CS 395T: Foundations of Modern Generative AI Fall 2026

Lecture 5: Mixture-of-Experts and Scaling Laws

Prof. Noah Golowich

Today we will finish up our discussion of architecture, discussing Mixture-of-experts transformers; then we will move on to scaling laws.

1 Mixture-of-experts transformers

A dense transformer applies the same MLP to every token at a given layer. A *mixture-of-experts* (MoE) layer instead maintains many MLPs and selects a small subset for each token. This lets us increase the number of learned parameters while controlling the amount of computation performed on each token.

1.1 Architecture and examples

Replacing the position-wise MLP. Let d be the model dimension, T the sequence length, and L the number of transformer layers. Write $X^{(r)} = [x_1^{(r)}, \dots, x_T^{(r)}] \in \mathbb{R}^{d \times T}$ for the representations after layer $r \in [L]$, with $X^{(0)}$ denoting the input embeddings. The attention sublayer computes

$$U^{(r)} = X^{(r-1)} + \mathrm{MHA}_r\Big(\mathrm{Norm}_{\mathrm{attn}}^{(r)}(X^{(r-1)}); M_{\mathrm{causal}}\Big), \tag{1}$$

where MHA_r is multi-head attention, normalization acts on each column, and M_{causal} is the mask with entry (j,t) equal to $\mathbf{1}\{j \le t\}$. Write $u_t^{(r)}$ for column t of $U^{(r)}$, and define the normalized MLP input

$$\widetilde{u}_t^{(r)} := \mathrm{Norm}_{\mathrm{mlp}}^{(r)}(u_t^{(r)}) \in \mathbb{R}^d.$$

A dense layer would set $x_t^{(r)} = u_t^{(r)} + \mathrm{MLP}_r(\widetilde{u}_t^{(r)})$. In an MoE layer, we replace MLP_r with N separately parameterized MLPs $\mathrm{MLP}_{r,1}, \dots, \mathrm{MLP}_{r,N} : \mathbb{R}^d \to \mathbb{R}^d$, called *experts*.

The router. A learned *router* scores the experts for each token. Let $W_R^{(r)} \in \mathbb{R}^{N \times d}$ be its weight matrix. The vector of scores before softmax, called the *router logits*, and the corresponding probabilities are

$$z_t^{(r)} := W_R^{(r)} \widetilde{u}_t^{(r)}, \qquad p_{i,t}^{(r)} := \frac{\exp(z_{i,t}^{(r)})}{\sum_{j=1}^N \exp(z_{j,t}^{(r)})}, \quad i \in [N]. \tag{2}$$

Thus the softmax is over experts, separately for each token.

Fix an integer $K \in \{1, \dots, N\}$, the number of experts selected for each token at this layer. Let $S_t^{(r)} \subseteq [N]$ be the indices of the K largest entries of $p_t^{(r)}$, breaking ties by a fixed rule. Define the sparse gate weights by

$$g_{i,t}^{(r)} := p_{i,t}^{(r)} \mathbf{1}\{i \in S_t^{(r)}\}. \tag{3}$$

The MoE sublayer then computes

$$x_t^{(r)} = u_t^{(r)} + \sum_{i \in S_t^{(r)}} g_{i,t}^{(r)} \, \mathrm{MLP}_{r,i}(\widetilde{u}_t^{(r)}). \tag{4}$$

The selected subset can change with both the token and the layer. The attention operation still mixes information across positions; expert selection determines which MLPs transform each resulting token representation. One can replace all dense MLPs or only those in selected layers.

Note that the selected probabilities are *not renormalized*, so their sum can be less than one. Another convention, used by Mixtral [11, Section 2.1], divides $g_{i,t}^{(r)}$ by $\sum_{j \in S_t^{(r)}} p_{j,t}^{(r)}$, making the selected weights sum to one.

Parameters and computation. For e.g., SwiGLU experts, the expert weights contain $3Ndd_{\text{ff}}$ parameters, but only $3Kdd_{\text{ff}}$ of these parameters participate in the expert matrix multiplications for one token. Evaluating these matrix multiplications costs $O(Kdd_{\text{ff}})$ arithmetic operations, in addition to $O(Nd)$ for router logits and the cost of selecting the top K scores. The fraction K/N describes the *active fraction* of experts.

Shared experts and model sizes. Some architectures additionally use N_s *shared experts*, which process every token regardless of the router. Their outputs are added to the routed expert output. In this convention, N counts only routed experts, so the layer has $N + N_s$ experts and evaluates $K + N_s$ per token. For instance, DeepSeekMoE 16B has 64 routed and 2 shared experts, and evaluates $6 + 2 = 8$ experts per token [8, Section 5.1.2].

Table 1 gives representative choices, including frontier open-weight releases available as of September 8, 2026.

Model / experiment	Routed N	K	Shared N_s	Source
Kimi K3	896	16	2	Model card [13]
Qwen3.8-2.4T-A95B	512	10	1	Model card [14]
DeepSeek-V4-Pro	384	6	1	Report [12]
DeepSeek-V4-Flash	256	6	1	Report [12]
GLM-5.3	256	8	1	Config. [15]
Kimi K2.5	384	8	1	Model card [16]
DeepSeek-V3.2	256	8	1	Config. [17]
DeepSeekMoE 16B	64	6	2	[8, Sec. 5.1.2]
Mixtral 8×7B	8	2	0	[11, Sec. 2]
Switch-Base / Large	128	1	0	[9, Table 9]
Switch-XXL / Switch-C	64 / 2048	1	0	[9, Table 9]
Shazeer et al. (2017), flat LM	4, 32, 256	4	0	[10, App. C.1]
Shazeer et al. (2017), hierarchical LM	up to 131072	4*	0	[10, Apps. C–D]
Shazeer et al. (2017), multilingual	512	2	0	[10, App. E]

Table 1: Expert counts and per-token selection. *Hierarchical routing selects two groups and two experts within each selected group, evaluating four leaf experts.

1.2 Load balancing

If most tokens select the same few experts, other experts receive little training, and the devices hosting the popular experts become computation bottlenecks. Thus it is common to add an auxiliary loss that penalizes assigning high router probability to experts already receiving a large share of the tokens [9, Section 2.2].

Hard loads and soft probabilities. Fix one MoE layer and first specialize to $K = 1$. Let B be the number of tokens in the batch used to compute the balancing loss; the batch may contain multiple sequences. Suppress the layer index and enumerate its normalized token representations as $v_1, \dots, v_B \in \mathbb{R}^d$. Set $z_t = W_R v_t$ and $p_t = \mathrm{softmax}(z_t)$ as in (2). Let $a_t \in [N]$ be the index maximizing $p_{i,t}$, with the same fixed tie-breaking rule as before. Define

$$f_i := \frac{1}{B} \sum_{t=1}^{B} \mathbf{1}\{a_t = i\}, \qquad P_i := \frac{1}{B} \sum_{t=1}^{B} p_{i,t}, \qquad i \in [N]. \tag{5}$$

Here f_i is the fraction of hard assignments to expert i, whereas P_i is its average soft router probability. Both vectors sum to one, but generally $f \neq P$: a probability of 0.51 and one of 0.99 both yield the same top-1 decision in a two-expert layer.

For a coefficient $\alpha > 0$, the Switch auxiliary loss is

$$\mathcal{L}_{\mathrm{bal}} := \alpha N \sum_{i=1}^{N} f_i P_i. \tag{6}$$

During training, this term is added for each MoE layer to the language-modeling loss.

When does equal load minimize the loss? New we observe the straightforward fact that (6) is minimized at uniform loads, subject to $P = f$:

Lemma 1.1 (Minimum when hard and soft loads agree)**.** *Let $N \geq 2$ and $\alpha > 0$. For nonnegative vectors $f, P \in \mathbb{R}^N$ with $\sum_i f_i = \sum_i P_i = 1$ and $P = f$, the Switch loss satisfies*

$$\mathcal{L}_{\mathrm{bal}} = \alpha + \alpha N \sum_{i=1}^{N} \left(f_i - \frac{1}{N} \right)^2 \geq \alpha, \tag{7}$$

with equality if and only if $f_i = P_i = 1/N$ for every i. Consequently, its minimum over these pairs is attained exactly at equal load.

Proof. Substituting $P = f$ into (6) gives $\mathcal{L}_{\mathrm{bal}} = \alpha N \sum_i f_i^2$. Since $\sum_i f_i = 1$,

$$\sum_{i=1}^{N} \left(f_i - \frac{1}{N} \right)^2 = \sum_{i=1}^{N} f_i^2 - \frac{2}{N} \sum_{i=1}^{N} f_i + \frac{N}{N^2} = \sum_{i=1}^{N} f_i^2 - \frac{1}{N}.$$

Rearranging yields (7). Each square is nonnegative, and their sum vanishes exactly when every $f_i = 1/N$. This vector, together with $P = f$, is feasible, proving the minimum and its equality characterization. □

Computing the actual router gradient. The hard assignments are treated as constants in backpropagation. Away from ties they are locally constant, so this also gives the ordinary derivative within a region where the assignments do not change.

First, the derivative with respect to a token's router probability is

$$\frac{\partial \mathcal{L}_{\mathrm{bal}}}{\partial p_{i,t}} = \frac{\alpha N}{B} f_i. \tag{8}$$

For indices $i, j \in [N]$, the quotient rule gives

$$\frac{\partial p_{i,t}}{\partial z_{j,t}} = \frac{\mathbf{1}\{i = j\} \exp(z_{i,t}) \sum_{k=1}^{N} \exp(z_{k,t}) - \exp(z_{i,t}) \exp(z_{j,t})}{\left(\sum_{k=1}^{N} \exp(z_{k,t}) \right)^2}$$

$$= p_{i,t}(\mathbf{1}\{i = j\} - p_{j,t}).$$

Define $\mu_t := \sum_{i=1}^{N} f_i p_{i,t}$, the average expert load when an expert is sampled from the soft distribution p_t. The chain rule now yields

$$\frac{\partial \mathcal{L}_{\text{bal}}}{\partial z_{j,t}} = \frac{\alpha N}{B} \sum_{i=1}^{N} f_i p_{i,t}(\mathbf{1}\{i = j\} - p_{j,t}) = \frac{\alpha N}{B} p_{j,t}(f_j - \mu_t). \tag{9}$$

For completeness, let $w_j^\top$ be row j of W_R. Since $z_{j,t} = w_j^\top v_t$, the contribution to the router-weight gradient, holding the layer inputs fixed, is

$$\nabla_{w_j} \mathcal{L}_{\text{bal}} = \frac{\alpha N}{B} \sum_{t=1}^{B} p_{j,t}(f_j - \mu_t) v_t. \tag{10}$$

Why this gradient promotes balance. An infinitesimal gradient-descent update on the logits decreases $z_{j,t}$ when $f_j > \mu_t$ and increases it when $f_j < \mu_t$. Thus, for each token, the balancing term pushes down scores of experts whose load exceeds that token's probability-weighted average, and pushes up scores of experts below that average. In particular, a maximally loaded expert has a nonnegative logit derivative, and a minimally loaded expert has a nonpositive one.

One can also see the balancing incentive directly in probability space. Hold f fixed and move an amount $\delta > 0$ of probability for one token from expert i to expert j, choosing δ so the resulting probabilities remain nonnegative. If $f_i > f_j$, the exact loss change is

$$\Delta \mathcal{L}_{\text{bal}} = \frac{\alpha N}{B} \delta (f_j - f_i) < 0.$$

The loss therefore rewards moving probability away from a more heavily loaded expert.

Extension to top-K routing. For $K > 1$, let S_t contain the K selected routed experts for token t. A normalized version of the same loss uses

$$f_i^{(K)} := \frac{1}{BK} \sum_{t=1}^{B} \mathbf{1}\{i \in S_t\}, \qquad P_i := \frac{1}{B} \sum_{t=1}^{B} p_{i,t}, \qquad \mathcal{L}_{\text{bal}}^{(K)} := \alpha N \sum_{i=1}^{N} f_i^{(K)} P_i.$$

Since $\sum_i f_i^{(K)} = 1$, equal load means $f_i^{(K)} = 1/N$, or BK/N assignments per expert. Holding the selected sets fixed gives exactly (9) with $f^{(K)}$ in place of f.

2 Scaling laws

Scaling laws describe how test loss changes as we increase the training data, model size, or computation. A common empirical pattern is a power-law decrease toward a nonzero limiting loss.

2.1 An early example

Cortes et al. [1] observed an early example of power-law dependence between the number of training examples and test loss. For a classifier trained on D examples, they modeled the test error as $L_{\text{test}}(D) \approx L_\infty + bD^{-q}$, with $b, q > 0$. Thus it is the error *above its limiting value* that decays as a power. Figure 1 shows their measurements and extrapolated learning curves.

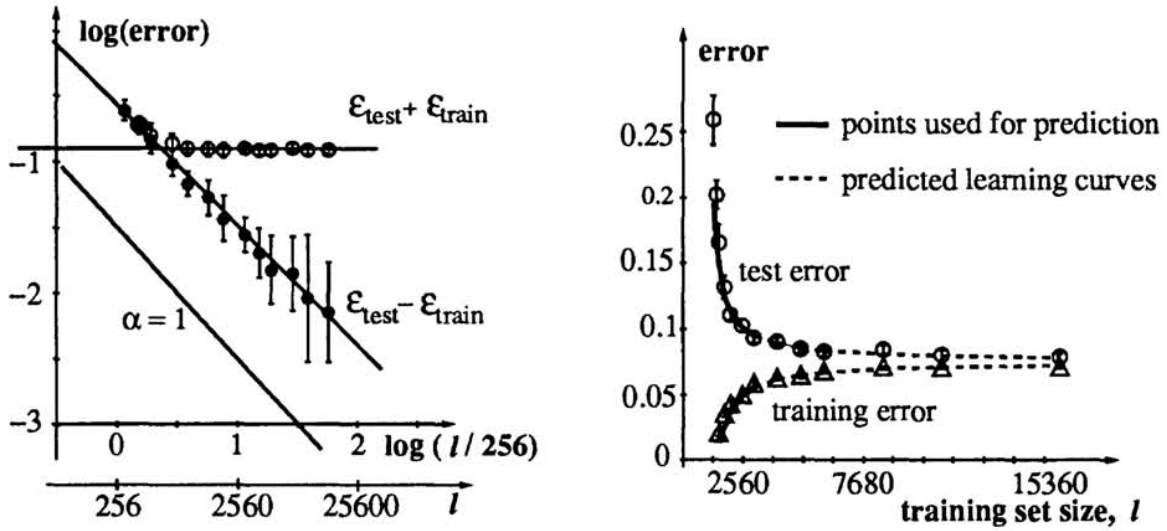

Figure 1: Figure 4 of [1]. Left: sums and differences of test and training errors on log-log axes. Right: measured and predicted learning curves.

2.2 Simple intuitions for scaling laws

Power laws already arise in elementary estimation and approximation problems.

Mean estimation. Let $x_1, \dots, x_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, and estimate μ by $\widehat{\mu} = n^{-1} \sum_{i=1}^{n} x_i$. Independence gives

$$\mathbb{E}[(\widehat{\mu} - \mu)^2] = \mathrm{Var}(\widehat{\mu}) = \frac{1}{n^2} \sum_{i=1}^{n} \mathrm{Var}(x_i) = \frac{\sigma^2}{n}.$$

This is a scaling law with exponent 1 for *mean squared* ℓ_2 error; the root mean squared error is $\sigma/\sqrt{n}$.

Approximating a function on a grid. Next we give a toy example from [2] which describes why loss might decay as a power law in *model size (i.e., number of parameters)*. Let $f : [0,1]^d \to \mathbb{R}$ be K-Lipschitz. Partition the domain into cubes of side length s and approximate f by its value at the center of each cube. There are $P = s^{-d}$ cubes and one fitted constant per cube. Every point is within $\sqrt{d}\, s/2$ of its cube's center, so the squared prediction loss under a uniform input is

$$L(P) := \int_{[0,1]^d} |f(x) - \widehat{f}(x)|^2 \, dx \le \frac{K^2 d}{4} s^2 = \frac{K^2 d}{4} P^{-2/d}.$$

If f has bounded second derivatives, use its affine Taylor approximation on each cube instead. Taylor's remainder is $O(s^2)$, hence the squared loss is $O(s^4)$. There are now $d+1$ coefficients per cube, giving

$$P = (d+1)s^{-d}, \qquad L(P) = O(P^{-4/d}) \quad \text{for fixed } d.$$

Note that we assume unlimited data. Moreover, these are only upper bounds, i.e., equality need not hold for every f. To summarize: error on each cell decreases polynomially with cell size, while the number of cells grows as s^{-d}.

2.3 Allocating parameters and data under a compute budget

Let P be the number of parameters in a dense language model and D the number of training tokens. Hoffmann et al. [3] propose the model

$$L(D, P) = E + \frac{A}{P^\alpha} + \frac{B}{D^\beta}, \qquad A, B, \alpha, \beta > 0. \tag{11}$$

Here we use P in place of their parameter-count notation N.

Interpreting the three terms. Let $\mathcal{R}(f)$ denote the population next-token cross-entropy of a predictor f. Write $f_\star$ for the Bayes predictor, $f_P^\star$ for the best predictor in the P-parameter model class, and $\widehat{f}_{D,P}$ for the output of training on D tokens. It is typicaly to decompose the population loss at $\widehat{f}_{D,P}$ as:

$$\mathcal{R}(\widehat{f}_{D,P}) = \underbrace{\mathcal{R}(f_\star)}_{\text{irreducible loss}} + \underbrace{\mathcal{R}(f_P^\star) - \mathcal{R}(f_\star)}_{\text{function approximation}} + \underbrace{\mathcal{R}(\widehat{f}_{D,P}) - \mathcal{R}(f_P^\star)}_{\text{finite-data training}}. \tag{12}$$

The first term, modeled by E, is the conditional entropy of the next token given its context. The second, modeled by $AP^{-\alpha}$, measures the limitation of the model class even with ideal training. The third, modeled by $BD^{-\beta}$, includes both using a finite sample and taking only finitely many optimization steps (typically one pass).

One might expect that $\alpha \approx 1/2$ using shallow-network approximation results of Siegel and Xu [4], and $\beta \approx 1/2$ using classical stochastic approximation [5] and the $D^{-1/2}$ rates of stochastic convex optimization [6]. As we shall see, the actual constants are somewhat worse: $\alpha \approx 0.34$ and $\beta \approx 0.28$.

The compute constraint. For a fixed compute budget C, measured in floating-point operations (FLOPs), we seek

$$(P_{\text{opt}}(C), D_{\text{opt}}(C)) \in \operatorname*{arg\,min}_{P,D:\, 6PD \le C} L(D, P).$$

The approximation $C \approx 6PD$ counts roughly $2P$ FLOPs per token for the forward pass (one multiplication and one addition per weight), and $4P$ for backpropagation (computing activation gradients and weight gradients). This counts the dominant dense matrix operations; attention and other overhead can change the cost, particularly at long context lengths.

To find the optimal tradeoff between P and D, we set $D = C/(6P)$. Differentiating $AP^{-\alpha} + B(6P/C)^\beta$ gives the balance condition $\alpha A P^{-\alpha} = \beta B D^{-\beta}$. Consequently, we obtain

$$\begin{aligned} P_{\text{opt}}(C) &= G(C/6)^{\beta/(\alpha+\beta)}, \\ D_{\text{opt}}(C) &= G^{-1}(C/6)^{\alpha/(\alpha+\beta)}, \end{aligned} \qquad G = \left(\frac{\alpha A}{\beta B}\right)^{1/(\alpha+\beta)}. \tag{13}$$

In particular, having equal loss exponents, i.e., $\alpha = \beta$, predicts $P_{\text{opt}}, D_{\text{opt}} \propto C^{1/2}$.

Three ways to estimate the allocation. Hoffmann et al. [3] compare three empirical approaches:

1. **Envelope of training curves.** Fix several model sizes and vary training length, with learning-rate schedules matched to each length. At each compute budget, take the lowest loss across the training curves, then fit power laws to the parameter and token counts attaining this envelope (Figure 2).
2. **IsoFLOP profiles.** Fix a compute budget and train different model sizes with token counts $D \approx C/(6P)$. Fit a parabola to final loss versus log model size, find its minimum, and repeat across budgets to fit the optimal allocation (Figure 3).

3. **Parametric loss fit.** Fit E, A, B, α, β in (11) to final losses across model sizes and training lengths (using a robust loss on log losses), then apply (13) (Figure 4).

The resulting exponents for $(P_{\text{opt}}, D_{\text{opt}})$ are approximately $(0.50, 0.50)$, $(0.49, 0.51)$, and $(0.46, 0.54)$, respectively: parameters and training tokens should grow at roughly equal rates as compute increases.

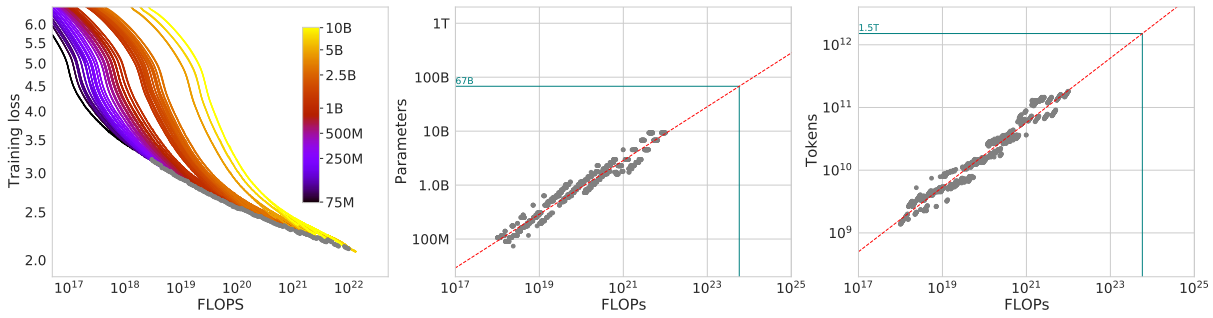

Figure 2: Approach 1: training-curve envelope (Figure 2 of [3]). Left: loss versus compute for different model sizes. Center and right: optimal parameter and token counts versus compute.

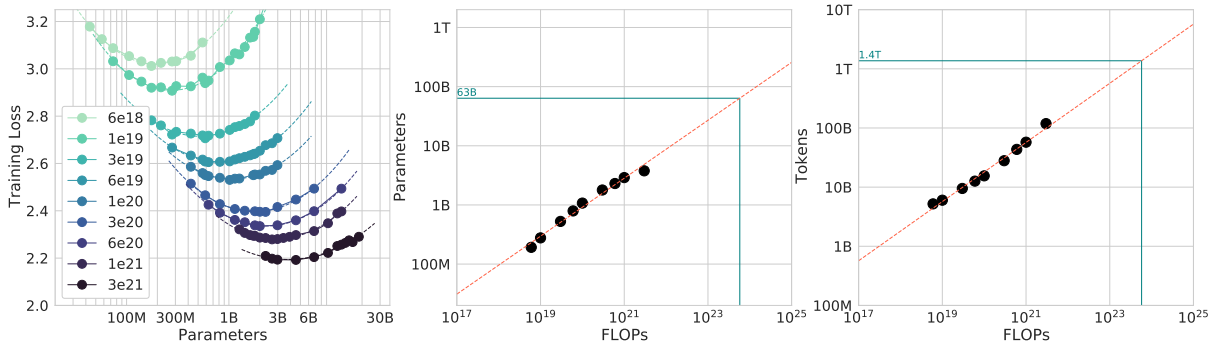

Figure 3: Approach 2: IsoFLOP profiles (Figure 3 of [3]). Left: loss versus model size at fixed compute budgets. Center and right: the allocations inferred from the minima.

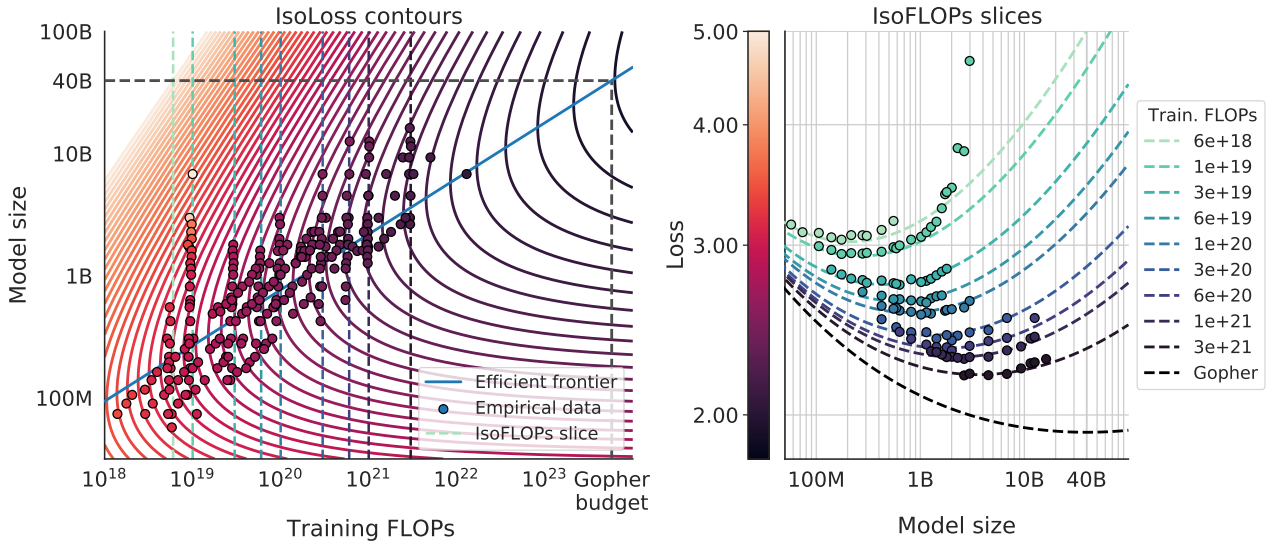

Figure 4: Approach 3: parametric loss fit (Figure 4 of [3]). Left: fitted equal-loss contours and the compute-efficient frontier in blue. Right: fitted loss curves and measurements at fixed compute budgets.

2.4 A scaling law for linear regression

In this section we discuss results of Lin et al. [7], who prove a joint parameter–data scaling law for linear regression trained by one-pass SGD. The exponents come from the decay of the data's covariance eigenvalues. We use P for the number of trainable parameters and D for the sample count, as above.

Data and model. Assume the input covariance is diagonal: for a fixed $a > 1$, let

$$x \sim \mathcal{N}(0, H), \qquad H = \operatorname{diag}(\lambda_1, \lambda_2, \ldots), \qquad \lambda_j \asymp j^{-a}.$$

We index the diagonal entries in decreasing order. The coordinates $x_j \sim \mathcal{N}(0, \lambda_j)$ are independent. Here $\asymp$ denotes upper and lower bounds up to positive constant factors. Draw a target $w_\star = (w_{\star,j})_{j \geq 1}$ with isotropic second moments, $\mathbb{E}[w_{\star,i} w_{\star,j}] = \mathbf{1}\{i = j\}$, independently of the covariates; independent standard Gaussian coordinates are one example. Labels are

$$y = \sum_{j \geq 1} x_j w_{\star,j} + \varepsilon, \qquad \mathbb{E}[\varepsilon \mid x, w_\star] = 0, \qquad \mathbb{E}[\varepsilon^2 \mid x, w_\star] = 1.$$

Although an isotropic $w_\star$ need not have finite Euclidean norm, its predictions are well defined because $\sum_j \lambda_j < \infty$.

To obtain a P-parameter model, draw a fixed random sketch S with P rows and infinitely many columns whose entries are independent $\mathcal{N}(0, 1/P)$ variables, independently of the target and data. The model observes $Sx \in \mathbb{R}^P$ and predicts $v^\top Sx$, where $v \in \mathbb{R}^P$ is trainable. The sketch is fixed throughout training and is not counted among the trainable parameters. Define its population risk by

$$\mathcal{L}_P(v) := \mathbb{E}_{x,y}[(v^\top Sx - y)^2 \mid w_\star, S].$$

One-pass SGD. Starting from $v_0 = 0$, use D independent training examples and output the last iterate:

$$v_t = v_{t-1} - \gamma_t (v_{t-1}^\top S x_t - y_t) S x_t, \qquad \gamma_t = \frac{\gamma}{2^{\lfloor t / D_{\text{eff}} \rfloor}}, \qquad D_{\text{eff}} := \frac{D}{\log_2 D}. \tag{14}$$

Thus the step size halves roughly every D_{eff} examples.

Theorem 2.1 ([7], Theorem 4.1)**.** *Under the setup above, let $D \geq 2$ and $P \geq 1$. For a sufficiently small initial step size $0 < \gamma \leq c$ and $\gamma D_{\text{eff}} \geq 1$, with probability at least $1 - \exp(-\Omega(P))$ over S, the output of (14) satisfies*

$$\mathbb{E}[\mathcal{L}_P(v_D) \mid S] = 1 + \Theta\Big(P^{-(a-1)} + (\gamma D_{\text{eff}})^{-(a-1)/a}\Big). \tag{15}$$

The expectation averages over the target and training data. In particular, the parameter exponent is $a - 1$ and the data exponent is $(a-1)/a$, up to the logarithmic factor in D_{eff}.

Proof sketch: how the exponents arise. The model's coefficient vector in the original input coordinates is $S^\top v$. Since H is diagonal and the label noise has conditional mean zero and variance one, the population loss is

$$\mathcal{L}_P(v) = 1 + \mathbb{E}_x\left[\left(\sum_{j \geq 1} x_j((S^\top v)_j - w_{\star,j})\right)^2 \,\middle|\, w_\star, S\right] = 1 + \sum_{j \geq 1} \lambda_j ((S^\top v)_j - w_{\star,j})^2.$$

Thus excess loss is the sum of squared coefficient errors, weighted by λ_j. We treat the sketch S as if it retained the leading covariance directions.

The parameter exponent. Heuristically, suppose the predictor learns the first j coordinates of the target and leaves the remaining coordinates unlearned:

$$(S^\top v)_i \approx w_{\star,i} \quad (i \le j), \qquad |(S^\top v)_i - w_{\star,i}| \approx \Theta(1) \quad (i > j).$$

Thus the coefficient error is approximately zero for $i \le j$ and has order-one magnitude for $i > j$. Here "order one" refers to its expected square, since $\mathbb{E}[w_{\star,i}^2] = 1$. Substituting into the population-loss formula gives

$$\mathcal{L}_P(v) - 1 \approx \sum_{i>j} \lambda_i w_{\star,i}^2, \qquad \mathbb{E}[\mathcal{L}_P(v)] - 1 \approx \sum_{i>j} \lambda_i \asymp j^{-(a-1)}.$$

The last step uses $\lambda_i \asymp i^{-a}$ and $\int_j^\infty u^{-a}\,du = j^{-(a-1)}/(a-1)$. A P-parameter sketch can fit on the order of P leading coordinates, so taking $j \asymp P$ gives the parameter exponent $a - 1$.

The data exponent. We use the heuristic/simplification that the rows of S consist of the first P basis vectors. In this idealization, $v_{t,i}$ is the learned coefficient of x_i, whose variance is λ_i. Its SGD update is

$$v_{t,i} = v_{t-1,i} - \gamma_t \left(\sum_{\ell=1}^{P} v_{t-1,\ell} x_{t,\ell} - y_t \right) x_{t,i}.$$

Fix $i \le P$ and condition on $w_\star$ and the previous training samples. Then v_{t-1} is fixed, and only the new sample (x_t, y_t) is random. Substitute $y_t = \sum_{\ell \ge 1} w_{\star,\ell} x_{t,\ell} + \varepsilon_t$ into the prediction error:

$$\sum_{\ell=1}^{P} v_{t-1,\ell} x_{t,\ell} - y_t = \sum_{\ell=1}^{P} (v_{t-1,\ell} - w_{\star,\ell}) x_{t,\ell} - \sum_{\ell>P} w_{\star,\ell} x_{t,\ell} - \varepsilon_t.$$

Multiply by $x_{t,i}$ and average over the new sample. On the following lines, all expectations condition on the fixed target and previous samples:

$$\begin{aligned}
&\mathbb{E}\left[\left(\sum_{\ell=1}^{P} v_{t-1,\ell} x_{t,\ell} - y_t \right) x_{t,i} \right] \\
&\quad = \sum_{\ell=1}^{P} (v_{t-1,\ell} - w_{\star,\ell}) \mathbb{E}[x_{t,\ell} x_{t,i}] - \sum_{\ell>P} w_{\star,\ell} \mathbb{E}[x_{t,\ell} x_{t,i}] - \mathbb{E}[\varepsilon_t x_{t,i}] \\
&\quad = \lambda_i (v_{t-1,i} - w_{\star,i}).
\end{aligned}$$

Indeed, independent centered input coordinates give $\mathbb{E}[x_{t,\ell} x_{t,i}] = 0$ for $\ell \ne i$, while $\mathbb{E}[x_{t,i}^2] = \lambda_i$. Thus only the $\ell = i$ term in the first sum survives; the entire second sum vanishes since $i \le P$. The noise term vanishes by conditioning first on x_t and using $\mathbb{E}[\varepsilon_t \mid x_t, w_\star] = 0$. Finally, subtract $w_{\star,i}$ from the SGD update and take its conditional expectation. Since $v_{t-1,i} - w_{\star,i}$ is fixed under this conditioning,

$$\begin{aligned}
\mathbb{E}[v_{t,i} - w_{\star,i} \mid w_\star, \text{previous samples}] &= v_{t-1,i} - w_{\star,i} - \gamma_t \lambda_i (v_{t-1,i} - w_{\star,i}) \\
&= (1 - \gamma_t \lambda_i)(v_{t-1,i} - w_{\star,i}).
\end{aligned}$$

Thus one step multiplies the mean coefficient error by $1 - \gamma_t \lambda_i$. Averaging over the previous samples and iterating, using $v_{0,i} = 0$, gives

$$\mathbb{E}[v_{D,i} - w_{\star,i} \mid w_\star] = -w_{\star,i} \prod_{t=1}^{D} (1 - \gamma_t \lambda_i).$$

Here the mean is over the training samples, with the target fixed. For small $\gamma_t \lambda_i$, the approximation $\log(1-u) \approx -u$ gives

$$\prod_{t=1}^{D} (1 - \gamma_t \lambda_i) \approx \exp\left(-\lambda_i \sum_{t=1}^{D} \gamma_t \right) = \exp(-\Theta(\gamma D_{\text{eff}} \lambda_i)),$$

where the last step uses $\sum_t \gamma_t \asymp \gamma D_{\text{eff}}$. A direction with larger variance therefore has a stronger expected gradient and is learned faster. Thus directions are substantially learned only when $\gamma D_{\text{eff}} \lambda_j \gtrsim 1$. For $\lambda_j \asymp j^{-a}$, the number of such directions, denoted by k, is $k \asymp (\gamma D_{\text{eff}})^{1/a}$. Combining the parameter and training limits gives an effective cutoff of $\min\{P, k\}$ directions, and hence error of order

$$\sum_{j>k} \lambda_j \asymp (k)^{-(a-1)} \asymp (\gamma D_{\text{eff}})^{-(a-1)/a}.$$

This explains the data exponent $(a-1)/a$.

Why variance does not spoil the scaling law. The preceding calculation controls the mean coefficient error, but the loss depends on the mean *squared* error. For each retained coordinate $i \le P$, these differ by the variance over the training samples:

$$\mathbb{E}[(v_{D,i} - w_{\star,i})^2 \mid w_\star] = \left(\mathbb{E}[v_{D,i} \mid w_\star] - w_{\star,i}\right)^2 + \mathrm{Var}(v_{D,i} \mid w_\star).$$

We must therefore bound the sum of these variances weighted by λ_i, and then averaged over the target.

We continue on assuming for simplicity that S selects the first P coordinates. At step t, label noise adds $\gamma_t \varepsilon_t x_{t,i}$ to coefficient i. Its expected square is $\gamma_t^2 \lambda_i$, since the conditional variance of the noise ε_t is one. Its contribution to prediction loss has one more factor of λ_i, so it starts at size $\gamma_t^2 \lambda_i^2$. Later updates contract this perturbation, by an argument similarly to the one above; in particular, the variance error is damped roughly by $\prod_{s>t}(1-\gamma_s\lambda_i)^2$. This heuristic argument allows us to bound the impact of variance on the error in estimating coordinate $i \le P$ as:

$$\lambda_i \mathbb{E}_{w_\star}[\mathrm{Var}(v_{D,i} \mid w_\star)] \le C \sum_{t=1}^{D} \gamma_t^2 \lambda_i^2 \exp\left(-\lambda_i \sum_{s=t+1}^{D} \gamma_s\right) \le \frac{C}{D_{\text{eff}}} \min\{1, (\gamma D_{\text{eff}} \lambda_i)^2\}.$$

The last inequality uses Lemma A.1. The schedule balances two effects: noise introduced early is damped by the remaining updates, while noise introduced late is small because the step size has decreased.

Recall that $k \asymp (\gamma D_{\text{eff}})^{1/a}$ is the number of directions that training can substantially learn. The variance bound contributes at most C/D_{eff} for each $i \le k$. For $i > k$, it is smaller by a factor of order $(k/i)^{2a}$. These remaining terms sum to $O(k/D_{\text{eff}})$ as well, since

$$\sum_{i>k} (k/i)^{2a} \le k^{2a} \int_k^\infty u^{-2a}\, du = \frac{k}{2a-1}.$$

There are only P retained directions, each contributing at most C/D_{eff}. Thus the total stochastic contribution is at most

$$C \frac{\min\{P, (\gamma D_{\text{eff}})^{1/a}\}}{D_{\text{eff}}} \le C\gamma (\gamma D_{\text{eff}})^{-(a-1)/a}.$$

For bounded γ, this is no larger in order than the data-dependent error already obtained. Thus stochastic fluctuations do not change the scaling exponent.

References

[1] Corinna Cortes, L. D. Jackel, Sara A. Solla, Vladimir Vapnik, and John S. Denker. Learning curves: Asymptotic values and rate of convergence. In *Advances in Neural Information Processing Systems 6*, pp. 327–334, 1993 conference. Paper.

[2] Utkarsh Sharma and Jared Kaplan. Scaling laws from the data manifold dimension. *Journal of Machine Learning Research*, 23(9):1–34, 2022. Paper.

[3] Jordan Hoffmann et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems 35*, 2022. arXiv.

[4] Jonathan W. Siegel and Jinchao Xu. Approximation rates for neural networks with general activation functions. *Neural Networks*, 128:313–321, 2020. arXiv.

[5] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[6] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3–4):231–357, 2015. arXiv.

[7] Licong Lin, Jingfeng Wu, Sham M. Kakade, Peter L. Bartlett, and Jason D. Lee. Scaling laws in linear regression: Compute, parameters, and data. arXiv:2406.08466, 2024; version 3, June 2025. arXiv.

[8] Damai Dai et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. arXiv:2401.06066, 2024. arXiv.

[9] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. arXiv:2101.03961, 2021. arXiv.

[10] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. arXiv.

[11] Albert Q. Jiang et al. Mixtral of experts. arXiv:2401.04088, 2024. arXiv.

[12] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report, 2026, Sections 2.1 and 4.2.1. arXiv.

[13] Moonshot AI. Kimi K3: Official model card, Section 2 (Model Summary). 2026; accessed September 8, 2026. `https://huggingface.co/moonshotai/Kimi-K3`.

[14] Qwen Team. Qwen3.8-2.4T-A95B: Official model card, Model Overview. 2026; accessed September 8, 2026. `https://huggingface.co/Qwen/Qwen3.8-2.4T-A95B`.

[15] Z.ai. GLM-5.3: Official model configuration. 2026; accessed September 8, 2026. Fields `n_routed_experts`, `num_experts_per_tok`, and `n_shared_experts`. `https://huggingface.co/zai-org/GLM-5.3/blob/main/config.json`.

[16] Moonshot AI. Kimi K2.5: Official model card, Section 2 (Model Summary). 2026; accessed September 8, 2026. `https://huggingface.co/moonshotai/Kimi-K2.5`.

[17] DeepSeek-AI. DeepSeek-V3.2: Official model configuration. Accessed September 8, 2026. Fields `n_routed_experts`, `num_experts_per_tok`, and `n_shared_experts`. `https://huggingface.co/deepseek-ai/DeepSeek-V3.2/blob/main/config.json`.

AI Use. I prepared these lecture notes with the aid of AI assistance; I take full responsibility for all content in the notes.

A Geometric step-size schedule

The following estimate controls the contribution of noise introduced at each SGD step. The constants $c, C > 0$ are absolute and may change from line to line.

Lemma A.1 (Geometric schedule)**.** *The schedule in* (14) *satisfies* $c\gamma D_{\text{eff}} \le \sum_{t=1}^{D} \gamma_t \le C\gamma D_{\text{eff}}$*. For any scalar* $\mu > 0$ *with* $\gamma\mu \le 1/4$*,*

$$\sum_{t=1}^{D} \gamma_t^2 \mu^2 \exp\left(-\mu \sum_{s=t+1}^{D} \gamma_s\right) \le \frac{C}{D_{\text{eff}}} \min\{1, (\gamma D_{\text{eff}}\mu)^2\}. \tag{16}$$

Proof. Group steps by $\ell = \lfloor t/D_{\text{eff}} \rfloor$. The groups are indexed by $0, \dots, \lfloor \log_2 D \rfloor$, with step size $\gamma 2^{-\ell}$ in group ℓ. Each group has at most $2D_{\text{eff}}$ steps; every group before the last has at least $D_{\text{eff}}/3$ steps. These bounds cover the first group starting at $t = 1$, since $D_{\text{eff}} > 1$. The first group gives the lower bound on total step size, and summing $2^{-\ell}$ gives the upper bound.

For $\ell \le \lfloor \log_2 D \rfloor - 2$, the complete next group contributes at least $\gamma D_{\text{eff}} 2^{-\ell}/6$ to the remaining step-size sum. Thus group ℓ contributes at most

$$\frac{2}{D_{\text{eff}}} (\gamma D_{\text{eff}} \mu 2^{-\ell})^2 \exp(-\gamma D_{\text{eff}} \mu 2^{-\ell}/6)$$

to the left side of (16). If $\gamma D_{\text{eff}}\mu \le 1$, the sum is bounded by a geometric series with first term $2(\gamma D_{\text{eff}}\mu)^2/D_{\text{eff}}$. Otherwise, the terms with $\gamma D_{\text{eff}}\mu 2^{-\ell} \le 1$ sum to $O(1/D_{\text{eff}})$; the others are bounded by a constant times $D_{\text{eff}}^{-1} \sum_{j \ge 0} 4^j e^{-2^j/6}$, which converges.

For the last two groups, omit the exponential. Since $2^{\lfloor \log_2 D \rfloor} > D/2$, their largest value of $\gamma D_{\text{eff}} \mu 2^{-\ell}$ is at most

$$\frac{4\gamma D_{\text{eff}}\mu}{D} = \frac{4\gamma\mu}{\log_2 D} \le 1.$$

Their squared terms are also bounded by $(\gamma D_{\text{eff}}\mu)^2$, giving (16). This includes $D = 2, 3$, when there are only two groups. □