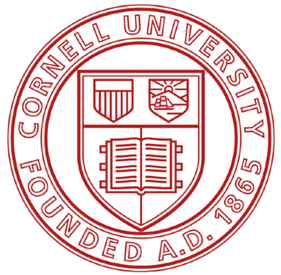
Lecture 22: Scaling generative models

CS 5788: Introduction to Generative Models



Many slides adapted from Phillip Isola

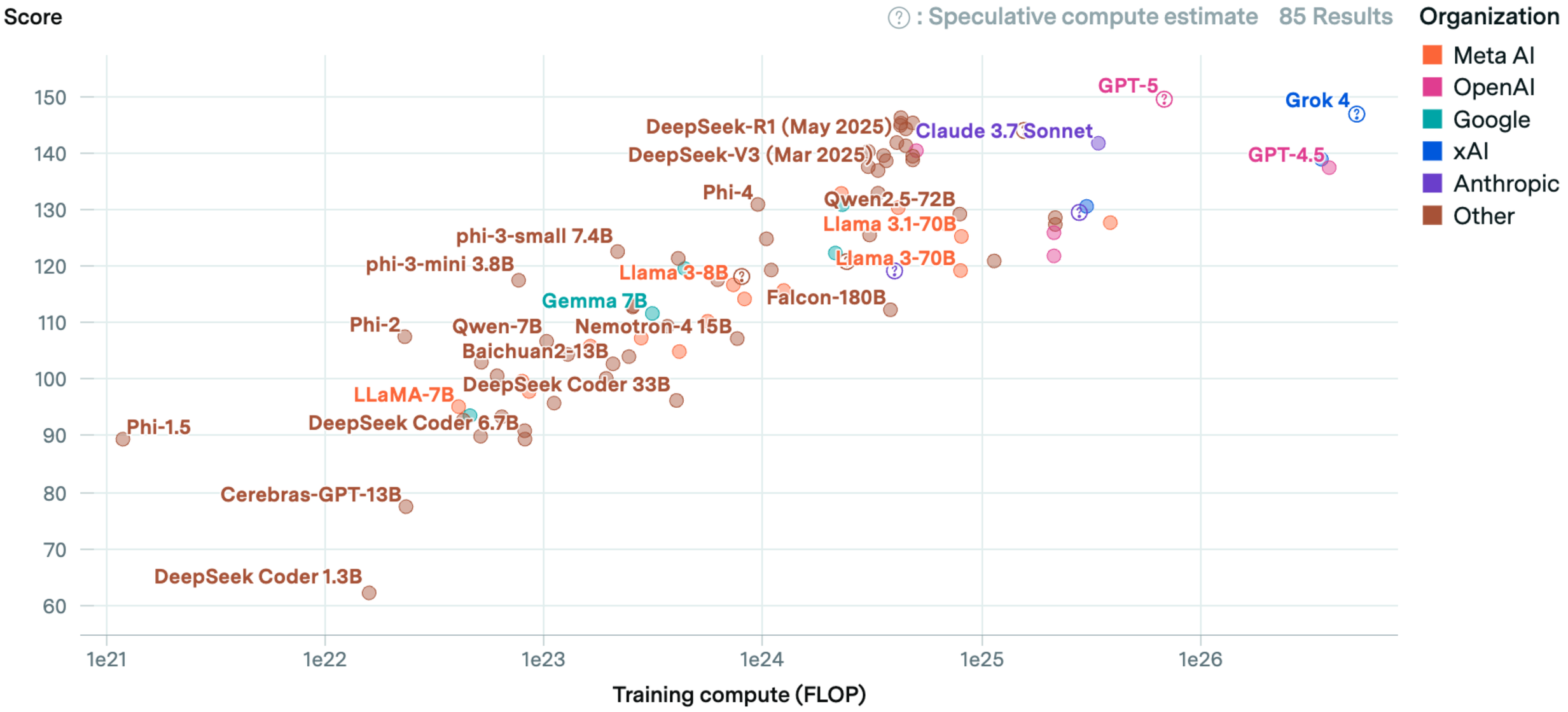
Exam reminder

- Tomorrow on Fri. 4/17, 2-5pm in auditorium.
- You'll have 3 hours.
- Mostly multiple choice, plus some written problems.
- Compared to the homework: more conceptual questions
- 1-page front and back hand-written "cheat sheet"
- No calculators (also unnecessary).
- No questions directly on:
  - Lec. 4 (neural network review)
  - Lec. 11 (generative models for computer vision guest lecture)

Today

- Scaling laws for generative models
- More exam review

Epoch Capabilities Index (ECI)



How does the *scale* affect these models?

There are many different things we can vary:

- The size of the network architecture
- The amount of training data
- The number of training iterations

What is the effect on the model's capabilities?

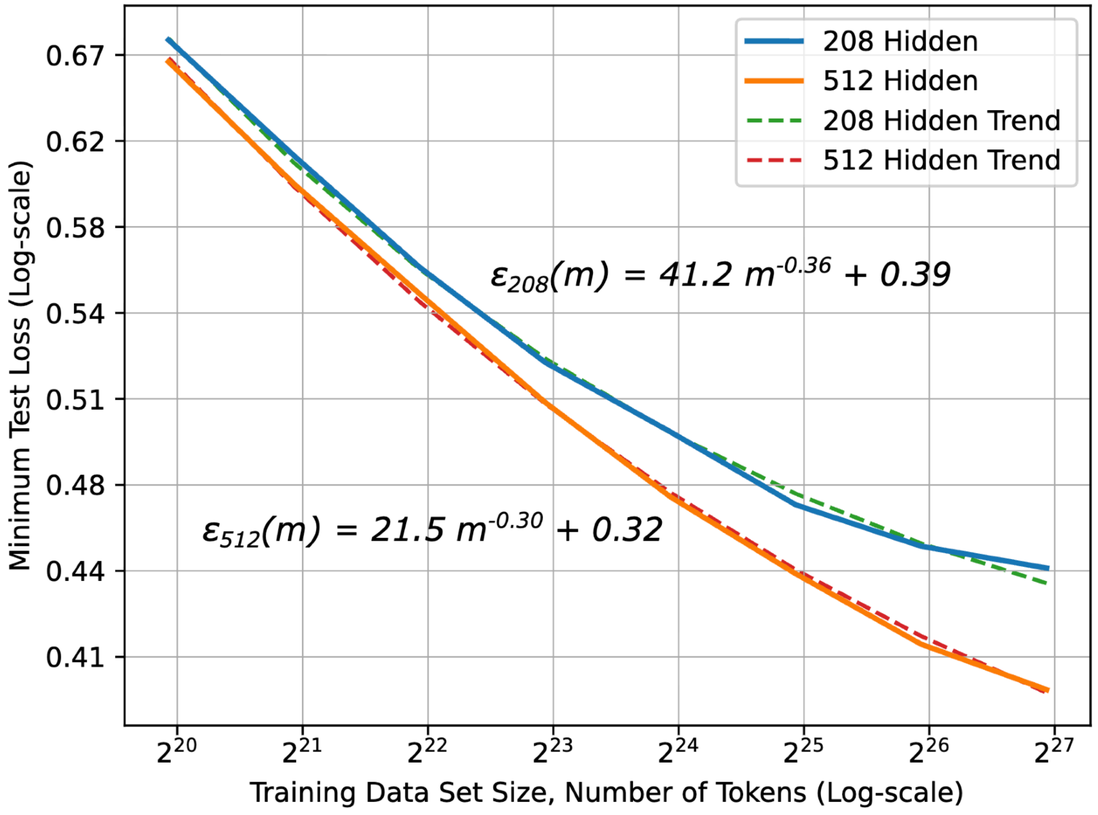
Discussion: performance vs. model size

When you increase the size of a neural net, how does the test accuracy change:

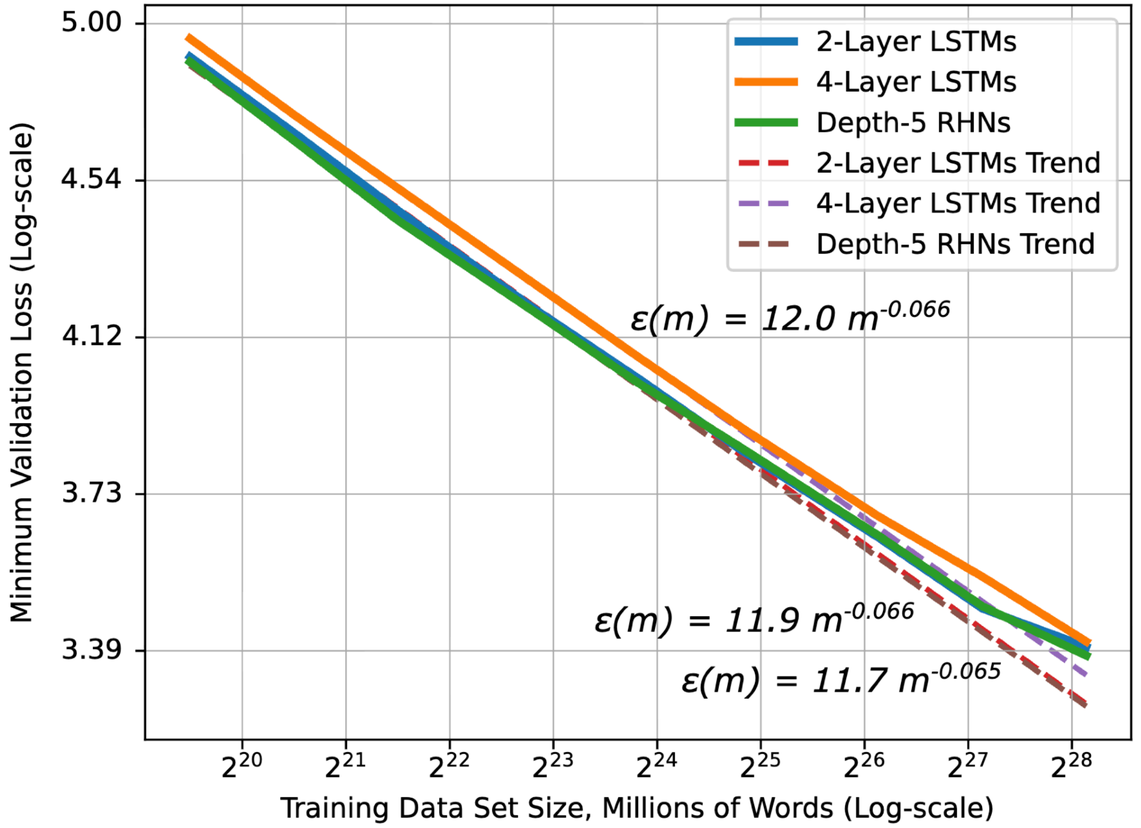
- On the training set?
- On the test set?
- When data is limited?

And at what rate?

Performance vs. model size



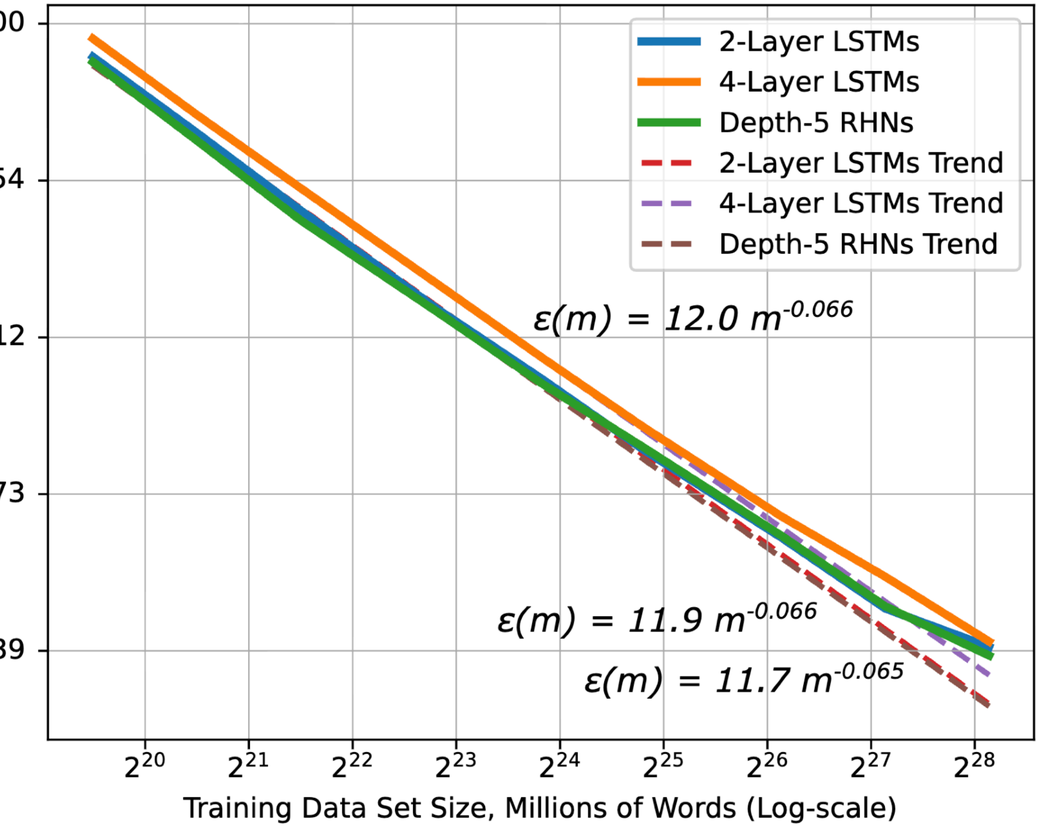
Machine translation



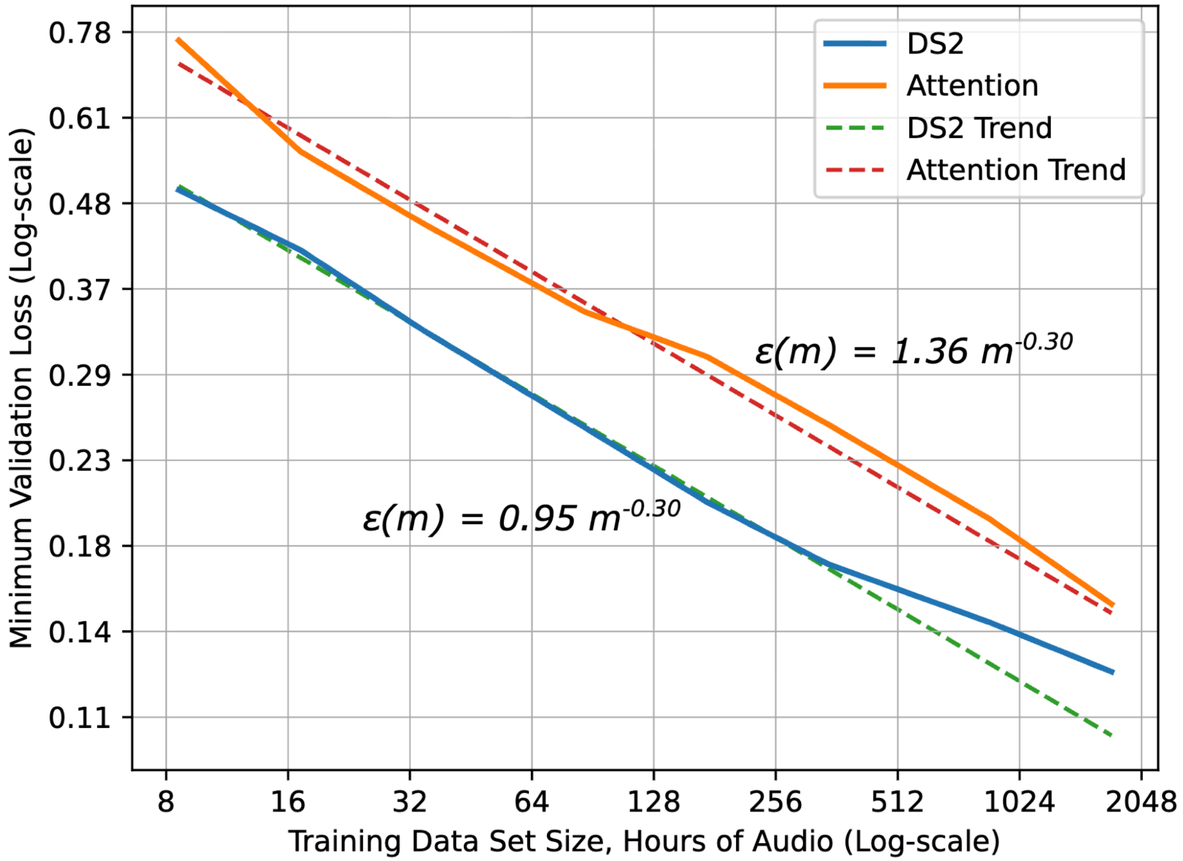
Language modeling

[Hestness et al., "Deep Learning Scaling Is Predictable, Empirically", 2017]

Performance vs. model size



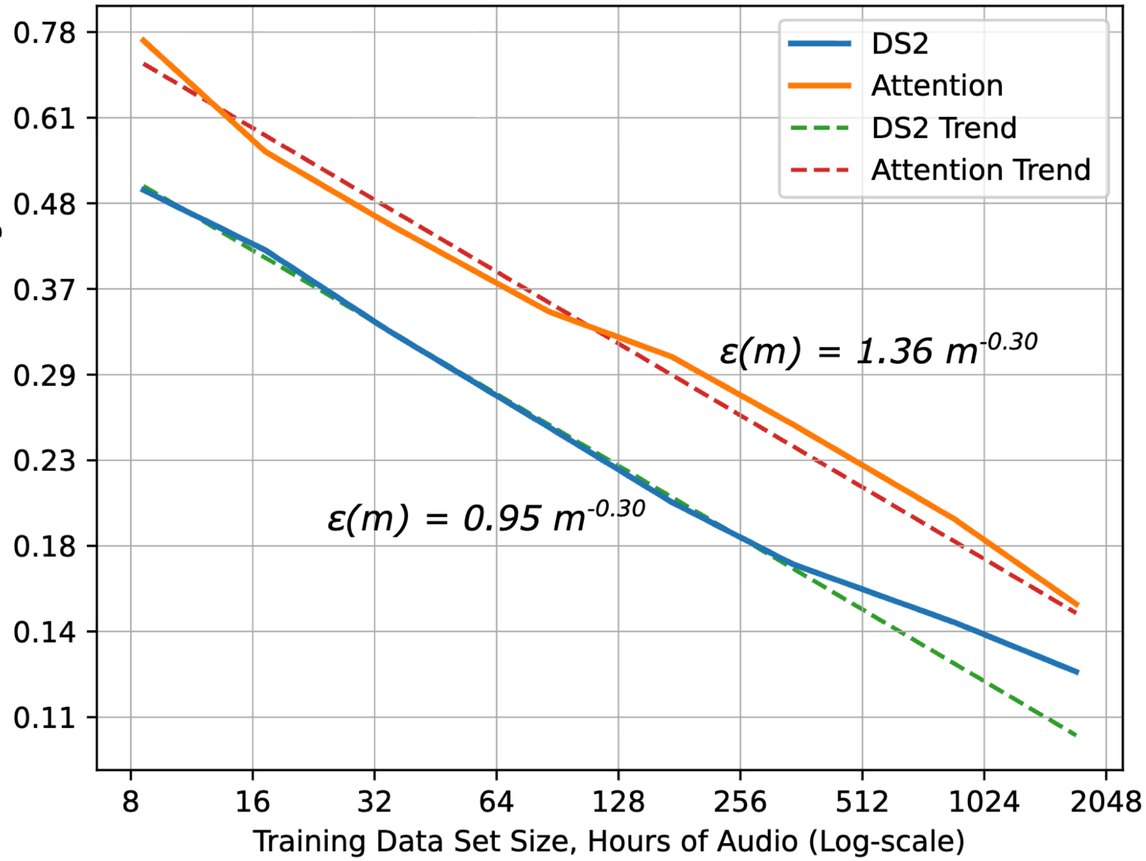
Language modeling



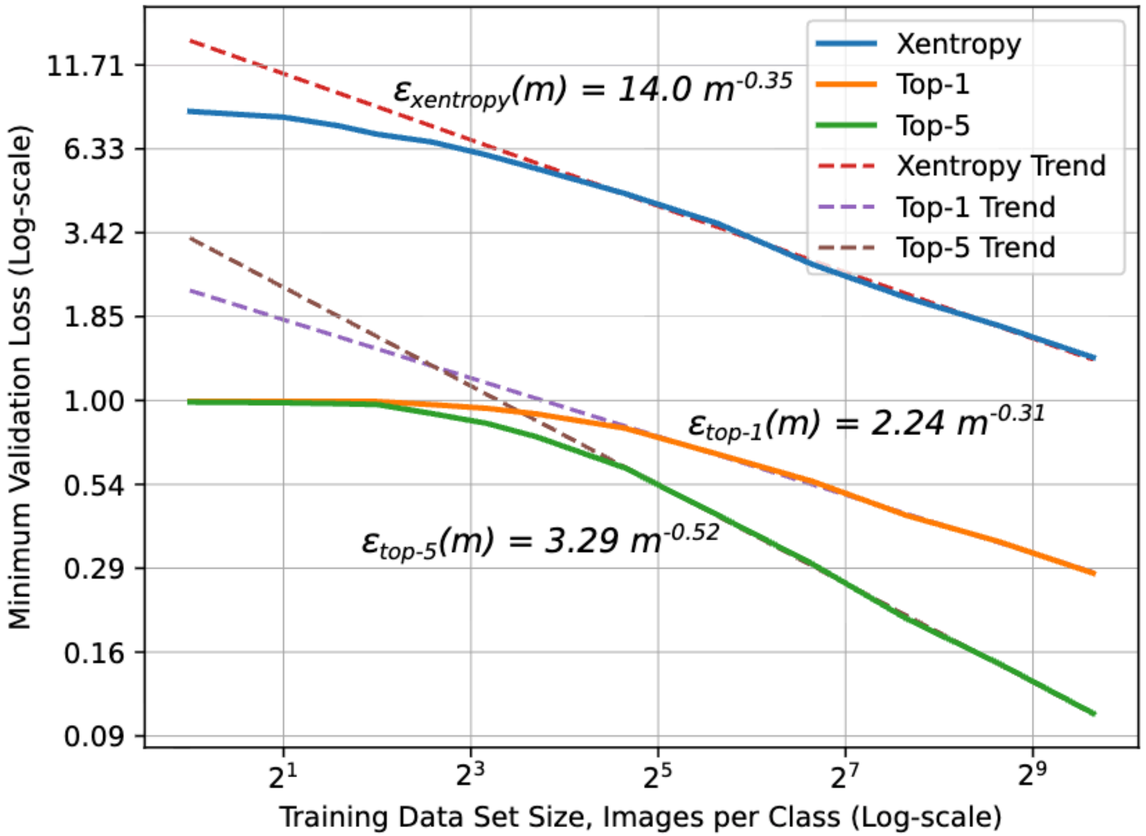
Speech recognition

[Hestness et al., "Deep Learning Scaling Is Predictable, Empirically", 2017]

Performance vs. model size



Speech recognition



Object recognition

[Hestness et al., "Deep Learning Scaling Is Predictable, Empirically", 2017]

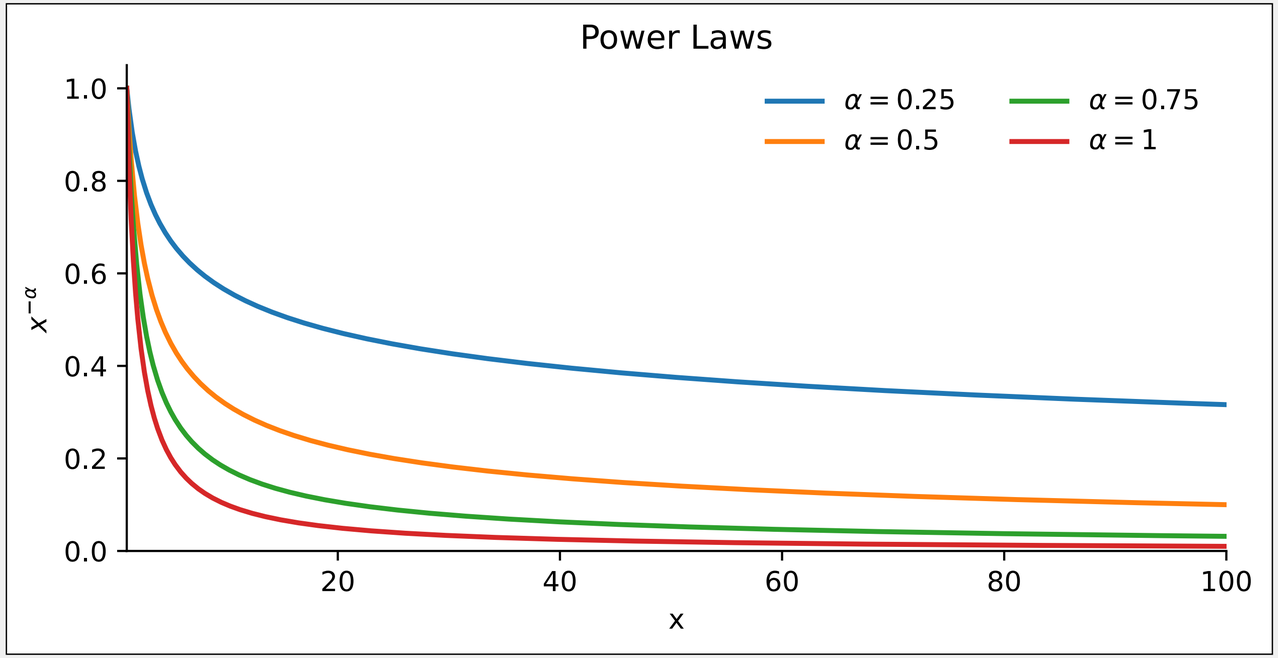
What’s the pattern?

Power laws

e.g., loss vs. parameters

$$L(X) = \left(\frac{1}{X}\right)^{\alpha}$$

Power law curves for various α parameter settings

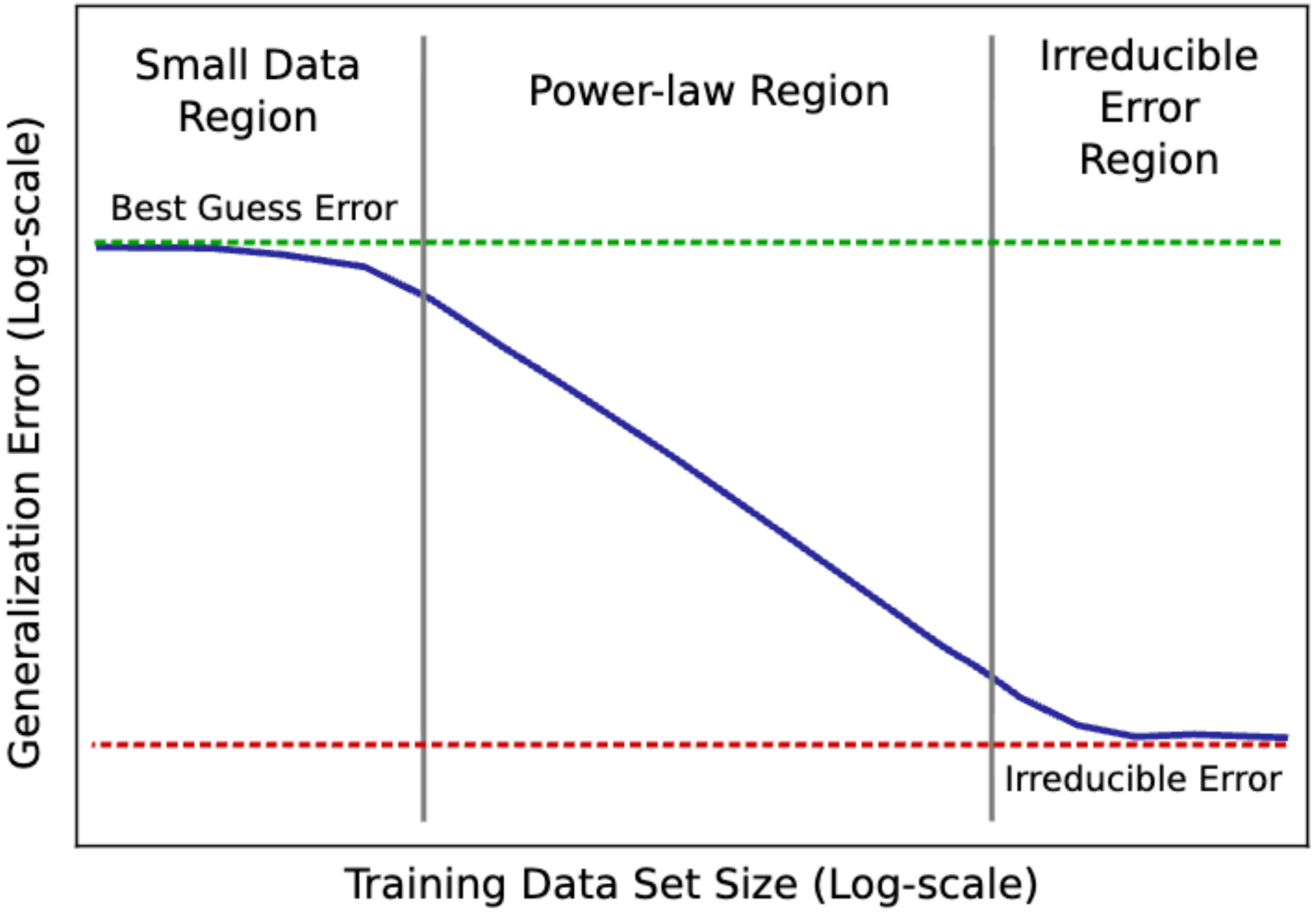


Power laws

Let's plug in some numbers.

$$L(10X) = \frac{1}{10^{\alpha}} L(X)$$

- Suppose that $\alpha = 1$ and you have a model with a 10% error rate. Then increasing the models scale by $10\times$ gives you a 1% error rate.
- Going from 10% to 0.01% error rate? Need a model that's $1000\times$ as large.
- Constants matter! If $\alpha = 0.5$, then $L(10X) \approx \frac{1}{3} L(X)$. Going from 10% to 0.01% would require a model that's 10^6 as large!



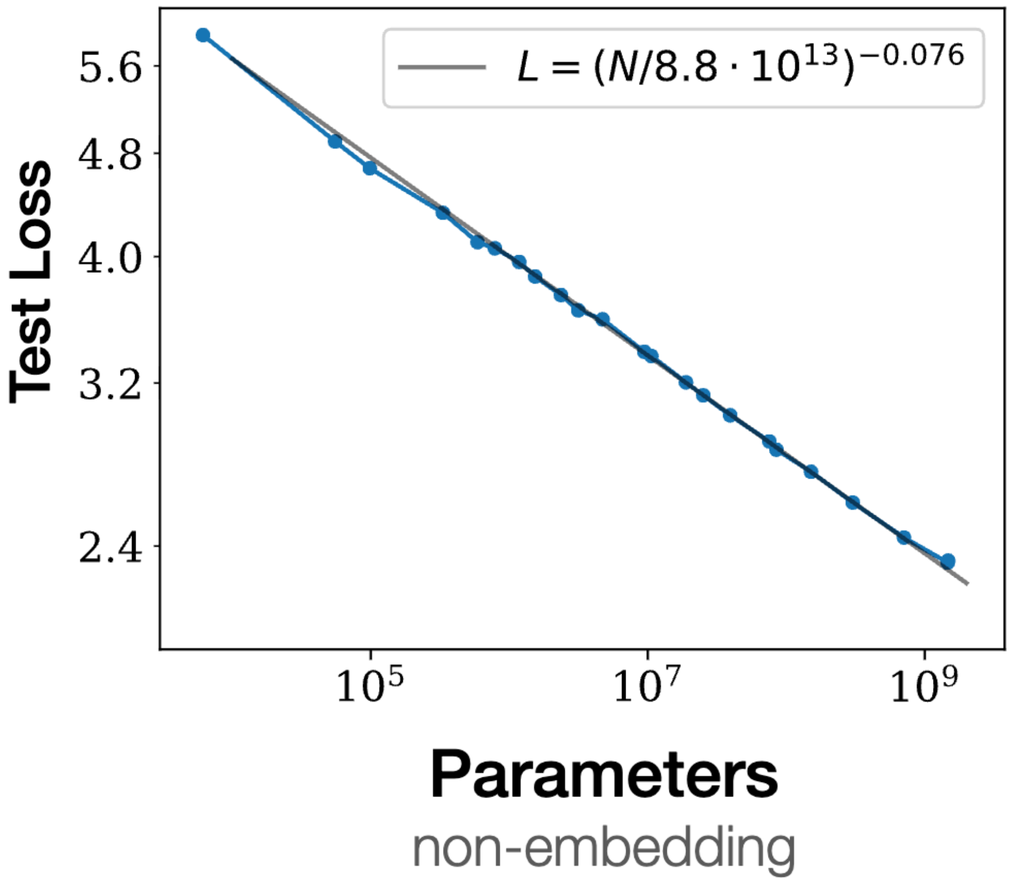
[Hestness et al., "Deep Learning Scaling Is Predictable, Empirically", 2017]

How do these laws apply to generative models?

Questions

1. What we really care about are *downstream* tasks. How does log likelihood going down affect this?
2. How favorable are the constant factors, like α?
3. What do these laws mean for *unsupervised* training, where you potentially have “infinite” data?
4. How predictive are these curves?
5. What scientifically can we learn by plotting performance on trend lines in general?

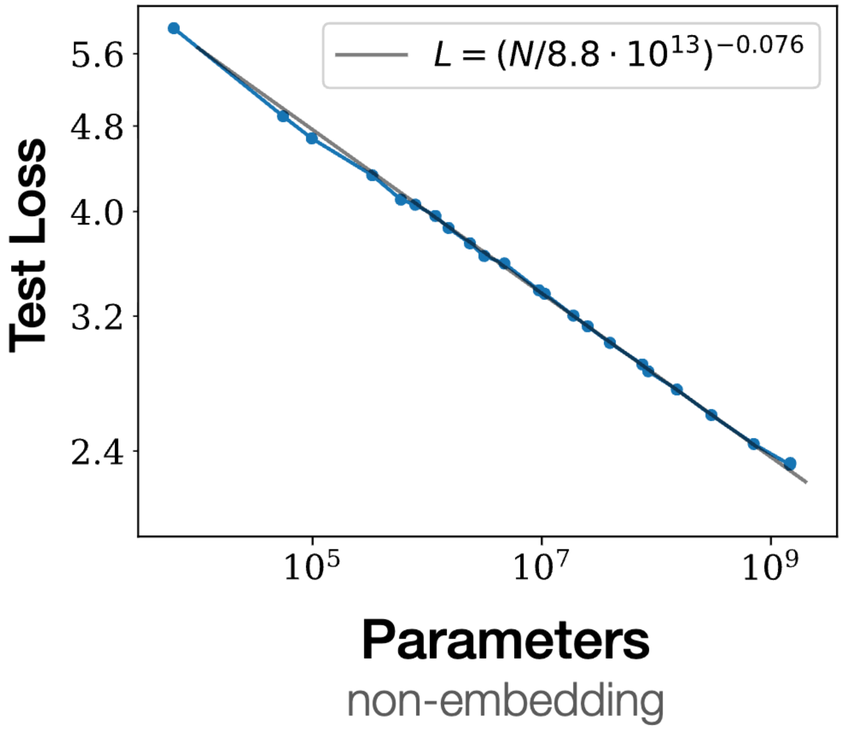
Scaling laws for language models

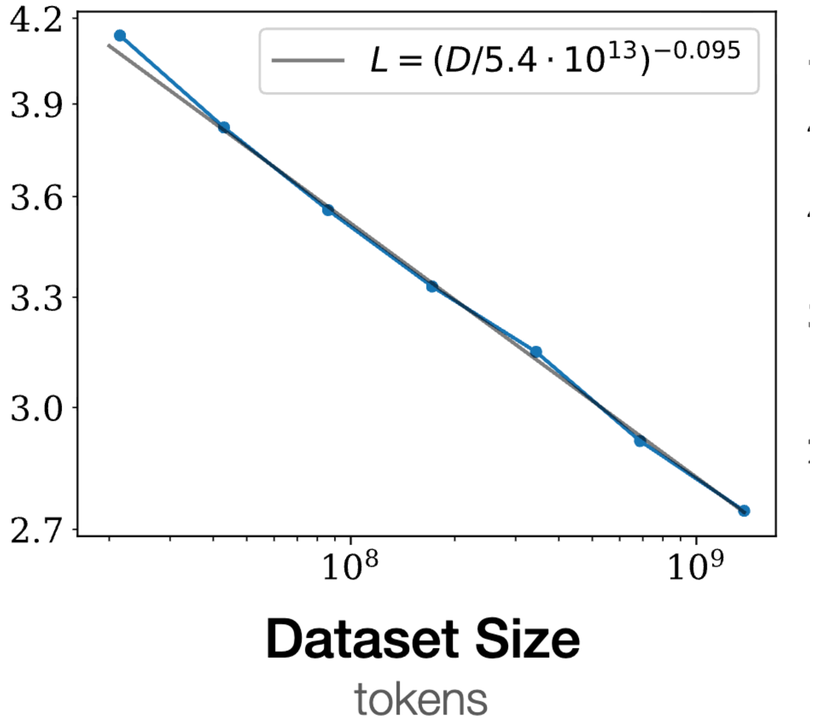


- Train GPT-style autoregressive transformers on ≈ 100 GB of text.
- Plot the test loss vs. the number of parameters in the model.

[Kaplan et al., "Scaling Laws for Neural Language Models", 2020]

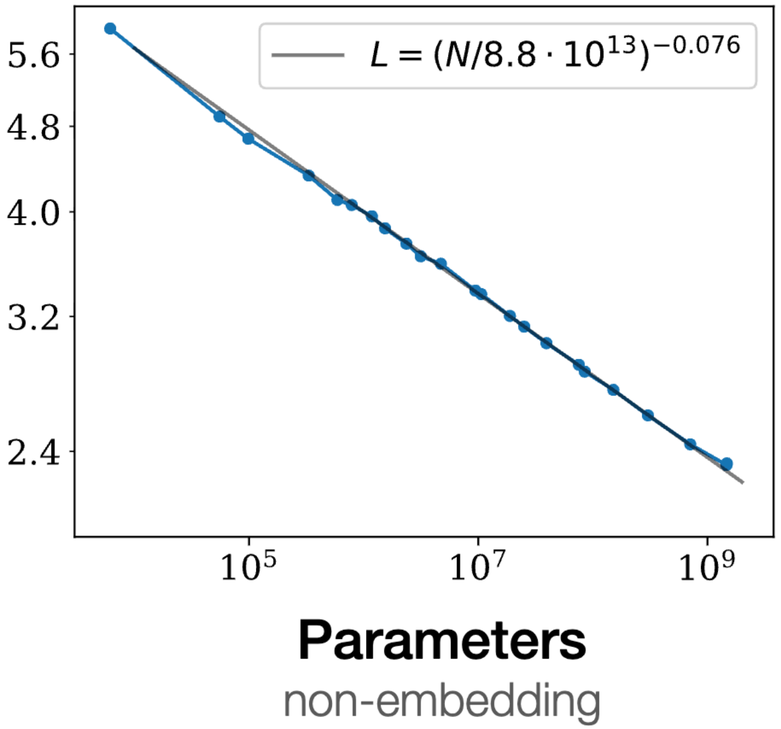
Scaling laws for language models





Source: [Kaplan et al., 2020]

Individual scaling laws:



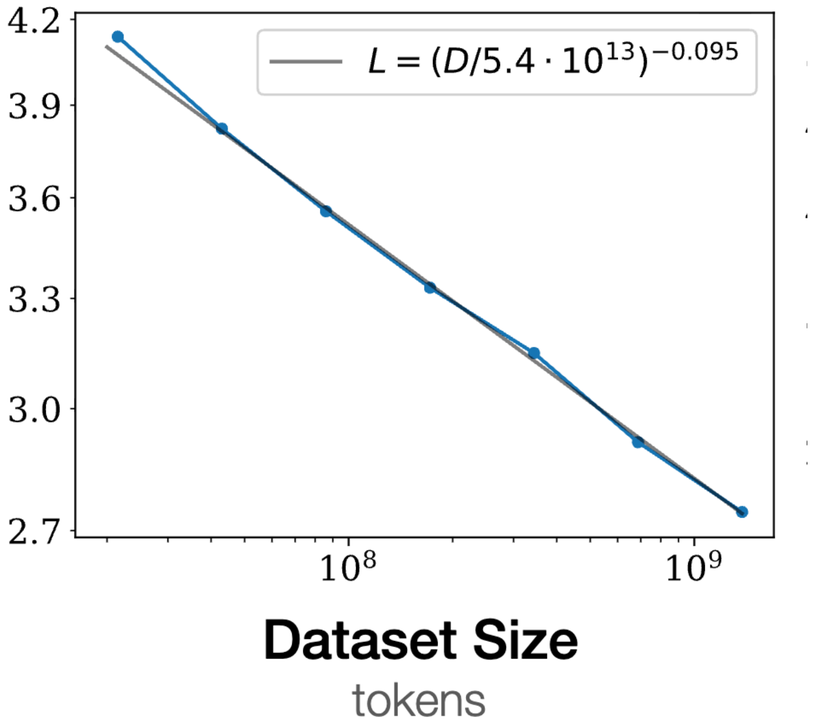
Scaling transformer parameters (ignoring embedding layer):

$$L(N) = (N_c/N)^{\alpha_N}$$

$$\alpha_N \sim 0.076, \quad N_c \sim 8.8 \times 10^{13}$$

Source: [Kaplan et al., 2020]

Individual scaling laws:



Scaling dataset size (in tokens), with early stopping:

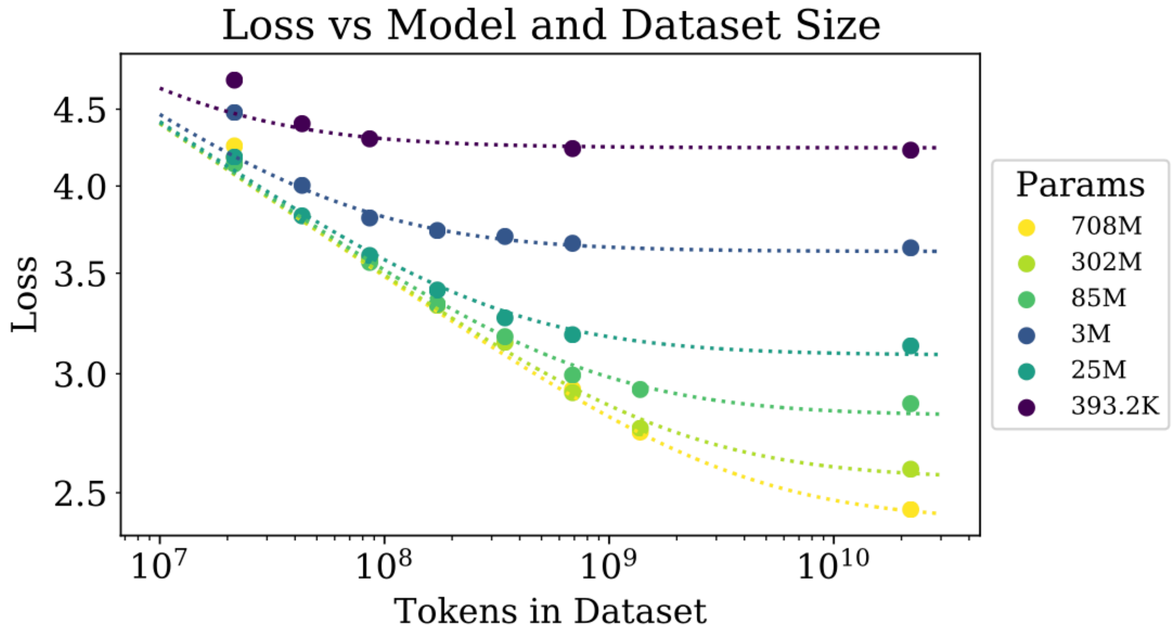
$$L(D) = (D_c/D)^{\alpha_D}$$

$$\alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13}$$

Source: [Kaplan et al., 2020]

Unifying data and parameter scaling

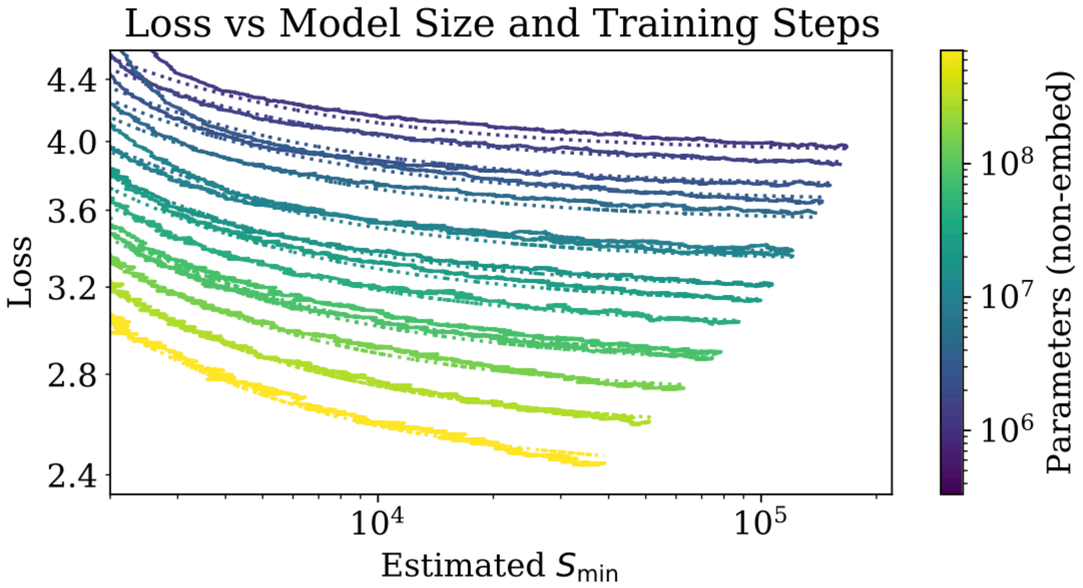
- Fit a curve based on N and D.
- Empirically-driven fit. Plus satisfying other desiderata like having $L(N)$ and $L(D)$ as special cases in infinite case limit.



$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D}$$

→ Suggests that parameters and dataset need to grow together.

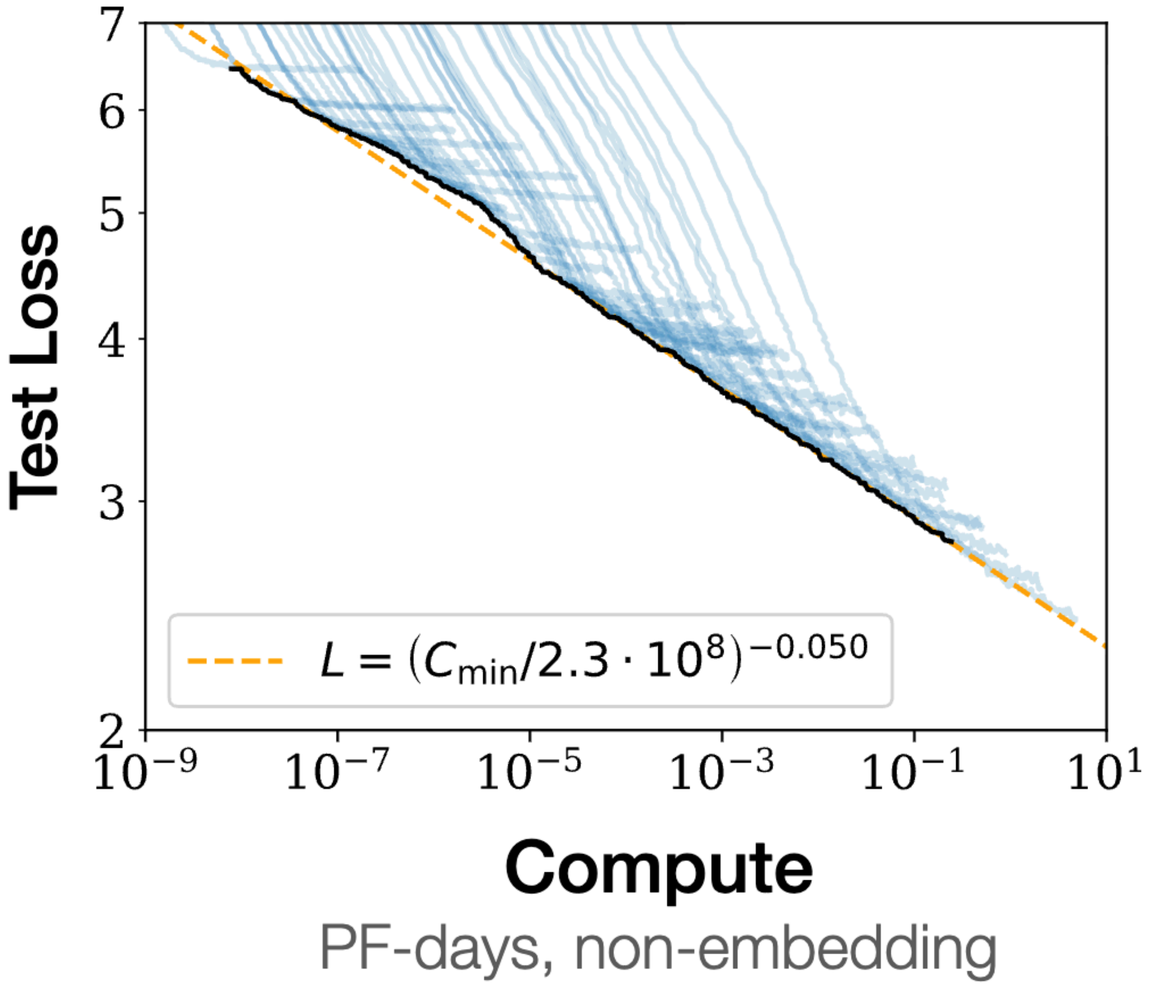
Considering the number of training steps



- Given that we often have “infinite” data on the web, natural to plot loss vs. steps S.
- Rather than using raw step count S, attempt to control for batch size, giving a batch-adjusted step count $S_{\text{min}}(S)$.

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\text{min}}(S)}\right)^{\alpha_S}$$

Another way that we can slice the data: compute



Approximate *compute* performed as:

$$C = 6NBS$$

- Estimated number of floating point operations (FLOPS) using their transformer setup.
- B = batch size, S = training steps
- In their transformer model, $\approx 6N$ operations per token, where N is the number of parameters.

Source: [Kaplan et al., 2020]

Why are these laws useful?

Transferring to downstream tasks

Increasing model size leads to better results on downstream tasks.

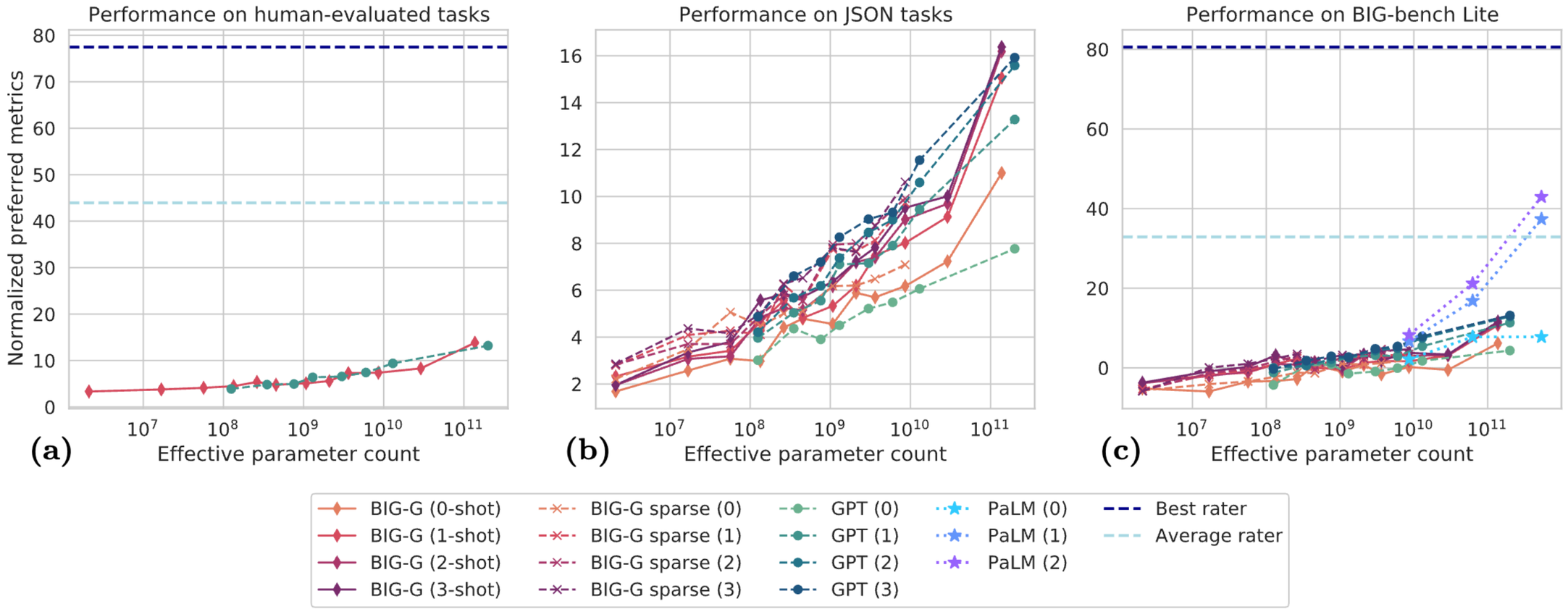
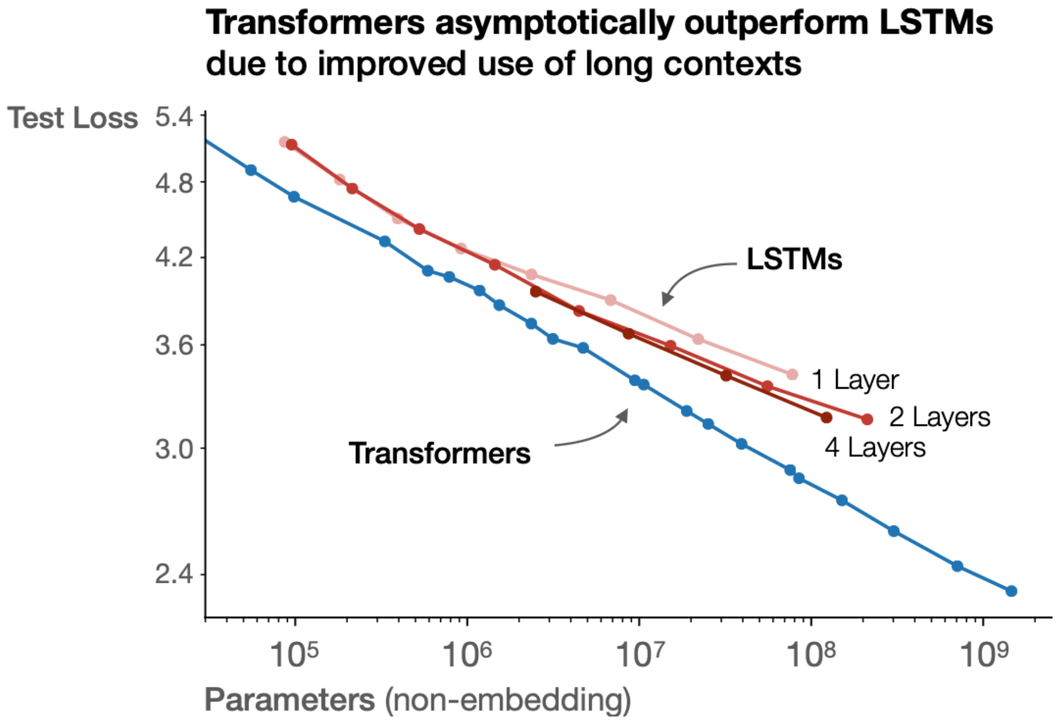
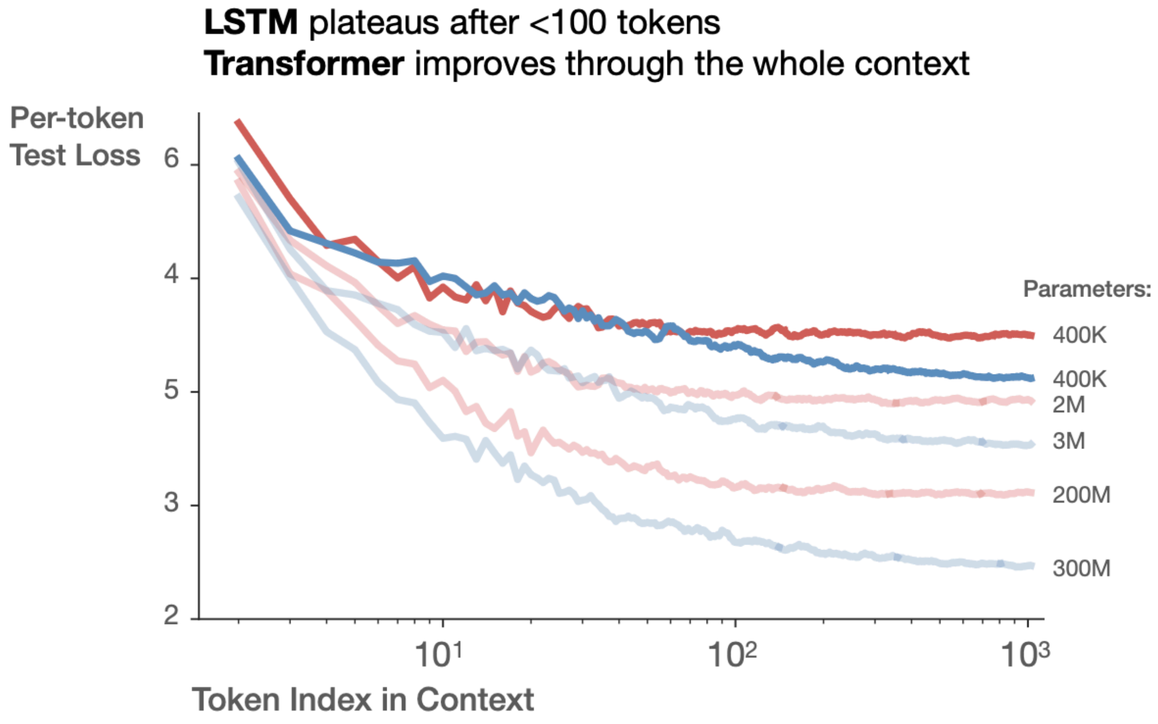


Figure source: [BIG-bench, 2022]

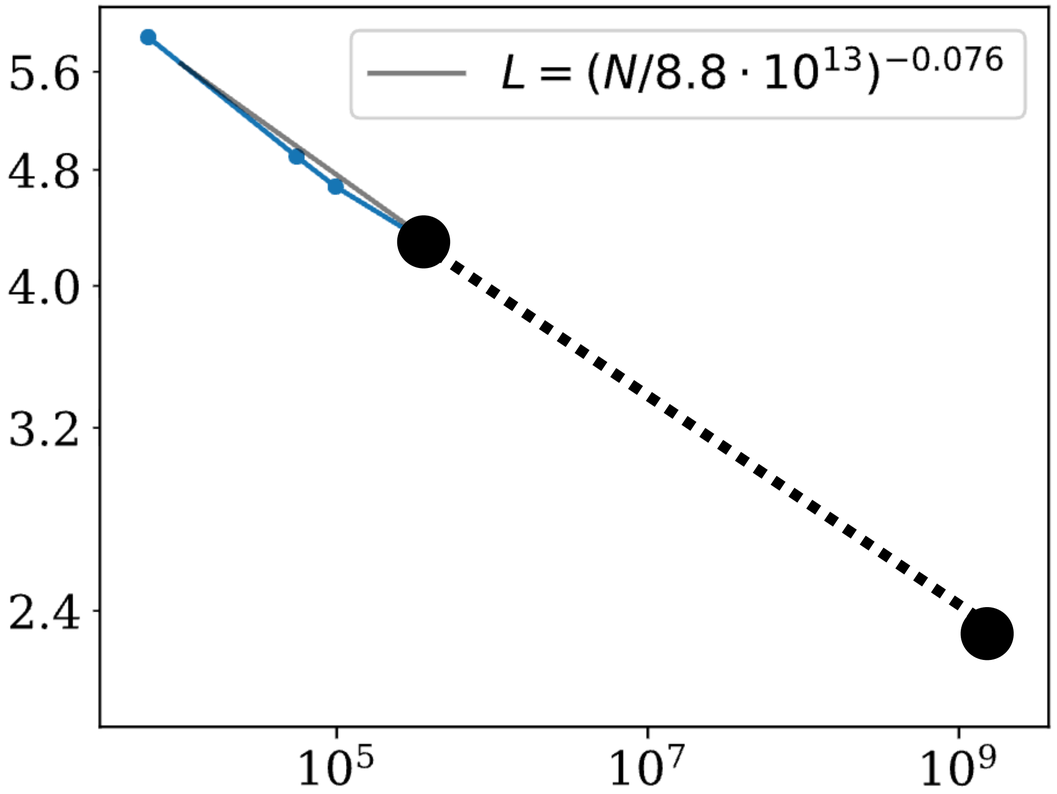
Comparing scaling laws for different architectures





Extrapolating performance

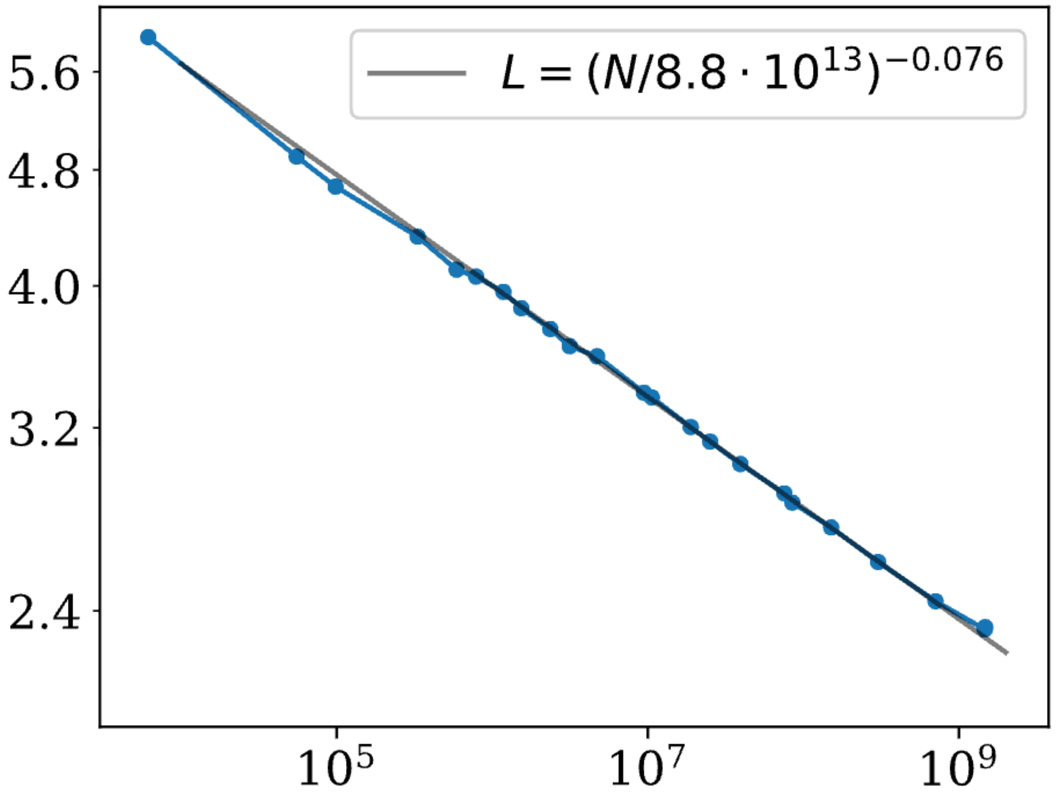
Allows you to develop a small model, then get predictable results when you scale it up by several orders of magnitude.



Inspired by slide from Phillip Isola

Extrapolating performance

Allows you to develop a small model, then get predictable results when you scale it up by several orders of magnitude.



Inspired by slide from Phillip Isola

How should we spend our compute budget?

- We modeled the scaling behavior as:

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)}\right)^{\alpha_S}$$

- Given that $C \propto NBS$ and other assumptions (e.g., optimal batch size), we can minimize the predicted loss for each hyperparameter:

$$N \propto C^{\alpha_C^{\min}/\alpha_N}, \quad B \propto C^{\alpha_C^{\min}/\alpha_B}, \quad S \propto C^{\alpha_C^{\min}/\alpha_S}, \quad D = B \cdot S$$

where $\alpha_C^{\min} = 1/\left(1/\alpha_S + 1/\alpha_B + 1/\alpha_N\right)$

Equations from [Kaplan et al., 2020]

How should we spend our computation budget?

Plugging in numbers, we get:

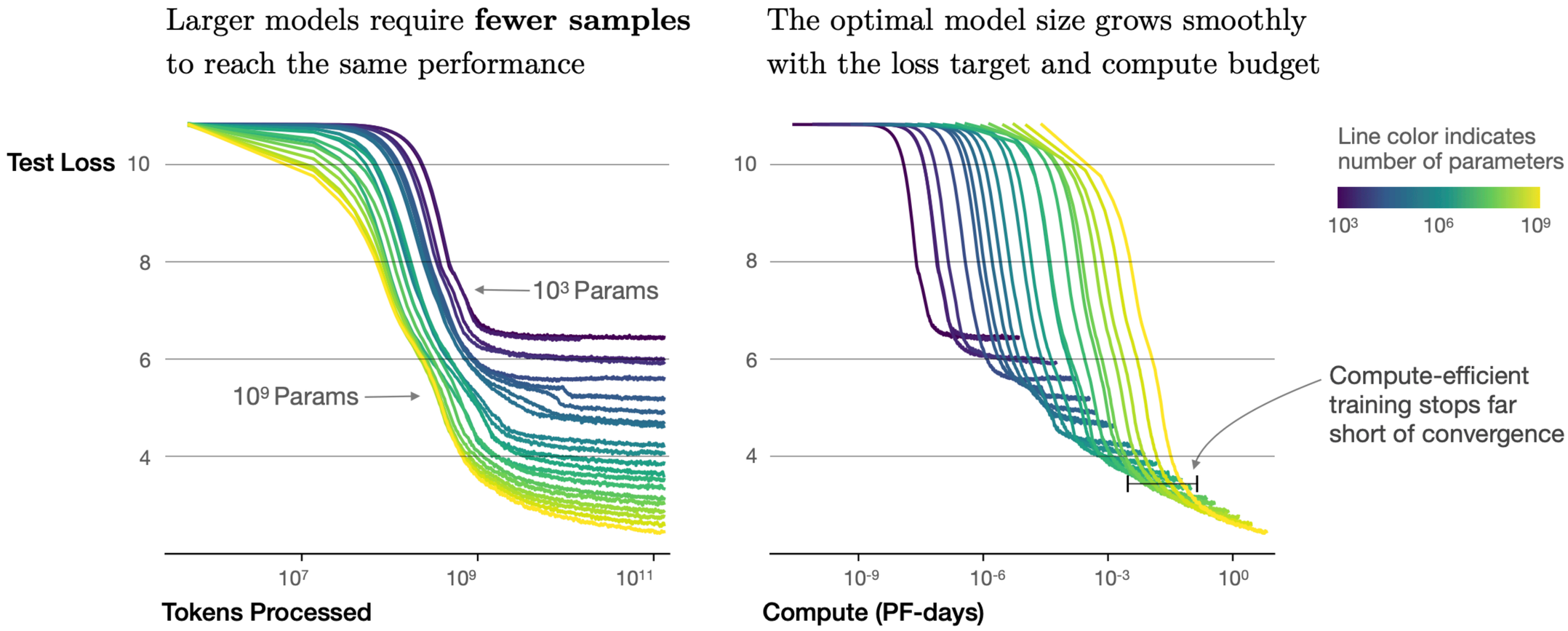
$$N \propto C_{\text{min}}^{0.73},\ B \propto C_{\text{min}}^{0.24},\ \text{and}\ S \propto C_{\text{min}}^{0.03}$$

Claim from from [Kaplan et al., 2020] about how to allocate resources:

which closely matches the empirically optimal results $N \propto C_{\text{min}}^{0.73}$, $B \propto C_{\text{min}}^{0.24}$, and $S \propto C_{\text{min}}^{0.03}$. As the computational budget C increases, it should be spent primarily on larger models, without dramatic increases in training time or dataset size (see Figure 3). This also implies that as models grow larger, they become

Source: [Kaplan et al., 2020]

One implication: big vs. small models



Also suggests larger models are more sample efficient.

Source: [Kaplan et al., 2020]

Models after Kaplan et al.

For 2 years after this paper, frontier models matched this training recipe, e.g., training large models on 300B tokens.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion

Figure source: [Hoffmann et al., 2022]. Slide adapted from Phillip Isola

DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

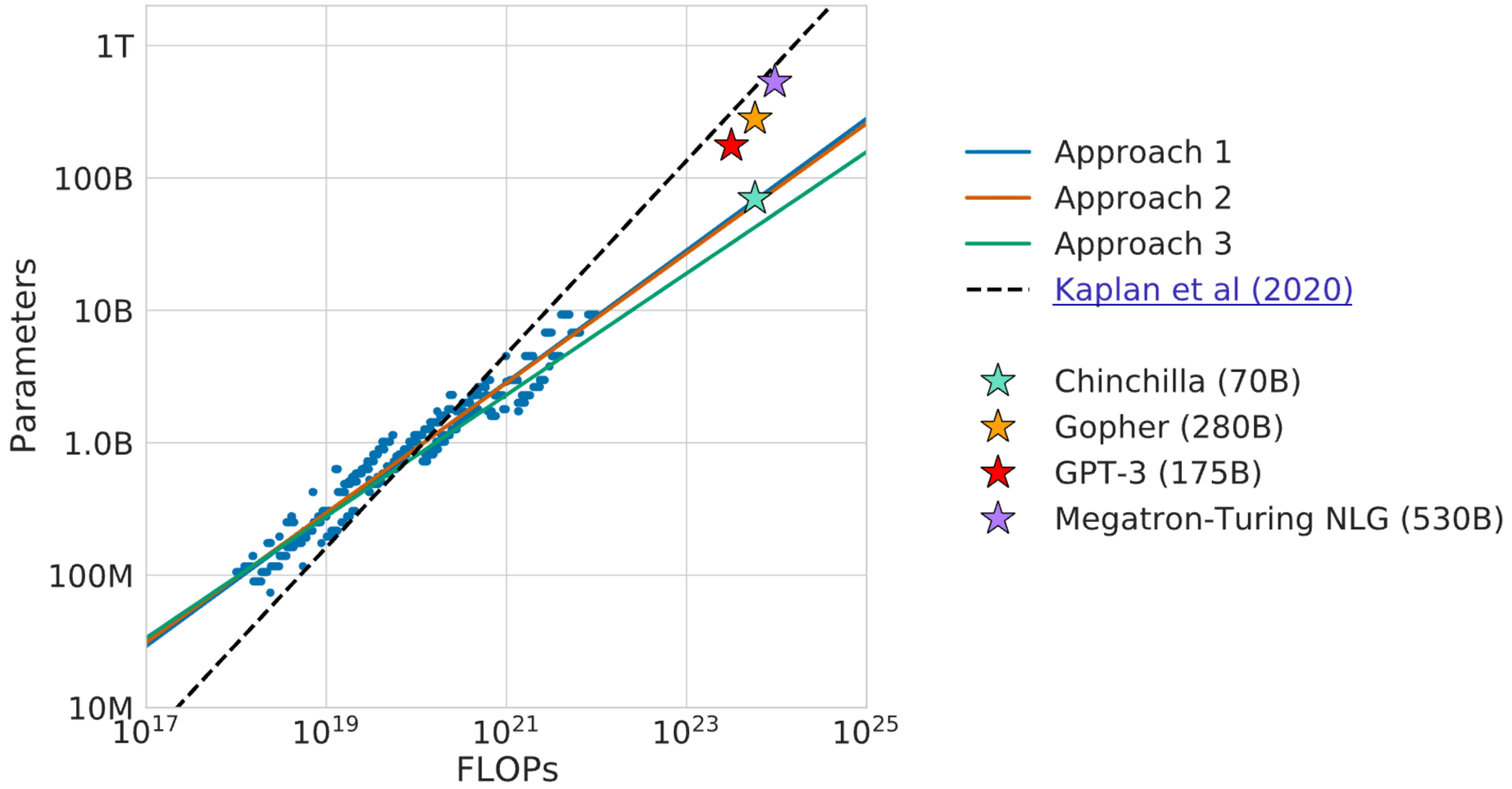
We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Slide adapted from Phillip Isola

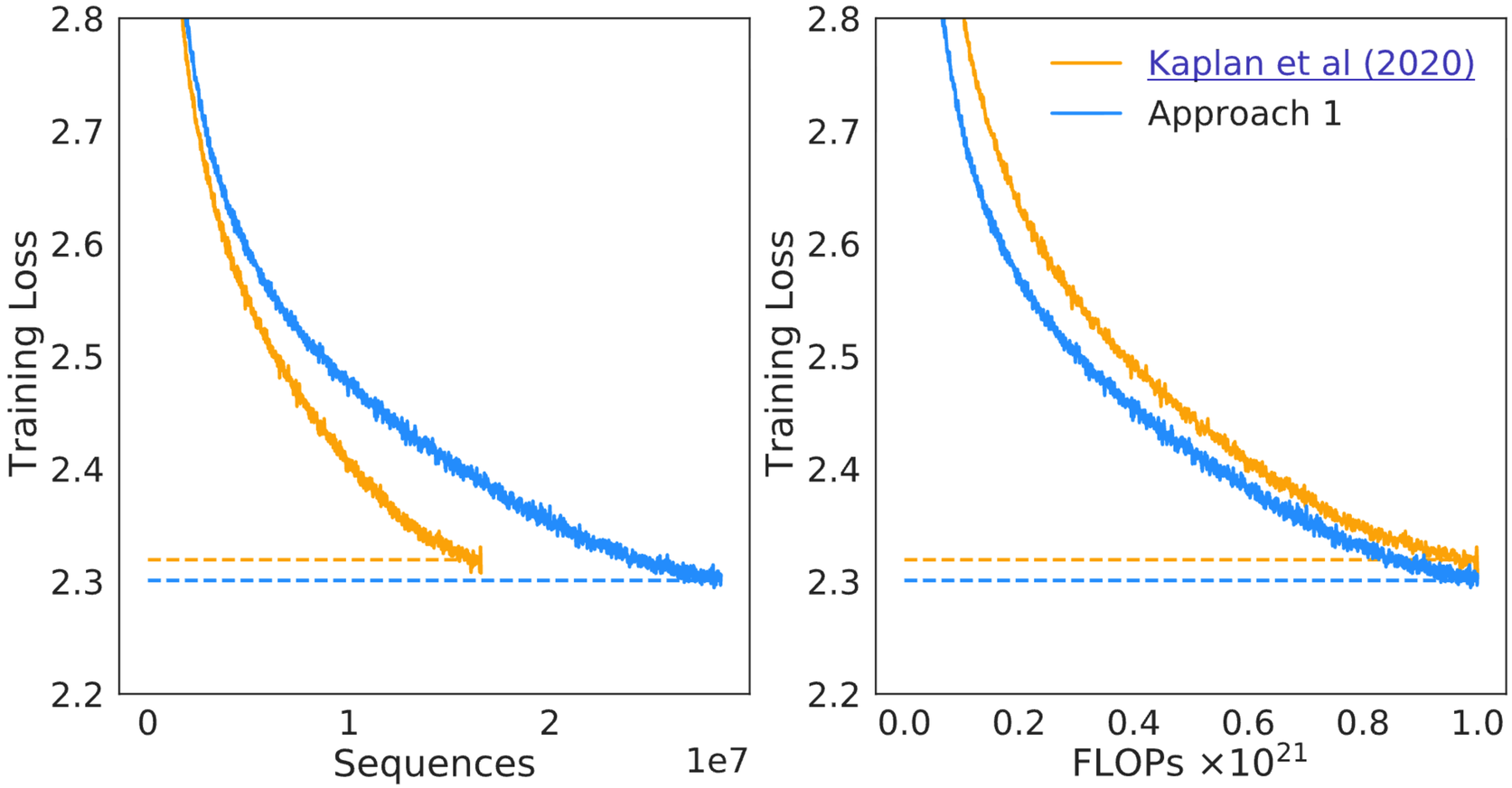
Revisiting the scaling laws

- Previous work [Kaplan et al., 2022] used a fixed the learning rate schedule for all models, rather than choosing it as a function of the dataset size.
  - Underestimates the quality of the model on smaller datasets.
- It also focuses on models that are less than 1B parameters.
  - There is some curvature in the loss surface for larger models (e.g., as you go up to 16B)

Revisiting the scaling laws



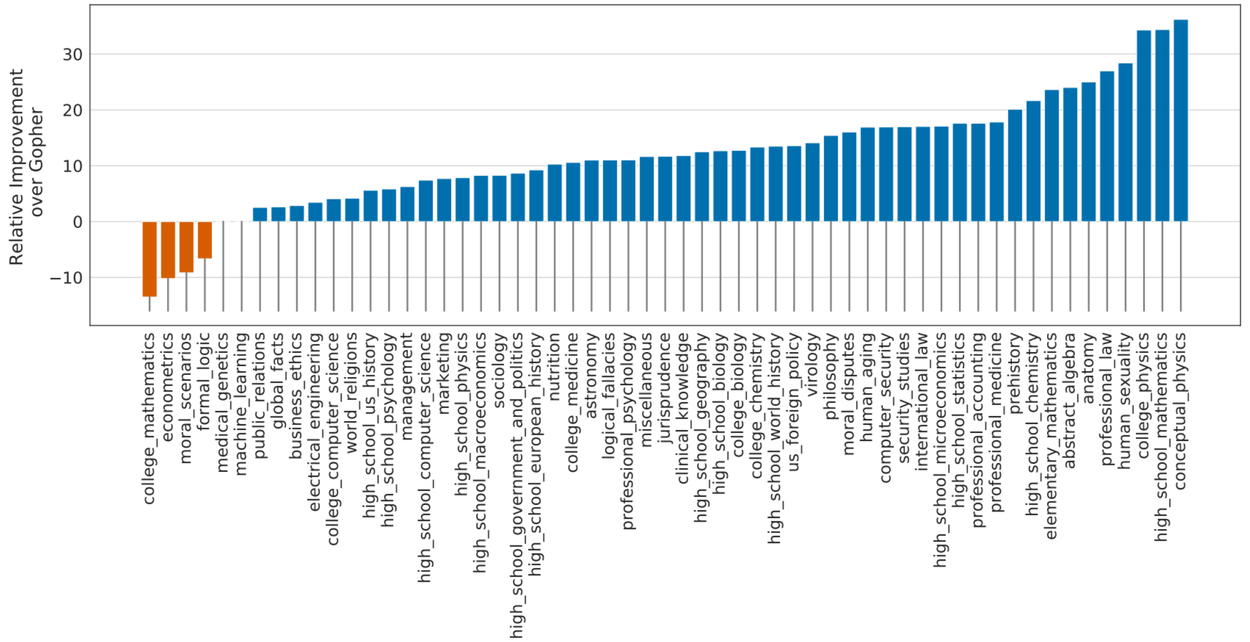
Revisiting the scaling laws



Revisiting the scaling laws

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

Model comparisons



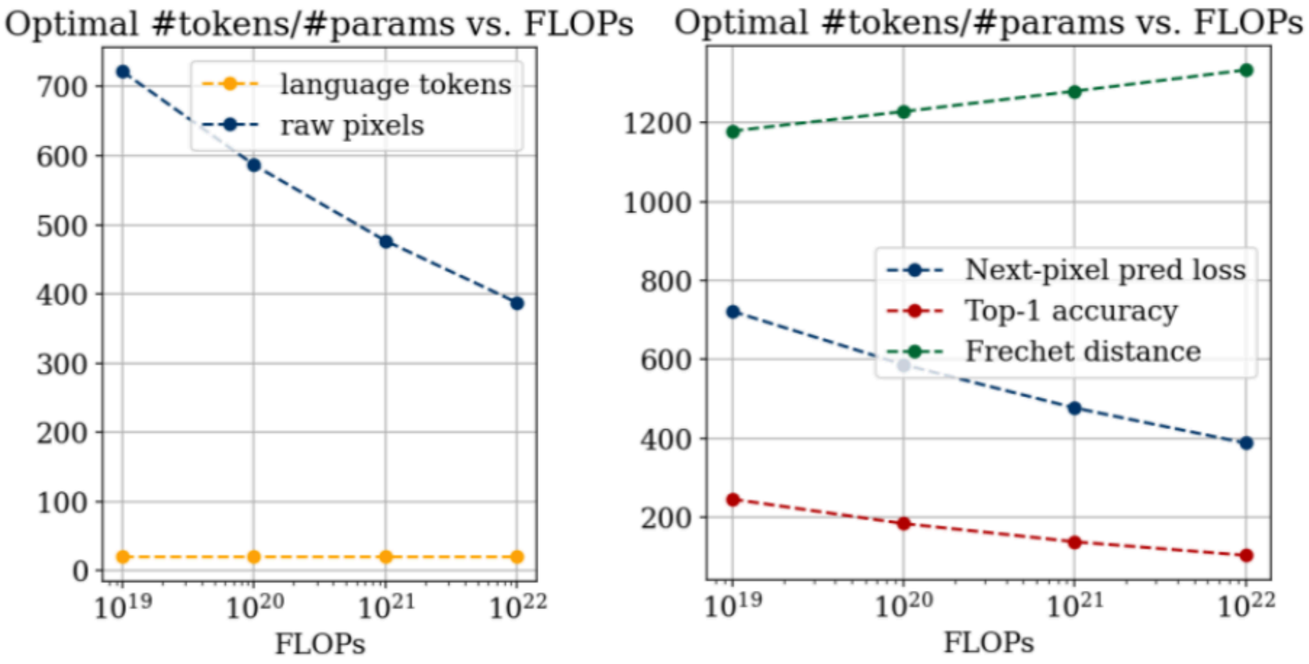
Performance on MMLU benchmark vs. Gopher

Source: [Hoffmann et al., 2022]

Revisiting the scaling laws

- They call these new laws *Chinchilla* scaling laws.
- A model that is $4\times$ smaller and trained on $4\times$ as much data does just as well as a larger model (Gopher) previously trained by Google.
- These models are more efficient to run at inference time, too.
- Not necessarily optimal for all cases. For example, you may want to “overtrain” a small model if it needs to optimize compute at *deployment* time.

Scaling laws for autoregressive image generation



- Scaling laws suggest that pixel-by-pixel image generation, relative to language, requires $10-20\times$ token-to-parameter ratio.
- Very compute (rather than data) constrained.

[Yan et al., “Rethinking generative image pretraining: How far are we from scaling up next-pixel prediction?”, 2025]

Scaling laws for generative models

- Scaling laws allow you to:
  - Design small models, extrapolate to large ones
  - Choose “optimal” hyperparameters given a compute budget
- They are hard to fit! Tiny experimental issues have a big impact.
- Lots of caveats:
  - Unclear how much to extrapolate, and what the true shape of the curve is, especially at extremes (very large and small models).
  - Theoretical explanations are an open problem.
- Might not capture important properties, like data quality, that are hard to quantify.
  - For example, pruning the dataset can improve scaling [Sorscher et al., 2022]

More exam review

Sample question

How does adding a constant to the energy function $E'_\theta(\mathbf{x}) = E_\theta(\mathbf{x}) + C$ change the resulting probability density for an EBM?

Sample question

Suppose that an image generation model *A* obtains better log likelihood on a validation set than does model *B*.

Does this mean that the samples from *A* have higher visually quality than those of *B*? Explain why or why not.

Recall: does MLE lead to good samples?

- Suppose that we mix a really good model of a distribution, $p(x)$, with pure random noise, $q(x)$.

$$p_M(x) = 0.01p(x) + 0.99q(x)$$

- What happens to the log likelihood for high dimensional data?

$$\log\left[0.01p(x) + 0.99q(x)\right] \geq \log(0.01p(x)) = \log p(x) - \log 100$$

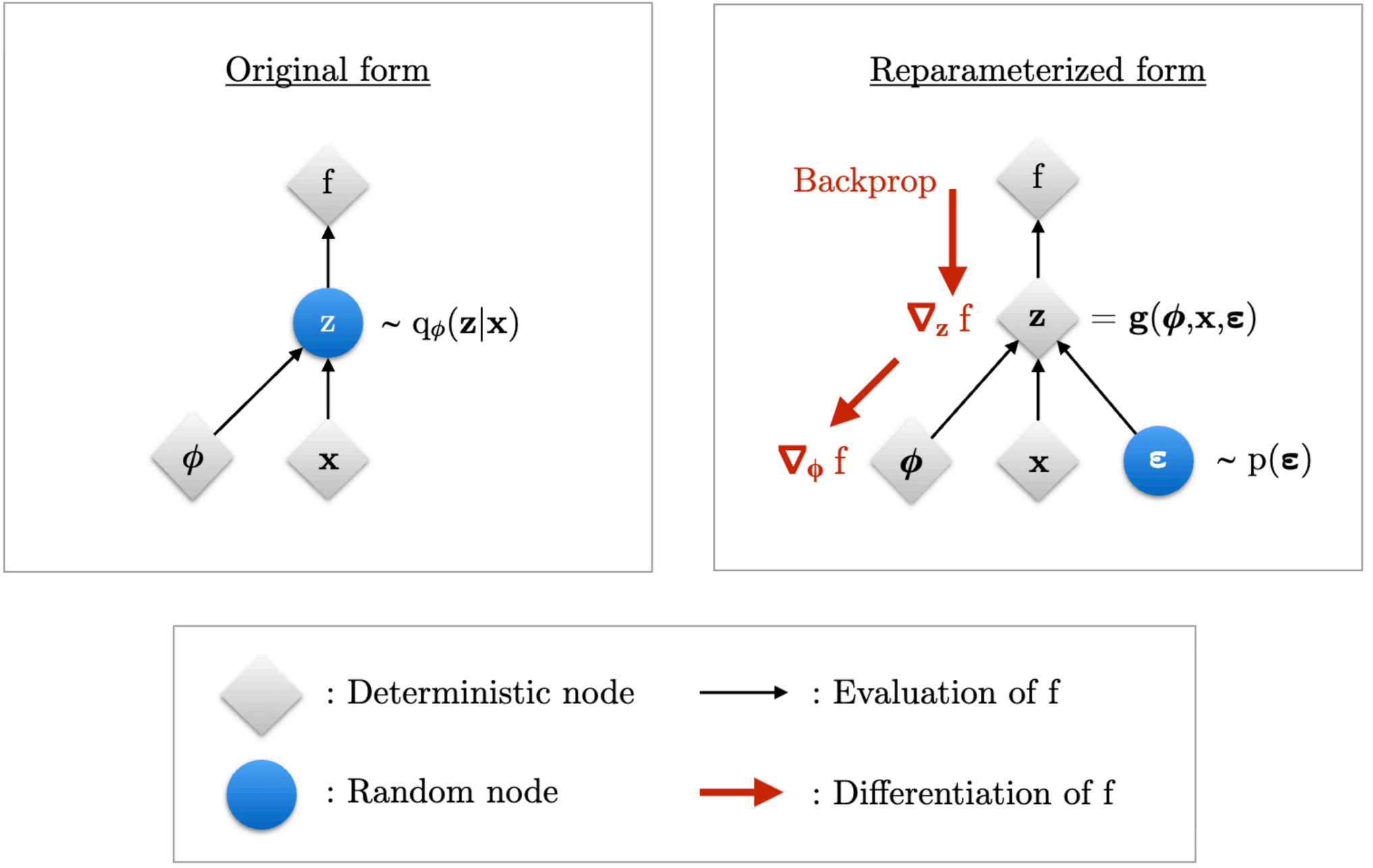
- But $\log p(x)$ is proportional d and $\log(100) \approx 4.61$ is a constant.
- Different models might differ in $\log(p(x))$ by $10000\times$
- So, a model can generate bad samples most of the time without affecting log likelihood much.

Source: ["A note on the evaluation of generative models", Theis et al. 2016]

Sample question

Why do we use the reparameterization trick $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \epsilon$ when training Variational Autoencoders (VAEs)?

A. It reduces the underlying dimensionality of the latent space to prevent overfitting on small datasets.
B. It allows gradients to backpropagate through the stochastic sampling node during training.
C. It forces the prior distribution to exactly match an isotropic standard normal distribution without requiring a KL divergence penalty.
D. It increases the variance of the generated samples to prevent mode collapse commonly observed in standard autoencoders.



[Source: Kingma & Welling, 2019]

Sample question

True or false: In comparison to the *forward* KL divergence $D_{KL}(p^* \parallel p_\theta)$, the *reverse* KL divergence $D_{KL}(p_\theta \parallel p^*)$ often results in "mode seeking" behavior. That is, the learned model generative model p_θ places probability mass across only a small number of modes in the true distribution p^*.

Recall the reverse KL: $\mathrm{D}_{KL}(p_\theta \,\|\, p^*) = \sum_x p_\theta(x) \log \left(\frac{p_\theta(x)}{p^*(x)} \right)$

$$\mathrm{D}_{KL}(p_\theta \,\|\, p^*) = \sum_x p_\theta(x) \log \left(\frac{p_\theta(x)}{p^*(x)} \right)$$

Good luck!