mar 2026)	Width, depth, tokens & MoE granularity	Transfers a **single base learning rate** simultaneously across all four axes

With **HyperP** and optimizer-aware μP, engineering teams can tune hyperparameters once on small proxy runs even when targeting deep, fine-grained MoE models trained with Muon.

Caution: Upstream Loss vs. Downstream Accuracy (Tay et al., 2023)

Before relying exclusively on pretraining cross-entropy to lock in a model shape, consider an empirical warning from Tay et al. (2023), who evaluated the exact same set of Transformer models—spanning diverse depth/width shapes (`NL24`, `Large`, `NL8-XXL`, `NL32-XL`, `NL6-XXXL`, and `NL12`)—on both upstream pretraining perplexity and downstream SuperGLUE accuracy:

- **Left panel (Negative Log-Perplexity vs. Log Params):** All six models fall tightly along a clean upward diagonal line. Upstream perplexity scales smoothly and depends almost exclusively on total parameter count, seemingly confirming Kaplan et al.'s flat shape plateau.
- **Right panel (SuperGLUE Accuracy vs. Log Params):** The exact same six models scatter wildly. Most strikingly, the deep model **`NL32-XL`** shoots to the very top of the plot—decisively beating much wider, higher-parameter models such as **`NL6-XXXL`** and **`NL12`** even though those wider models achieved *better* upstream perplexity.

> **COMMON MISCONCEPTION**
> You might think that because upstream pretraining loss sits on a flat $40\times$ aspect-ratio plateau, any shallow-and-wide shape that matches a deep-and-narrow model's perplexity will deliver identical downstream capability. That breaks because downstream benchmarks such as SuperGLUE are strongly **shape-sensitive**, often rewarding depth far more than pretraining perplexity suggests.

The practical engineering rule is clear: use pretraining loss scaling laws to prune clearly inferior designs and predict compute budgets, then verify promising model shapes directly on downstream evaluations before launching the full run.

KEY IDEA

1. Batch size follows McCandlish et al.'s hyperbolic trade-off centered at the critical batch size $\mathcal{B}_{\text{crit}} \approx 2.1 \times 10^8 L^{-4.8}$, which grows rapidly as target loss drops; meanwhile, μP and its 2025–2026 extensions (CompleteP, Qiu et al., HyperP) allow a single base learning rate to transfer across width, depth, tokens, MoE granularity, and Muon.
2. Smooth upstream perplexity scaling hides strong shape sensitivity in downstream benchmarks like SuperGLUE, so loss scaling must be paired with downstream verification of candidate shapes.

We can now predict data quality effects, pick an architecture and optimizer, select sparsity and shape, set the batch size relative to $\mathcal{B}_{\text{crit}}$, and transfer learning rates with μP/HyperP. One central resource-allocation question remains: given a fixed compute budget C, exactly how many parameters N should we build and how many tokens D should we train on? Part 3 resolves compute allocation.

✎ Check Your Understanding

1. **Quick computation:** Under μP, if you quadruple the width ($r = 4$) of a proxy Transformer trained with AdamW, by what factor should you multiply (a) the learning rate of hidden attention/FFN matrices, (b) the learning rate of token embeddings, and (c) the output `lm_head` multiplier?
   Answer: (a) Multiply hidden matrix learning rates by $1/4$; (b) keep embedding learning rates unchanged (factor of 1); (c) multiply the output `lm_head` multiplier by $1/4$.
2. **Yes/No with justification:** If you increase your training batch size from $B = \mathcal{B}_{\text{crit}}$ to $B = 10\,\mathcal{B}_{\text{crit}}$, will you cut the number of serial training steps roughly by a factor of 10?
   Answer: No. Since $S = S_{\min}(1 + \mathcal{B}_{\text{crit}}/B)$, increasing B from $\mathcal{B}_{\text{crit}}$ to $10\,\mathcal{B}_{\text{crit}}$ only reduces serial steps from $2S_{\min}$ to $1.1S_{\min}$ (less than a $2\times$ speedup) while increasing total compute from $2E_{\min}$ to $11E_{\min}$ ($5.5\times$ more compute).

Section 6: Joint Model–Data Scaling Laws and the Kaplan–Chinchilla Debate

The single-variable power laws from Section 1—$L(N)$ at infinite data and $L(D)$ at infinite parameters—cannot answer our core budget question on their own. Feeding trillions of tokens to a tiny model wastes compute once model capacity saturates; building a massive model and training it on too few tokens leaves its weights underfitted. To allocate a fixed training compute budget C_0 optimally between parameters N and tokens D, we need a **joint scaling law** $L(N, D)$.

Joint Parametric Loss Surfaces and Constrained Minimization

Two classic parametric forms model joint (N, D) scaling:

- **Rosenfeld et al. (2020):**

$$\text{Error}(N, D) = n^{-\alpha} + m^{-\beta} + C$$

  (Here n denotes model scale, m denotes data scale, and C is the irreducible error floor; in standard LLM notation, $L(N, D) = A/N^{\alpha} + B/D^{\beta} + E$.)

- **Kaplan et al. (2020):**

$$\text{Error}(N, D) = \left[m^{-\alpha} + n^{-1}\right]^{\beta}$$

Rosenfeld et al. (2020) tested whether fitting this simple additive surface on tiny grids actually predicts large-scale loss accurately:

Dataset (Rosenfeld et al., 2020)	Fit Grid Used	Extrapolation Bias (μ)	Extrapolation Std (σ)
WikiText-103 (test loss)	$\leq 1/16$ model, $\leq 1/8$ data	$+0.5\%$	1.689%
ImageNet (top-1 error)	$\leq 1/16$ model, $\leq 1/8$ data	-4.5%	4.681%

Fitting on models at most $1/16$ the target size and datasets at most $1/8$ the target size extrapolated to full WikiText-103 test loss with just $+0.5\%$ mean bias.

Once $L(N, D)$ is fitted, **compute-optimal allocation** becomes a straightforward constrained minimization problem:

$$\min_{N,D} \; L(N, D) \qquad \text{subject to} \qquad C(N, D) \leq C_0$$

For a standard dense Transformer, recall that forward plus backward training compute is approximately $C(N, D) \approx 6ND$ FLOPs (roughly $2N$ FLOPs per token in the forward pass and $4N$ FLOPs per token in the backward pass). Let us solve this constrained optimization analytically for the additive form $L(N, D) = AN^{-\alpha} + BD^{-\beta} + E$ subject to $6ND = C$:

1. Eliminate D using the compute constraint $D = C/(6N)$:

$$L(N) = AN^{-\alpha} + B\left(\frac{C}{6}\right)^{-\beta} N^{\beta} + E$$

2. Differentiate with respect to N and set $\frac{dL}{dN} = 0$:

$$-\alpha A N^{-\alpha-1} + \beta B \left(\frac{C}{6}\right)^{-\beta} N^{\beta-1} = 0$$

3. Multiply through by N and recognize $(C/6)^{-\beta}N^{\beta} = D^{-\beta}$:

$$\alpha A N^{-\alpha} = \beta B D^{-\beta}$$

   Notice the clean economic optimality condition: at the compute-optimal point, the weighted marginal loss contributions of parameters and data are equal.
4. Substitute $D = C/(6N)$ back in and solve for N_{opt} and D_{opt}:

$$N_{\text{opt}}^{\alpha+\beta} = \frac{\alpha A}{\beta B}\left(\frac{C}{6}\right)^{\beta} \implies N_{\text{opt}} \propto C^{a}, \qquad D_{\text{opt}} \propto C^{b}$$

   where the optimal allocation exponents are:

$$a = \frac{\beta}{\alpha+\beta}, \qquad b = \frac{\alpha}{\alpha+\beta}, \qquad a + b = 1$$

Because $a + b = 1$, every extra factor of $10\times$ in compute C must split its exponents between N and D so their product $ND \propto C^{a+b} = C^{1}$ matches the compute budget. Notice the inverse pairing: a (the parameter scaling exponent) is proportional to β (the *data* decay exponent), while b (the data scaling exponent) is proportional to α (the *parameter* decay exponent). Whichever resource improves loss more steeply per order of magnitude requires *less* of the compute budget to keep up, freeing more budget for the slower-improving resource.

> **DEFINITION**
> **Compute-Optimal Scaling ($N_{\text{opt}} \propto C^{a},\ D_{\text{opt}} \propto C^{b}$):** The allocation of parameters N and training tokens D that minimizes training/validation loss for a fixed training FLOP budget C, with exponents satisfying $a + b = 1$ under $C \approx 6ND$.

Kaplan vs. Chinchilla: A Historic Disagreement

When two landmark papers measured (a, b) for autoregressive Transformers, they arrived at dramatically different prescriptions, visualized on a log-log plot of **Parameters** N ($10\text{M} \to 1\text{T}$) against **Training Compute FLOPs** ($10^{17} \to 10^{25}$):

- **Kaplan et al. (2020)** traced a steep dashed line:

$$N_{\text{opt}} \propto C^{0.73}, \qquad D_{\text{opt}} \propto C^{0.27}$$

  This predicted that as compute grows, almost three-quarters of the exponent should go into building gigantic models trained on relatively few tokens. Historical models built in this era—**GPT-3 175B (2020)**, **Gopher 280B**, and **Megatron-Turing 530B**—all sit high up near Kaplan's steep line.
- **Chinchilla (Hoffmann et al., 2022)** traced a substantially flatter solid blue line:

$$N_{\text{opt}} \propto C^{0.50}, \qquad D_{\text{opt}} \propto C^{0.50}$$

  This prescribed scaling parameters N and tokens D **equally**—specifically around ≈ 20 **tokens per**

parameter. At the exact same compute budget as Gopher 280B (5.76×10^{23} FLOPs), the $4\times$ smaller **Chinchilla 70B (2022)** trained on $4\times$ more data decisively outperformed Gopher 280B.

Chinchilla in Depth: Three Fitting Approaches

To ensure their $a \approx 0.50, b \approx 0.50$ finding was not an artifact of one curve-fitting procedure, Hoffmann et al. (2022) estimated (a, b) via **three independent approaches**:

Approach	Coeff. a ($N_{\text{opt}} \propto C^a$)	Coeff. b ($D_{\text{opt}} \propto C^b$)	Gopher Budget (5.76×10^{23} FLOPs)
1. Minimum over training curves	$0.50\ (0.488, 0.502)$	$0.50\ (0.501, 0.512)$	67B params, 1.5T tokens
2. IsoFLOP profiles	$0.49\ (0.462, 0.534)$	$0.51\ (0.483, 0.529)$	63B params, 1.4T tokens
3. Parametric loss fit (original)	$0.46\ (0.454, 0.455)$	$0.54\ (0.542, 0.543)$	40B params *(later corrected)*
Kaplan et al. (2020) *(comparison)*	0.73	0.27	Much larger N, fewer tokens

Understanding the visual geometry of **Methods 1 and 2** makes clear why Method 2 became an industry standard:

- **Method 1 — Training Curve Envelope:** Plotting **Train Loss** against **Training FLOPs** ($10^{17} \to 10^{22}$) for models ranging from 70M to 10B parameters—each trained across 4 different cosine learning-rate cycle lengths—yields a dense bundle of overlapping downward trajectories. Extracting the lowest achievable loss at each FLOP count traces a straight dashed **lower envelope**, and recording the (N^*, D^*) of the runs touching that envelope gives power laws with $a = 0.50, b = 0.50$.
- **Method 2 — IsoFLOP Profiles:** Fixing a discrete sequence of constant FLOP budgets (from 6×10^{18} up to 3×10^{21} FLOPs), matching the cosine learning-rate decay length exactly to each budget, and sweeping parameter count N ($100\text{M} \to 30\text{B}$) produces a stacked family of clean **convex U-shaped valleys** when plotting Train Loss against N. Fitting a parabola to each U-valley locates its minimum $N^*(C)$, and connecting those minima across FLOP budgets traces a clean straight power law ($a = 0.49, b = 0.51$).

> **DEFINITION**
> **IsoFLOP Profile:** A curve obtained by holding the total training FLOP budget C strictly constant (with learning-rate decay matched to that budget) and plotting final loss against parameter count N (with $D = C/(6N)$ adjusting inversely), forming a convex U-shaped valley whose minimum directly identifies the compute-optimal model size $N^*(C)$.

Method 3 and the Besiroglu et al. (2024) Forensic Reanalysis

In the original Chinchilla paper, **Method 3**—running models across an (N, D) grid and fitting the joint parametric surface $\hat{L}(N, D) = E + A/N^{\alpha} + B/D^{\beta}$—produced slightly asymmetric exponents ($a = 0.46, b = 0.54$), predicting only 40B parameters for the Gopher budget and implying that the optimal token-to-parameter ratio D/N should climb rapidly with compute.

Besiroglu et al. (2024) performed a forensic reanalysis by recovering the raw data points directly from Hoffmann et al.'s published figures. They discovered a **systematic negative residual bias** in Hoffmann et al.'s original Method 3 optimization fit. Re-estimating the joint parametric surface properly on the recovered Chinchilla data yields:

$$L(N, D) = \frac{482}{N^{0.35}} + \frac{2085}{D^{0.37}} + 1.82 \qquad \text{(Besiroglu et al., 2024)}$$

Decomposing this equation term by term:

- 1.82 is the irreducible cross-entropy floor E (in nats per token) of the text distribution.
- $482/N^{0.35}$ is the finite-capacity excess loss from having N parameters rather than an infinite model.
- $2085/D^{0.37}$ is the finite-data excess loss from training on D tokens rather than an infinite corpus.

With these corrected coefficients ($482, 2085, 1.82$—note that these are Besiroglu et al.'s 2024 re-estimation, not Hoffmann et al.'s original table), Method 3 aligns cleanly with Methods 1 and 2 and the $D/N \approx 20$ rule of thumb, eliminating the false claim that optimal D/N diverges rapidly upward with compute.

WORKED EXAMPLE

Worked Example: Solving Compute-Optimal Allocation on the Besiroglu et al. (2024) Surface

Apply our constrained minimization derivation to $L(N, D) = 482N^{-0.35} + 2085D^{-0.37} + 1.82$ at the Gopher compute budget $C = 5.76 \times 10^{23}$ FLOPs (using $C = 6ND$).

1. **Compute allocation exponents a and b:** With $\alpha = 0.35$ and $\beta = 0.37$, we have $\alpha + \beta = 0.72$:

   $$a = \frac{\beta}{\alpha + \beta} = \frac{0.37}{0.72} \approx 0.514, \qquad b = \frac{\alpha}{\alpha + \beta} = \frac{0.35}{0.72} \approx 0.486$$

   Both sit right near 0.50, confirming nearly equal scaling of N and D.
2. **Solve for N_{opt}:** First compute $C/6 = (5.76 \times 10^{23})/6 = 9.6 \times 10^{22}$ and the prefactor ratio:

   $$\frac{\alpha A}{\beta B} = \frac{0.35 \times 482}{0.37 \times 2085} = \frac{168.7}{771.45} \approx 0.2187$$

   Now evaluate $N_{\text{opt}}^{0.72} = 0.2187 \times (9.6 \times 10^{22})^{0.37}$ using base-10 logs:

$$\log_{10}(N_{\text{opt}}^{0.72}) = \log_{10}(0.2187) + 0.37\log_{10}(9.6\times10^{22}) \approx -0.6602 + 0.37(22.9823) \approx$$

$$\log_{10}(N_{\text{opt}}) = \frac{7.8432}{0.72} \approx 10.893 \implies N_{\text{opt}} \approx 78.2\text{B parameters}$$

3. **Solve for D_{opt} and predicted loss:**

$$D_{\text{opt}} = \frac{9.6\times10^{22}}{7.82\times10^{10}} \approx 1.23\text{T tokens}, \qquad \frac{D_{\text{opt}}}{N_{\text{opt}}} \approx 15.7\ \text{tokens/param}$$

$$L(N_{\text{opt}}, D_{\text{opt}}) = \frac{482}{(7.82\times10^{10})^{0.35}} + \frac{2085}{(1.23\times10^{12})^{0.37}} + 1.82 \approx 0.074 + 0.070 + 1.8$$

Interpretation: Even using the two-decimal rounded exponents (0.35 and 0.37) from the re-estimated surface, the optimal model lands right in the $\approx 60\text{B}$–80B Chinchilla neighborhood (rather than Kaplan's much larger models or the original Method 3's biased 40B estimate), and notice how the finite-parameter loss (0.074) and finite-data loss (0.070) are nearly balanced at the optimum.

KEY IDEA

1. Minimizing an additive joint loss surface $L(N, D) = AN^{-\alpha} + BD^{-\beta} + E$ under $C = 6ND$ yields power-law compute allocations $N_{\text{opt}} \propto C^{\beta/(\alpha+\beta)}$ and $D_{\text{opt}} \propto C^{\alpha/(\alpha+\beta)}$ with $a + b = 1$.
2. Across training-curve envelopes, IsoFLOP profiles, and Besiroglu et al.'s (2024) corrected Method 3 parametric fit, Chinchilla scaling establishes $a \approx 0.50, b \approx 0.50$ (≈ 20 tokens per parameter) for training-compute optimality.

All three Chinchilla methods now agree with each other—which makes the contrast with Kaplan et al. ($a = 0.73, b = 0.27$) even sharper. How could two rigorous research groups measure the same Transformer architecture and disagree so drastically on a? And once we resolve that discrepancy, is training-compute optimality even the right target for a deployed model? Section 7 resolves both questions.

✎ Check Your Understanding

1. **Quick computation:** For a joint loss surface $L(N, D) = AN^{-\alpha} + BD^{-\beta} + E$ with $\alpha = 0.30$ and $\beta = 0.45$ under $C = 6ND$, what are the compute-optimal allocation exponents a (for $N_{\text{opt}} \propto C^a$) and b (for $D_{\text{opt}} \propto C^b$)?
   Answer: $a = \frac{\beta}{\alpha+\beta} = \frac{0.45}{0.30+0.45} = 0.60$, and $b = \frac{\alpha}{\alpha+\beta} = \frac{0.30}{0.75} = 0.40$ (satisfying $a + b = 1$).
2. **Fill-in-the-blank:** In Chinchilla Method 2 (IsoFLOP profiles), holding total FLOPs C constant with matched cosine learning-rate decay and sweeping model size N produces a _____-shaped valley whose minimum gives $N^*(C)$.
   Answer: U-shaped (or convex parabolic).

Section 7: Reconciling Kaplan and Chinchilla, Accounting Pitfalls, and Inference-Optimal Scaling

For several years after Chinchilla was published, the conventional story blamed Kaplan et al.'s $a = 0.73$ exponent solely on their use of a fixed cosine learning-rate schedule instead of tuning cosine decay to each compute budget. Two 2024 papers—Porian et al. (NeurIPS 2024) and Pearce & Song (2024)—systematically dissected the discrepancy and uncovered a much deeper set of methodological and accounting lessons.

Reconciliation 1: Step-by-Step Empirical Ablation (Porian et al., NeurIPS 2024)

Porian et al. started by faithfully reproducing Kaplan et al.'s experimental setup and then changed one methodological choice at a time, tracking both the fitted exponent a (with confidence interval) and the projected optimal model size N^* at frontier budget. Tracing their 5-step ablation diagram reveals exactly where the gap came from:

- **Step a — Reproducing Kaplan:** $a = 0.835$ $(0.82, 0.85)$, projecting a massive $N^* = 3\text{T}$ parameters.
- **Step b — + Last-Layer FLOPs:** Including the FLOPs of the final unembedding/output layer (which Kaplan excluded) drops the exponent immediately to $a = 0.706$ $(0.69, 0.72)$ and slashes projected N^* nearly fourfold to 787B.
- **Step c — + Correct Warmup:** Fixing excessive learning-rate warmup (which penalized small-budget runs that spent too much of their budget warming up) drops the exponent further to $a = 0.602$ $(0.59, 0.62)$, lowering projected N^* to 292B.
- **From Step c, the ablation branches into two independent tests:**
  - **Step d — + Cosine Decay (no optimizer tuning):** Adding Chinchilla-style cosine decay *without* tuning batch size and learning rate only nudges the exponent slightly to $a = 0.571$ $(0.56, 0.59)$ ($N^* = 183\text{B}$).

- **Step e — + Optimizer Tuning (no cosine decay):** Tuning batch size and learning rate properly—even *without* cosine decay—brings the exponent all the way down to $a = 0.497$ $(0.49, 0.50)$ and projected $N^* = 77\text{B}$, matching Chinchilla!

> **COMMON MISCONCEPTION**
> You might think that cosine learning-rate decay was the main reason Chinchilla obtained $a = 0.50$ while Kaplan obtained $a = 0.73$. Porian et al.'s ablation disproves that: three concrete differences —**excluding last-layer FLOPs**, **excessive warmup** at small compute budgets, and **untuned batch size and learning rate**—drove the gap, while cosine decay itself had a much smaller effect than proper optimizer tuning.

Reconciliation 2: The Geometry of Embedding Accounting (Pearce & Song, 2024)

Pearce & Song (2024) revealed a second, purely mathematical source of the Kaplan–Chinchilla gap. Starting from a **single identical underlying loss curve**—Besiroglu et al.'s (2024) re-fitted Chinchilla surface $482/N_T^{0.35} + 2085/D^{0.37} + 1.82$, simulated over Kaplan's exact small-to-medium model size range (1K to 1.5B parameters)—they branched the analysis solely by *how parameters and FLOPs are counted*:

- **Upper branch (Chinchilla accounting — Count Total Parameters N_T and Total Compute C_T):** Computing the optimal frontier yields a clean global power law $N_T^* \propto C_T^{0.51}$, matching Chinchilla's 0.50.
- **Lower branch (Kaplan accounting — Count Non-Embedding Parameters $N_{\setminus E}$ and Non-Embedding Compute $C_{\setminus E}$):** Re-plotting the *exact same optimal models* with embedding parameters and embedding FLOPs stripped out creates a **curved frontier** at small scales whose local log-log slope is $N_{\setminus E}^* \propto C_{\setminus E}^{0.78}$—closely mimicking Kaplan's 0.73, even though the true asymptotic large-N slope is 0.51.

Why does stripping out embeddings inflate the local slope? At very small model sizes, the vocabulary embedding matrix ($V \cdot d_{\text{model}}$) represents the majority of total parameters N_T. As the model grows, hidden layers scale quadratically with width while embeddings scale linearly, so $N_{\setminus E}$ grows much faster than N_T across small-to-medium models.

> **WORKED EXAMPLE**
> **Worked Example: How Excluding Embeddings Inflates the Fitted Exponent a**
>
> Consider two models that lie strictly on Chinchilla's $a = 0.50$ frontier ($D = 20N_T$, $C_T = 6N_T D$):
>
> - **Small model 1:** Non-embedding parameters $N_{\setminus E,1} = 1\text{M}$, embedding parameters $N_{\text{emb},1} = 25.6\text{M} \implies$ total $N_{T,1} = 26.6\text{M}$, $D_1 = 20N_{T,1} = 532\text{M}$ tokens.
> - **Medium model 2:** Non-embedding parameters $N_{\setminus E,2} = 64\text{M}$, embedding parameters $N_{\text{emb},2} = 102.4\text{M} \implies$ total $N_{T,2} = 166.4\text{M}$, $D_2 = 20N_{T,2} = 3.328\text{B}$ tokens.

Compare the log-log slope $a = \frac{\log(N_2/N_1)}{\log(C_2/C_1)}$ under total accounting versus non-embedding accounting:

1. **Total accounting (N_T, $C_T = 6N_T D$):**

$$C_{T,1} = 6(26.6 \times 10^6)(532 \times 10^6) \approx 8.49 \times 10^{16}, \quad C_{T,2} = 6(166.4 \times 10^6)(3.328 \times 1($$

$$a_{\text{total}} = \frac{\ln(166.4/26.6)}{\ln(3.32 \times 10^{18}/8.49 \times 10^{16})} = \frac{1.833}{3.667} = 0.500$$

2. **Non-embedding accounting ($N_{\setminus E}$, $C_{\setminus E} = 6N_{\setminus E} D$):**

$$C_{\setminus E,1} = 6(10^6)(532 \times 10^6) = 3.19 \times 10^{15}, \quad C_{\setminus E,2} = 6(64 \times 10^6)(3.328 \times 10^9) = 1.2$$

$$a_{\setminus E} = \frac{\ln(64/1)}{\ln(1.28 \times 10^{18}/3.19 \times 10^{15})} = \frac{4.159}{5.993} \approx 0.694$$

Interpretation: Without changing a single training run, merely stripping embedding parameters out of both axes inflates the measured slope from the true 0.500 to 0.694—reproducing the exact mechanism identified by Pearce & Song (2024). Together with Porian et al. (2024), this completely reconciles the Kaplan–Chinchilla discrepancy.

FLOP and MFU Accounting Caveats in 2026

The Kaplan–Chinchilla reconciliation establishes an ironclad methodological rule: **always state the exact FLOP formula behind every scaling curve and efficiency comparison.** Dropping last-layer and embedding FLOPs alone shifted Porian et al.'s fitted exponent from 0.835 to 0.706.

In late-2026 training practice, Model FLOPs Utilization (MFU) faces two additional accounting pitfalls:

- **MoE reporting:** Because sparse routing and communication complicate standardized MFU definitions, vendors increasingly publish raw **TFLOP/s per GPU** for MoE models rather than MFU percentages.
- **Optimizer FLOP inflation:** Matrix-orthogonalized optimizers like **Muon** run iterative **Newton–Schulz matrix multiplications** during the optimizer step. Those dense matrix multiplications are easy to execute at high hardware utilization, which **inflates counted MFU** without directly translating into faster loss reduction per wall-clock second.

Best practice is therefore to report **wall-clock time** alongside an **explicitly defined FLOP formula** that specifies whether and how attention, embeddings, MoE routing, and optimizer FLOPs are counted.

Training-Optimal Is Rarely Deployment-Optimal

Even with perfect FLOP accounting, Chinchilla's 20 tokens/parameter rule minimizes loss only for a fixed **training** FLOP budget. It assigns zero cost to serving the model after training finishes.

In real deployments serving millions of users, lifetime **inference compute** dwarfs training compute. Because inference cost scales linearly with active parameters per generated token ($\approx 2N_a$ FLOPs per token), paying extra training compute upfront to **overtrain** a smaller—or sparser—model dramatically

lowers total lifetime cost. Tracing the historical progression to 2026 frontier models shows how aggressively the field moved past Chinchilla-optimal training:

Era	Model (Released)	Tokens / Param (or Scale)	Regime
Historical	**GPT-3 175B** (2020)	2 tokens / param	Undertrained (Kaplan era)
Historical	**Chinchilla 70B** (2022)	20 tokens / param	Training-compute optimal
Historical	**LLaMA 65B** (2023)	22 tokens / param	Near Chinchilla
Historical	**Llama 2 70B** (2023)	29 tokens / param	Mildly overtrained
Historical	**Llama 3 70B** (July 2024)	215 tokens / param (15T tokens)	Heavily overtrained dense
2026 Frontier	**DeepSeek-V4.1-Flash** (Sept 2026)	**45T tokens**; 552B total, **16B active** decode (8B prefill)	Overtrained + highly sparse MoE

The higher your expected serving traffic, the further past Chinchilla-optimal you should push both training tokens and MoE sparsity.

WORKED EXAMPLE

Worked Example: Break-Even Inference Traffic for Overtraining

Using Besiroglu et al.'s surface $L(N, D) = 482N^{-0.35} + 2085D^{-0.37} + 1.82$, compare a Chinchilla-optimal dense 70B model trained on 1.4T tokens (20 tok/param) against a smaller, overtrained dense 30B model that matches the exact same validation loss. At what lifetime inference volume D_{inf} (using $C_{\text{train}} = 6ND$ and $C_{\text{inf}} = 2ND_{\text{inf}}$) does the overtrained 30B model become cheaper overall?

1. **Loss of the 70B Chinchilla model:**

$$L(70\text{B}, 1.4\text{T}) = \frac{482}{(70 \times 10^9)^{0.35}} + \frac{2085}{(1.4 \times 10^{12})^{0.37}} + 1.82 \approx 0.0771 + 0.0669 + 1.82 =$$

2. **Tokens D_{30} needed for a 30B model to match $L = 1.9640$:** The 30B parameter term is $482/(30 \times 10^9)^{0.35} \approx 0.1037$, leaving $1.9640 - 1.82 - 0.1037 = 0.0403$ for the data term:

$$\frac{2085}{D_{30}^{0.37}} = 0.0403 \implies D_{30} = \left(\frac{2085}{0.0403}\right)^{1/0.37} \approx 5.53\text{T tokens } (184 \text{ tok/param})$$

3. **Training compute comparison:**

$$C_{\text{train},70} = 6(70 \times 10^9)(1.4 \times 10^{12}) = 5.88 \times 10^{23} \text{ FLOPs}$$

$$C_{\text{train},30} = 6(30 \times 10^9)(5.53 \times 10^{12}) = 9.95 \times 10^{23} \text{ FLOPs}$$

The overtrained 30B model costs 4.07×10^{23} *extra* FLOPs to train.

4. **Break-even inference volume D_{inf}:** Each served token saves $2(70\text{B} - 30\text{B}) = 80 \times 10^9$ FLOPs:

$$D_{\text{inf}} = \frac{4.07 \times 10^{23}}{80 \times 10^9} \approx 5.09\text{T served tokens}$$

Interpretation: Once a deployment serves more than 5.09T inference tokens—a threshold quickly crossed by popular production models—the smaller overtrained 30B model matches the 70B model's quality at strictly lower total lifetime compute and more than $2\times$ faster per-token decode. Pairing overtraining with sparse MoE (such as DeepSeek-V4.1-Flash's 45T tokens and 16B active decode parameters) amplifies this serving advantage further.

IsoFLOPs Everywhere: Beyond Dense Autoregressive LMs

Finally, the IsoFLOP methodology introduced in Section 6 generalizes cleanly far beyond dense autoregressive Transformers. Whenever you compare two distinct modeling paradigms, comparing single arbitrary model sizes is misleading because one paradigm may simply be misallocated for its FLOP budget. Comparing their **IsoFLOP frontiers** eliminates that confound.

For example, Gulrajani et al. (2023) compared **Autoregressive vs. Diffusion Language Models** by sweeping IsoFLOP U-valleys for both families and plotting **Validation NLL** against **Non-Embedding FLOPs** ($10^{16} \to 10^{20}$) on log-log axes. Both paradigms form clean U-shaped IsoFLOP valleys whose minima trace **parallel straight power-law frontiers**: the **Autoregressive** line sits cleanly below the **Diffusion** line, separated by a constant horizontal dashed segment representing a **compute multiplier**—the exact factor of extra FLOPs Diffusion LMs need to match Autoregressive NLL at any scale. Likewise, as we saw in Section 4, Abnar et al. (2025) generalized IsoFLOP curves to 2D IsoFLOP surfaces that jointly optimize sparsity S and active/total parameters for sparse MoEs.

Bringing Scaling Laws Together in Practice

Across all three parts of this lecture, scaling laws turn frontier language model design from expensive guesswork into predictable small-scale science:

- **Data scaling:** Log-log power laws arise from polynomial estimation rates whose shallow slopes reflect high intrinsic data dimensionality; mixture composition shifts the offset, repetition saturates between ≈ 4 and 40 epochs, and filtering thresholds must loosen with compute.
- **Model and hyperparameter engineering:** Small-scale scaling sweeps predict architecture winners (watching for rank inversions), optimizer efficiency gains (AdamW $\to$ Muon), shape plateaus, critical batch size ($\mathcal{B}_{\text{crit}} \propto L^{-4.8}$), and learning-rate transfer ($\mu\text{P} \to$ CompleteP, Qiu et al., and HyperP).
- **Compute allocation:** Chinchilla's $N \propto C^{0.50}, D \propto C^{0.50}$ (≈ 20 tok/param) optimizes training FLOPs, whereas production deployments overtrain heavily and decouple N from N_a via sparse MoEs (such as DeepSeek-V4.1-Flash and Kimi K3).

- **Methodological hygiene:** State exact FLOP formulas (guarding against embedding exclusion, MoE reporting gaps, and Muon Newton–Schulz MFU inflation), recognize that scaling laws are recipe-specific bounds rather than universal ceilings, and verify loss-scaling predictions on downstream tasks.

KEY IDEA

1. The Kaplan–Chinchilla discrepancy ($a = 0.73$ vs. 0.50) is fully explained by Porian et al.'s three experimental factors (excluding last-layer FLOPs, excessive warmup, and untuned batch size/LR) together with Pearce & Song's proof that stripping embedding parameters inflates the local small-scale slope.
2. Always report wall-clock time alongside explicit FLOP definitions (especially with sparse MoEs and Muon's Newton–Schulz matmuls), compare modeling paradigms via their IsoFLOP frontiers, and overtrain smaller or sparser models whenever serving traffic dominates lifetime compute.

✎ Check Your Understanding

1. **Yes/No with justification:** If switching an optimizer from AdamW to Muon increases your cluster's reported MFU percentage by 10 points, does that automatically guarantee faster loss reduction per wall-clock second?
   Answer: No. Muon's Newton–Schulz matrix multiplications add dense matrix operations that inflate counted MFU without directly translating into faster loss reduction per wall-clock second, so you must compare loss versus wall-clock time with an explicitly defined FLOP formula.
2. **What would happen if…:** Two researchers evaluate the exact same set of Chinchilla-optimal small-to-medium models (1K–1.5B parameters), where Researcher A counts total parameters N_T and total FLOPs C_T, while Researcher B excludes vocabulary embeddings ($N_{\setminus E}$ and $C_{\setminus E}$). How will their fitted scaling exponents a compare?
   Answer: Researcher A will recover the true Chinchilla slope near $a \approx 0.50$–0.51, whereas Researcher B will fit a much steeper local slope near $a \approx 0.78$ because embeddings dominate total parameters at small scales and shrink as a fraction at larger scales.