LLM Scaling Laws

Hello

- Who am I?
  - I’m Thom Foster
  - 3rd year PhD student with Jakob Foerster
  - Been at Meta for the last year building AI Research Agents
- What’s my connection to LLM scaling?
  - My research is on training LLMs in the ‘infinite compute, limited data’ regime
  - In this setting the primary importance is ‘what you train on’ not how

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

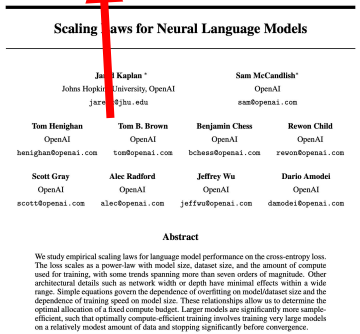
We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

Timeline of GPT Model Releases

GPT-1
Jun 2018

GPT-2
Feb 2019

GPT-3
Jun 2020



Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jared@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

GPT-3.5
Nov 2022

GPT-4
Mar 2023

GPT-4o
May 2024

GPT-5
Aug 2025

2018 2019 2020 2021 2022 2023 2024 2025 2026

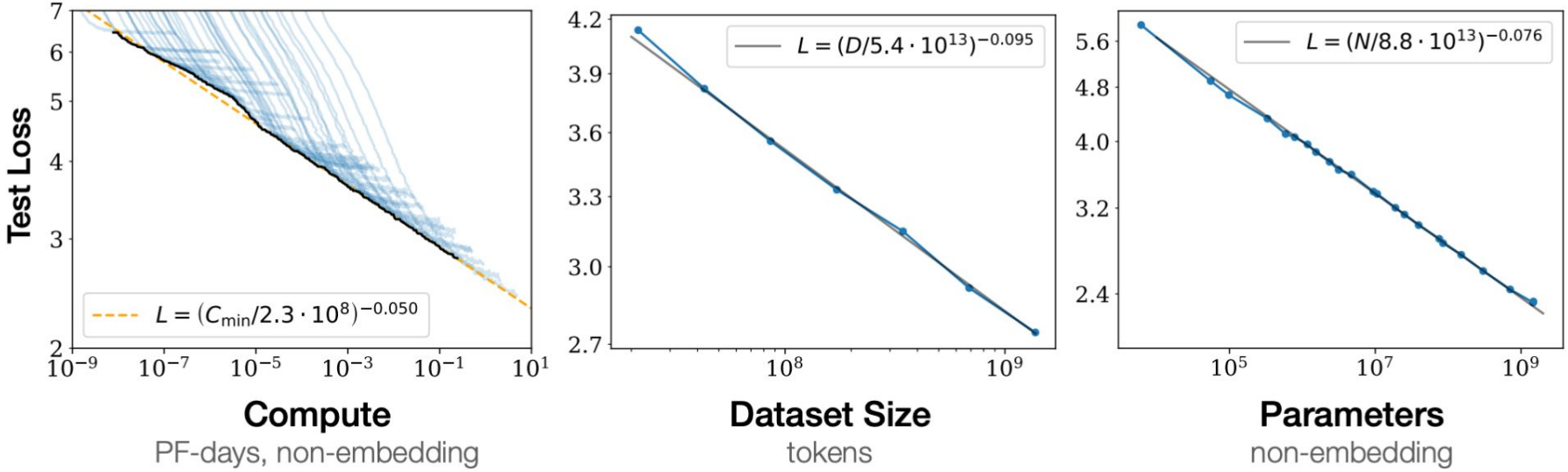


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

What is a power law?

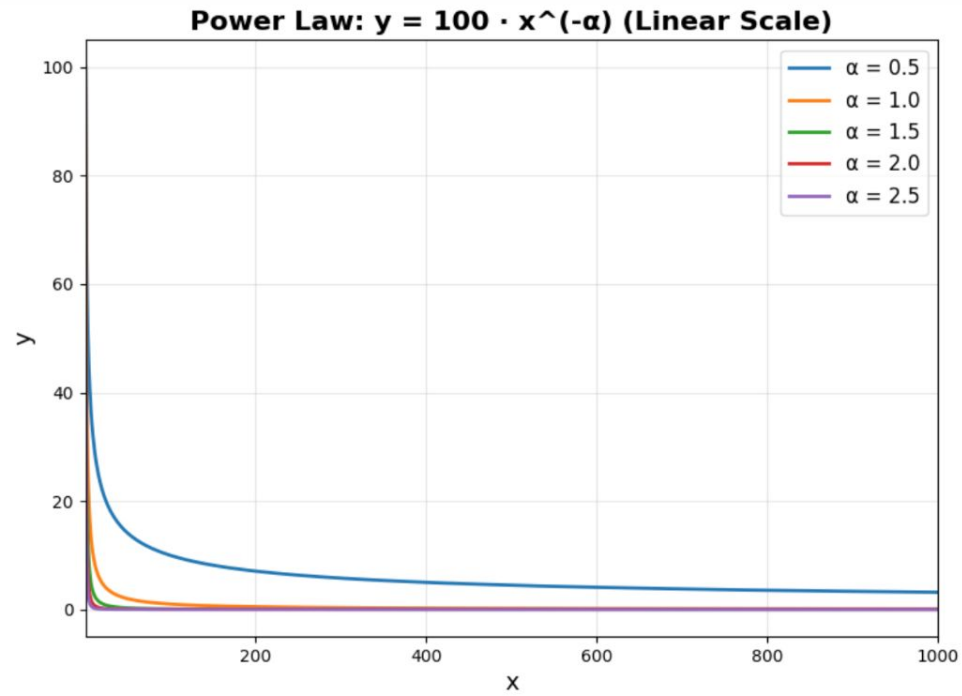
$Y = a x^k$

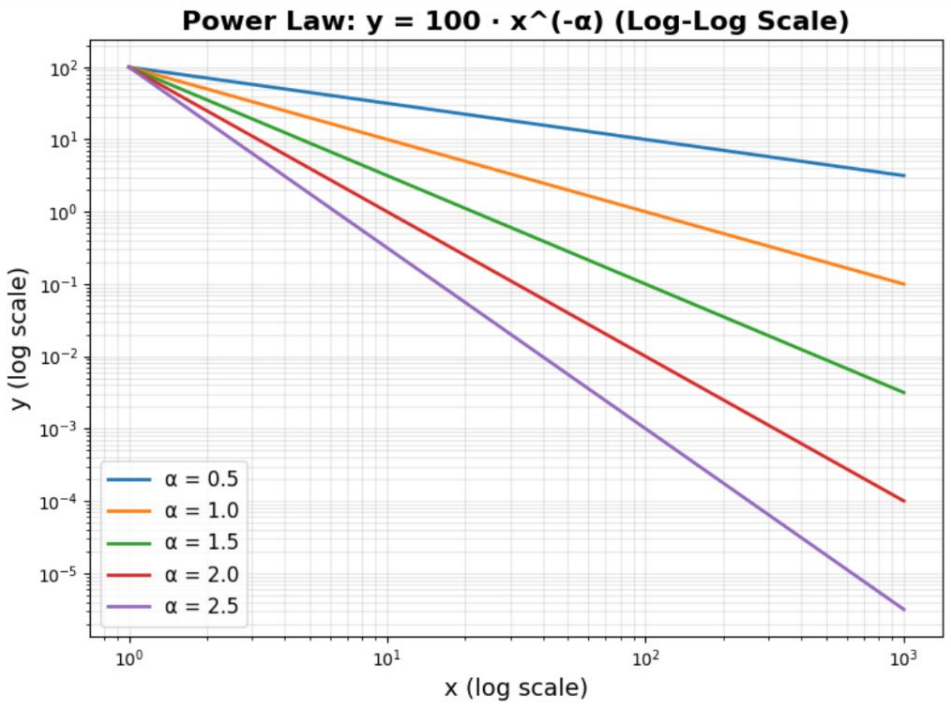
$\log(y)$

$= \log(ax^k)$

$= \log(a) + \log(x^k)$

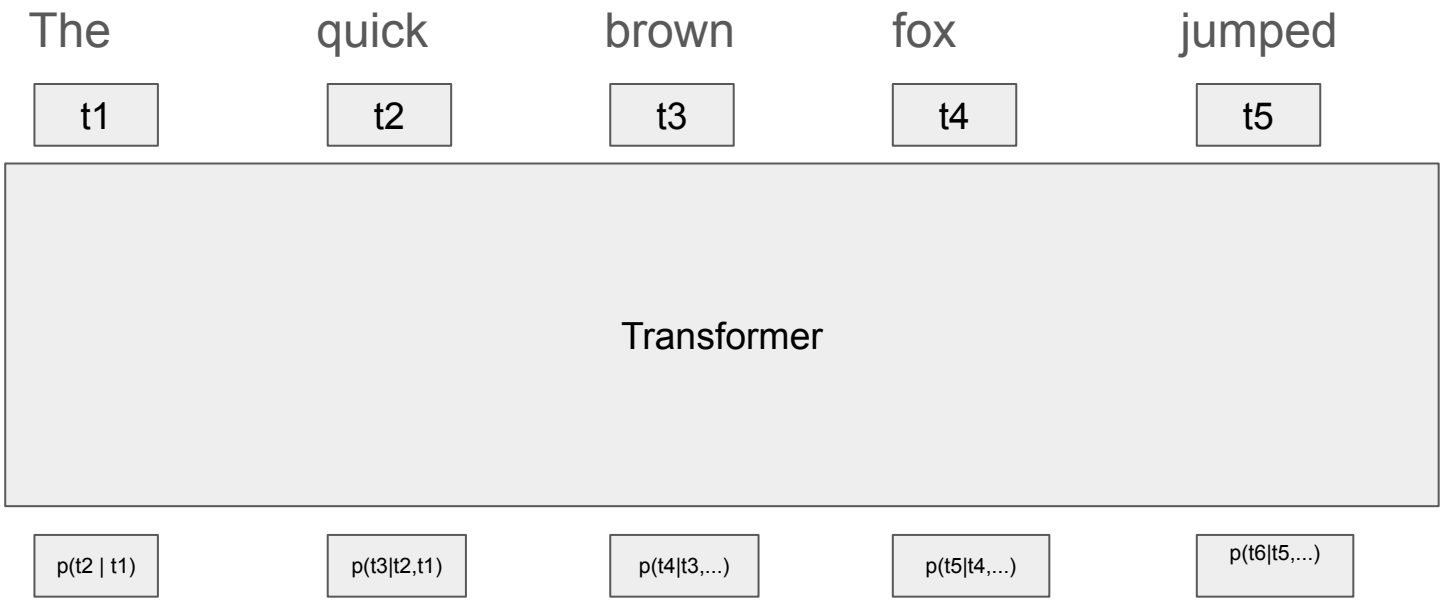
$= \log(a) + k\log(x)$ -> straight line on log scale


Power Law: y = 100 · x^(-α) (Linear Scale)
α = 0.5
α = 1.0
α = 1.5
α = 2.0
α = 2.5
y
x
0
20
40
60
80
100
200
400
600
800
1000


Power Law: y = 100 · x^(-α) (Log-Log Scale)
α = 0.5
α = 1.0
α = 1.5
α = 2.0
α = 2.5
y (log scale)
x (log scale)
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{0}
10^{1}
10^{2}
10^{3}

What is the experiment they're doing here?

We train language models on WebText2, an extended version of the WebText [RWC+19] dataset, tokenized using byte-pair encoding [SHB15] with a vocabulary size $n_{\text{vocab}} = 50257$. We optimize the autoregressive log-likelihood (i.e. cross-entropy loss) averaged over a 1024-token context, which is also our principal performance metric. We record the loss on the WebText2 test distribution and on a selection of other text distributions. We primarily train decoder-only [LSP+18, RNSS18] Transformer [VSP+17] models, though we also train LSTM models and Universal Transformers [DGV+18] for comparison.



Cross entropy loss:

$$H(p, q) = -\sum_{x \in classes} p(x) \log q(x)$$

True probability distribution

Your model's predicted probability distribution

Just becomes the log probability of the dataset

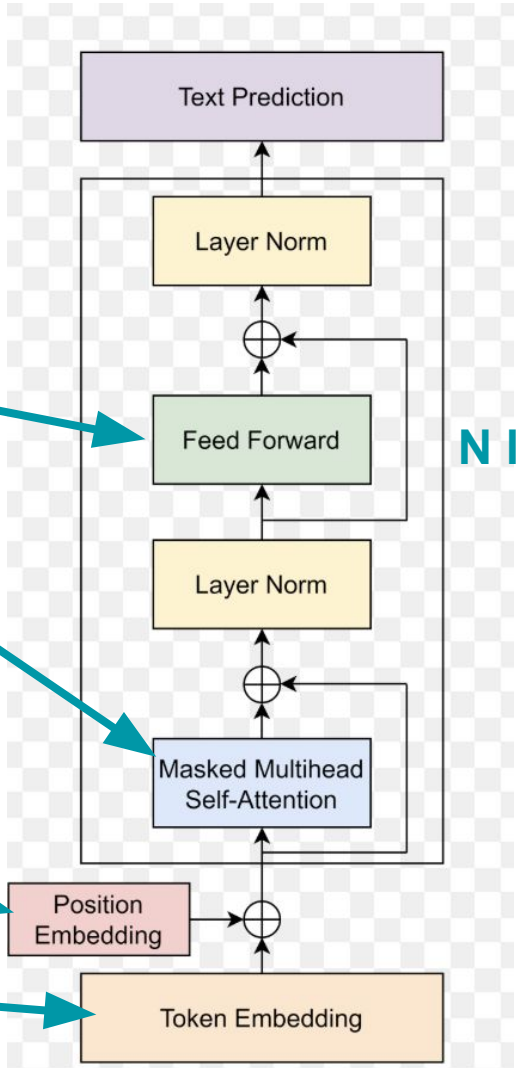
Loss = Avg(Log p(t2 | t1) , Log p(t3|t2,t1) , Log p(t4|t3,...) , Log p(t5|t4,...))

Reminder of transformer shapes

Weight Matrix	Shape	Parameter Count
W_1 (Up-proj)	$[d_{model}, d_{ff}]$	$d_{model} \times d_{ff}$
W_2 (Down-proj)	$[d_{ff}, d_{model}]$	$d_{ff} \times d_{model}$
Total FFN	(assuming $d_{ff} = 4d_{model}$)	$8 \times d_{model}^2$

Weight Matrix	Shape	Parameter Count
W_Q, W_K, W_V	$[d_{model}, d_{model}]$ each	$3 \times d_{model}^2$
W_O (Output)	$[d_{model}, d_{model}]$	d_{model}^2
Total Attention		$4 \times d_{model}^2$

Component	Shape	Parameter Count
Token Embeddings (W_e)	$[V, d_{model}]$	$V \times d_{model}$
Positional Embeddings (W_p)	$[T, d_{model}]$	$T \times d_{model}$



- L – the cross entropy loss in nats. Typically it will be averaged over the tokens in a context, but in some cases we report the loss for specific tokens within the context.
- N – the number of model parameters, *excluding all vocabulary and positional embeddings*
- $C \approx 6NBS$ – an estimate of the total non-embedding training compute, where B is the batch size, and S is the number of training steps (ie parameter updates). We quote numerical values in PF-days, where one PF-day $= 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ floating point operations.
- D – the dataset size in tokens

Scale over shape

Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width. (Section 3)

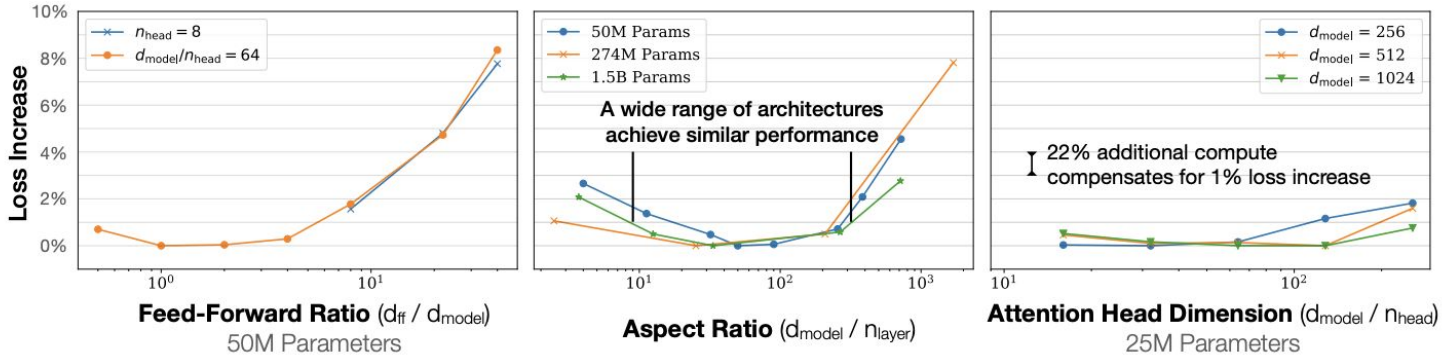


Figure 5 Performance depends very mildly on model shape when the total number of non-embedding parameters N is held fixed. The loss varies only a few percent over a wide range of shapes. Small differences in parameter counts are compensated for by using the fit to $L(N)$ as a baseline. Aspect ratio in particular can vary by a factor of 40 while only slightly impacting performance; an $(n_{\text{layer}}, d_{\text{model}}) = (6, 4288)$ reaches a loss within 3% of the $(48, 1600)$ model used in [RWC+19].

Scale over Data

Universality of overfitting: Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. The performance penalty depends predictably on the ratio $N^{0.74}/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty. (Section 4)

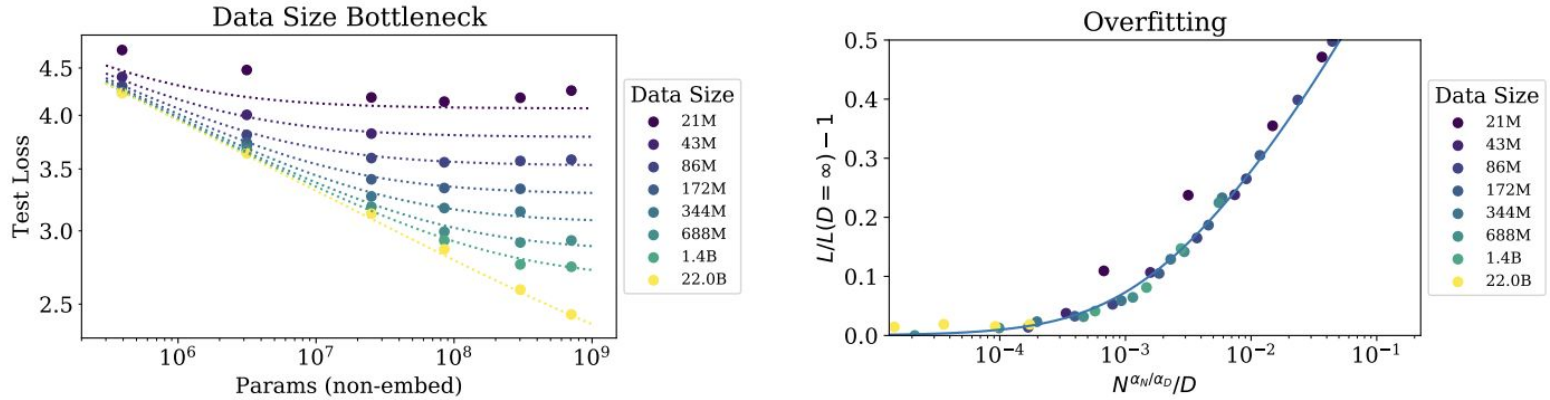


Figure 9 The early-stopped test loss $L(N, D)$ depends predictably on the dataset size D and model size N according to Equation (1.5). **Left**: For large D, performance is a straight power law in N. For a smaller fixed D, performance stops improving as N increases and the model begins to overfit. (The reverse is also true, see Figure 4.) **Right**: The extent of overfitting depends predominantly on the ratio $N^{\frac{\alpha_N}{\alpha_D}}/D$, as predicted in equation (4.3). The line is our fit to that equation.

Beginnings of transfer learning

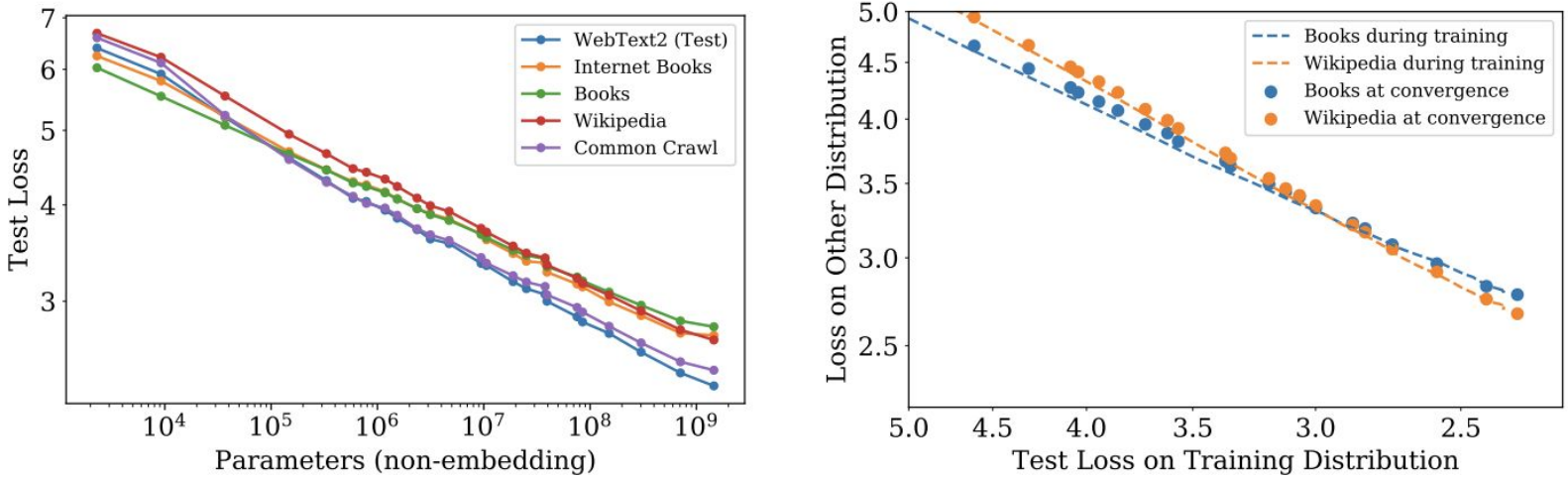


Figure 8 **Left:** Generalization performance to other data distributions improves smoothly with model size, with only a small and very slowly growing offset from the WebText2 training distribution. **Right:** Generalization performance depends only on training distribution performance, and not on the phase of training. We compare generalization of converged models (points) to that of a single large model (dashed curves) as it trains.

Optimal Batch Size

A simple empirical theory for the batch size dependence of training was developed in [MKAT18] (see also [SLA+18, ZLN+19]). It was argued that there is a critical batch size B_{crit} for training; for B up to B_{crit} the batch size can be increased with very minimal degradation in compute-efficiency, whereas for $B > B_{\text{crit}}$ increases in B result in diminishing returns. It was also argued that the gradient noise scale provides a simple prediction for B_{crit}, and that neither depends directly on model size except through the value of the loss that has been attained. These results can be used to predict how training time and compute will vary with the batch size. To utilize both training time and compute as effectively as possible, it is best to train with a batch size $B \approx B_{\text{crit}}$. Training at $B \gg B_{\text{crit}}$ minimizes the number of training steps, while $B \ll B_{\text{crit}}$ minimizes the use of compute.

Optimal Batch Size

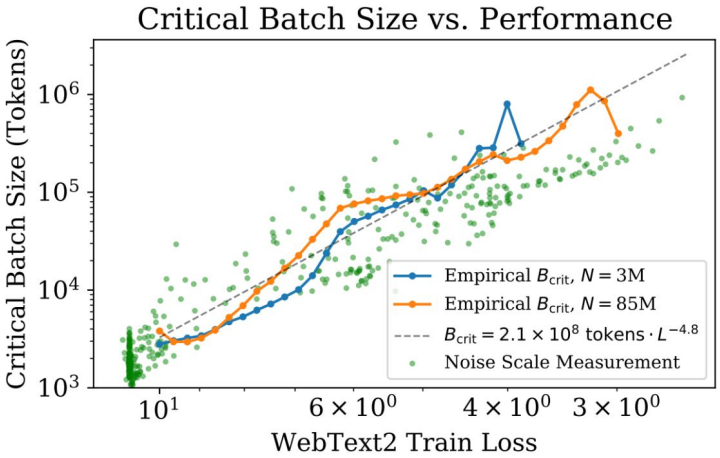


Figure 10 The critical batch size B_{crit} follows a power law in the loss as performance increase, and does not depend directly on the model size. We find that the critical batch size approximately doubles for every 13% decrease in loss. B_{crit} is measured empirically from the data shown in Figure 18, but it is also roughly predicted by the gradient noise scale, as in [MKAT18].

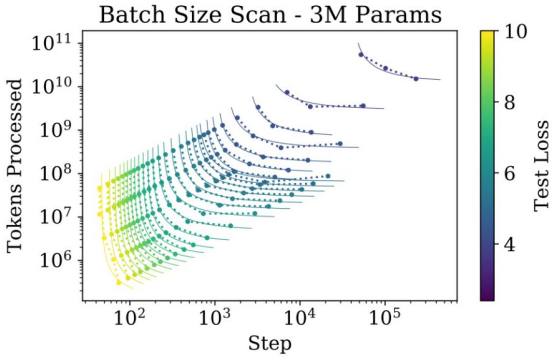
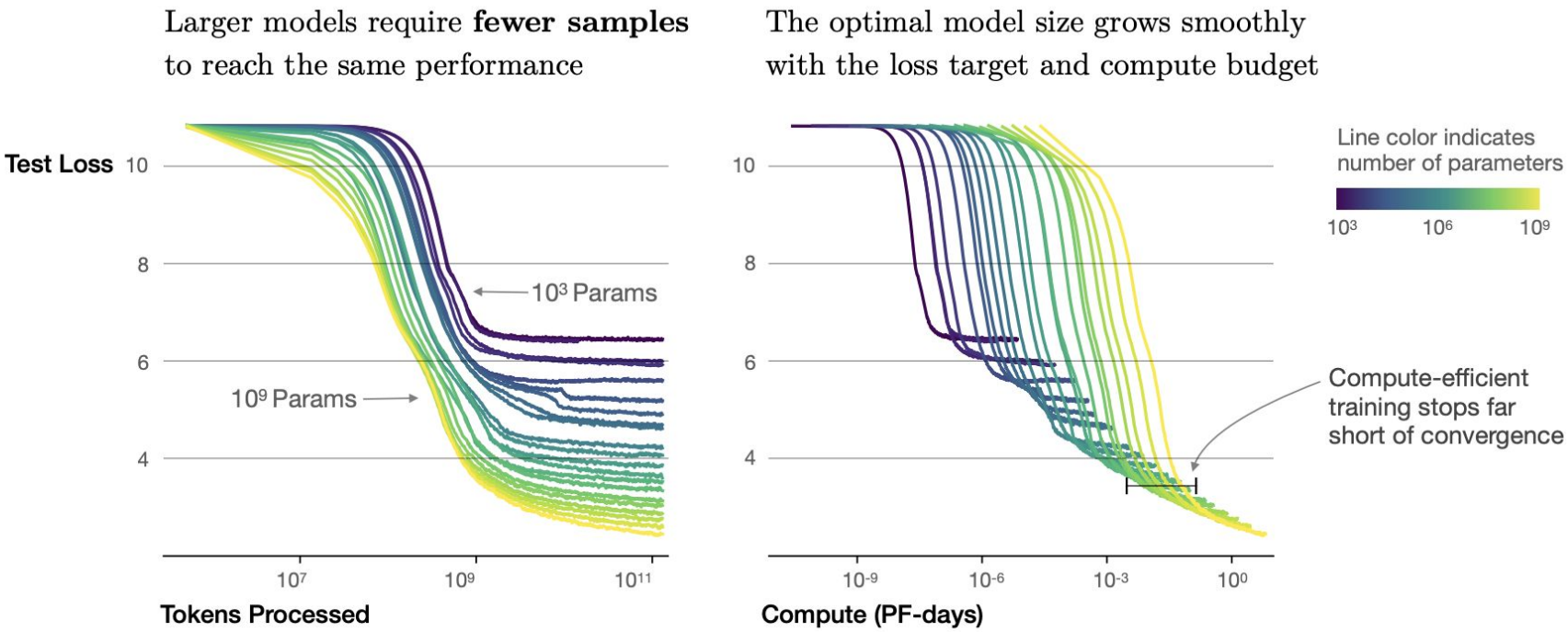


Figure 18 These figures demonstrate fits to Equation (5.1) for a large number of values of the loss L, and for two different Transformer model sizes. These fits were used to measure $B_{\text{crit}}(L)$ for Figure 10.

Convergence is inefficient



(Aside) Chinchilla

Timeli

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

GPT-1
Jun 2018

GPT-2
Feb 2019

GPT-3
Jun 2020

GPT-3.5
Nov 2022

GPT-4
Mar 2023

GPT-4o
May 2024

GPT-5
Aug 2025

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jared@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

2018 2019 2020 2021 2022 2023 2024 2025 2026

What did Kaplan do wrong?

Kaplan:

1. Fixed model size, increased data, observed diminishing returns
2. Fixed data size, increase params, observed slower diminishing returns
3. Fixed both model and data, observed optimal batch size and compute

From (1) and (2) he concluded that you should scale params more than data. But in doing this you are inherently using more compute!

What he didn’t do:

- Fixed compute: Compared optimal ratio of model size to dataset size

(Aside) Chinchilla

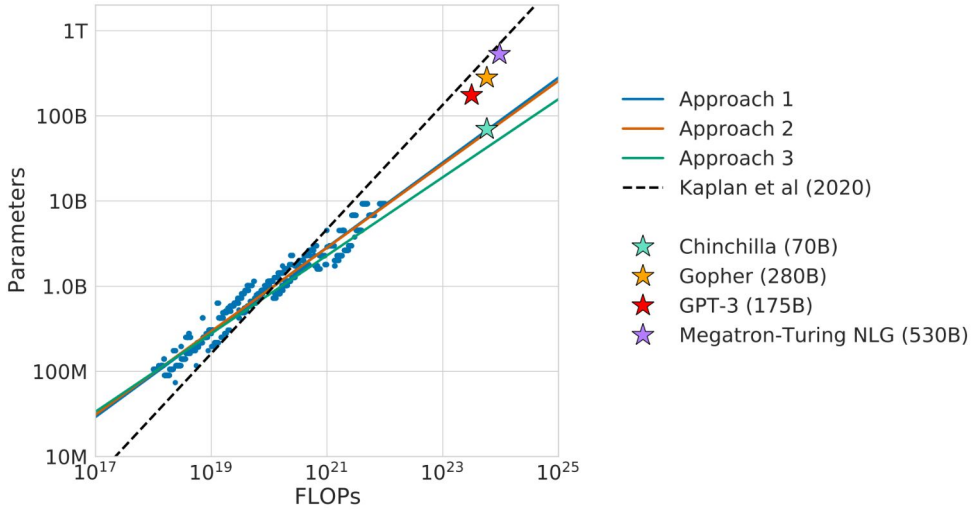
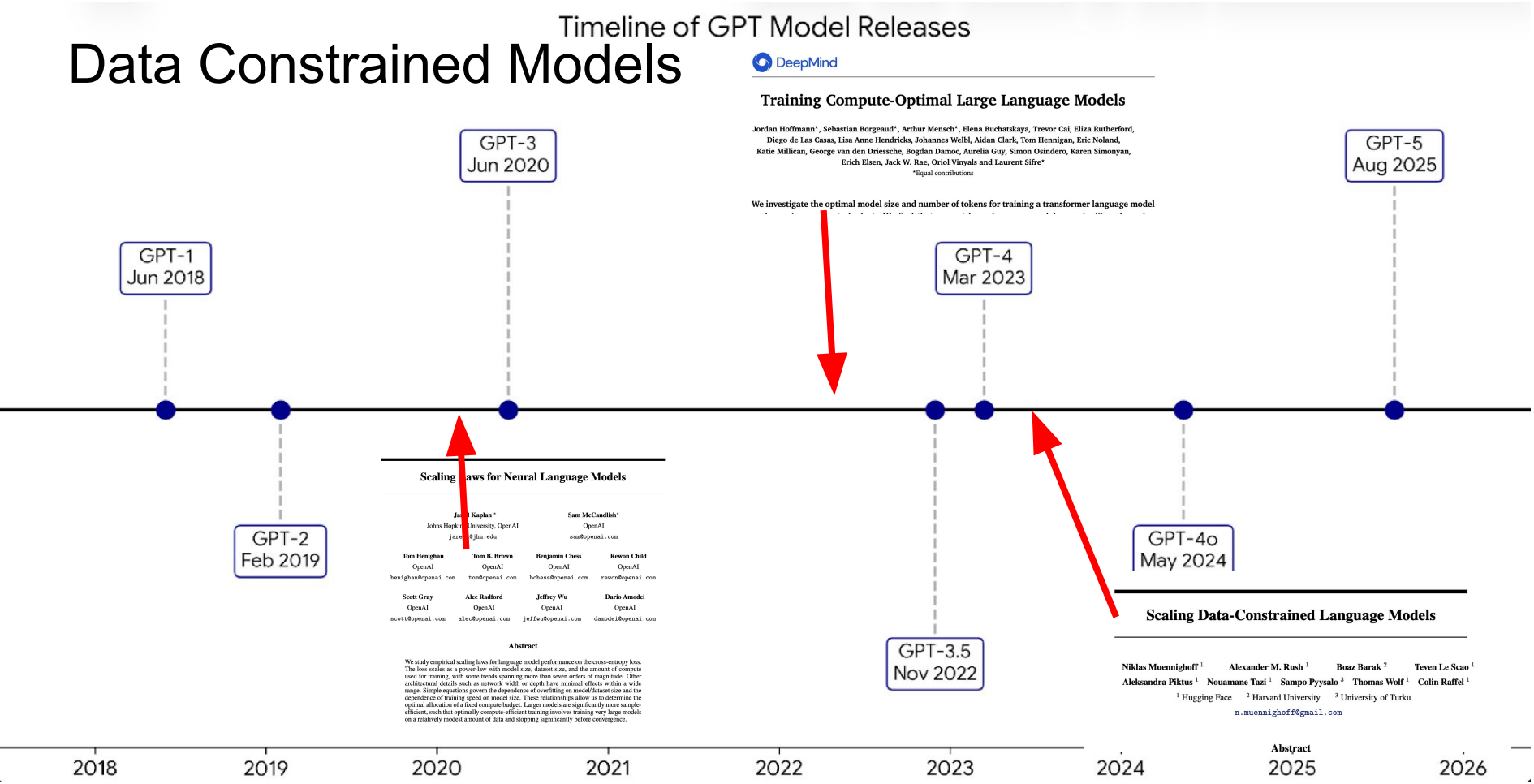
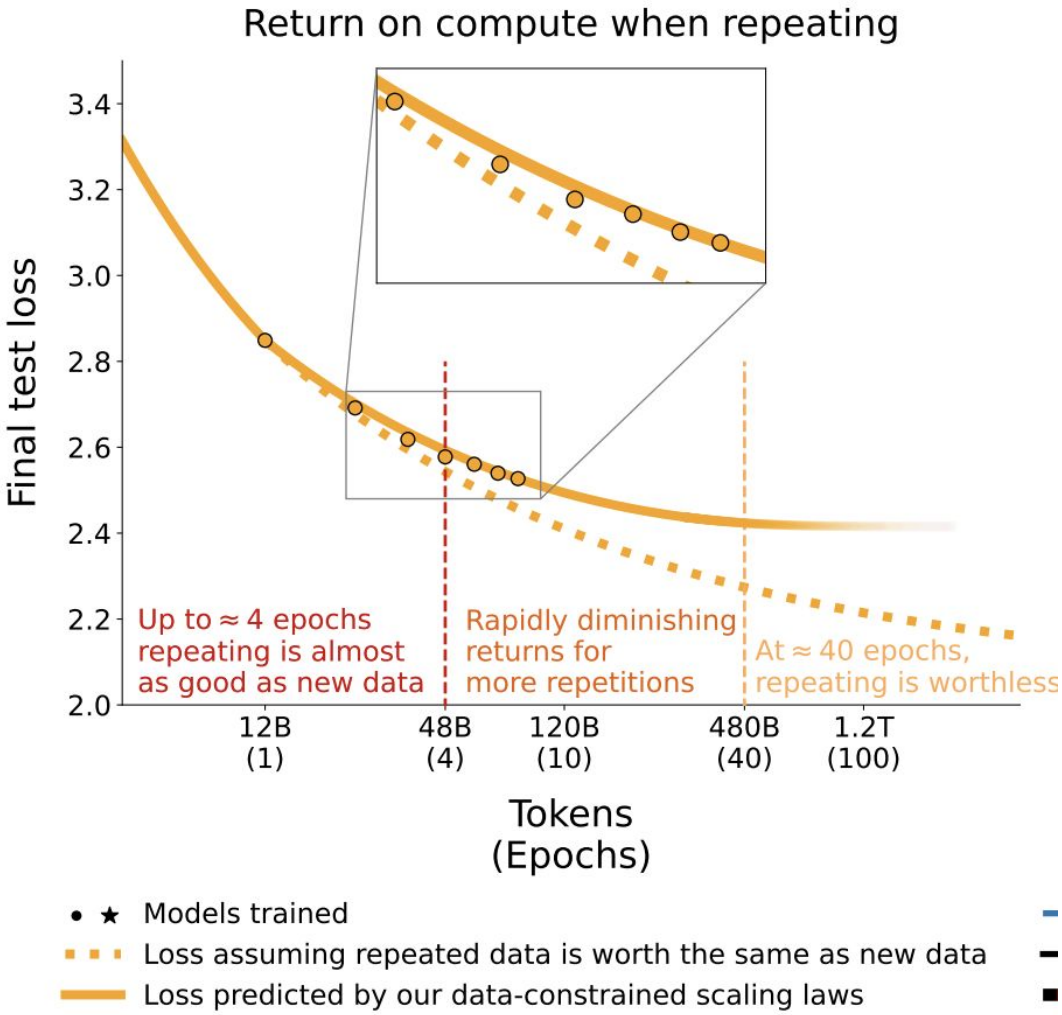
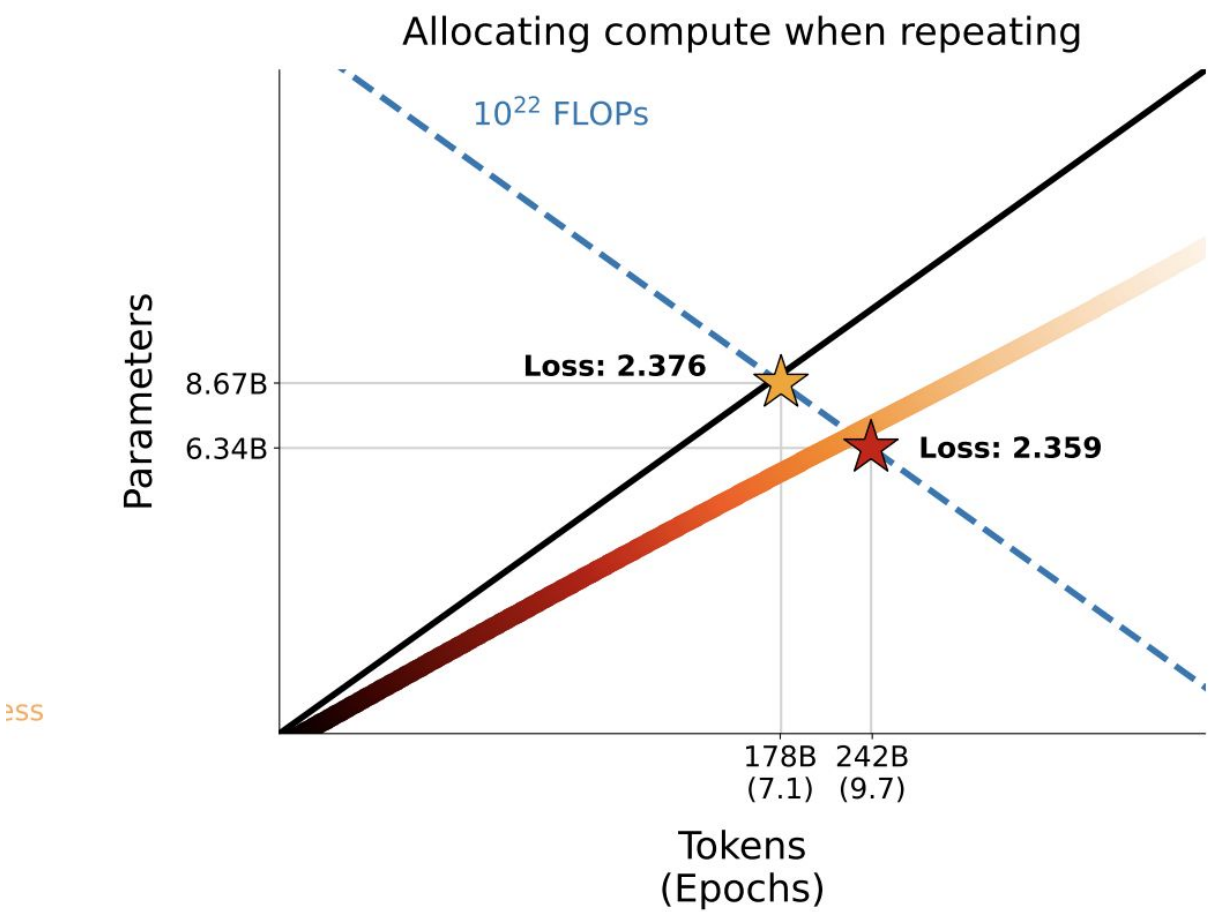


Figure 1 | **Overlaid predictions.** We overlay the predictions from our three different approaches, along with projections from Kaplan et al. (2020). We find that all three methods predict that current large models should be substantially smaller and therefore trained much longer than is currently done. In Figure A3, we show the results with the predicted optimal tokens plotted against the optimal number of parameters for fixed FLOP budgets. ***Chinchilla* outperforms *Gopher* and the other large models (see Section 4.2).**

Data Constrained Models




Return on compute when repeating
Final test loss
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
Up to ≈ 4 epochs repeating is almost as good as new data
Rapidly diminishing returns for more repetitions
At ≈ 40 epochs, repeating is worthless
12B (1)
48B (4)
120B (10)
480B (40)
1.2T (100)
Tokens (Epochs)
Models trained
Loss assuming repeated data is worth the same as new data
Loss predicted by our data-constrained scaling laws


Allocating compute when repeating
10^{22} FLOPs
Loss: 2.376
Loss: 2.359
Parameters
8.67B
6.34B
178B (7.1)
242B (9.7)
Tokens (Epochs)
Regime of same compute (IsoFLOP)
Efficient frontier assuming repeated data is worth the same as new data
Efficient frontier predicted by our data-constrained scaling laws

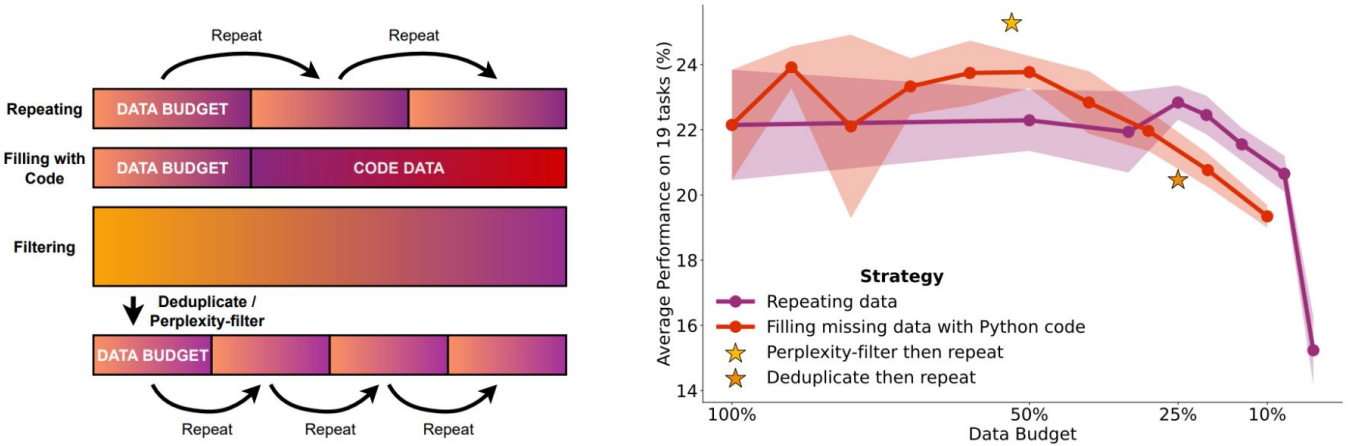
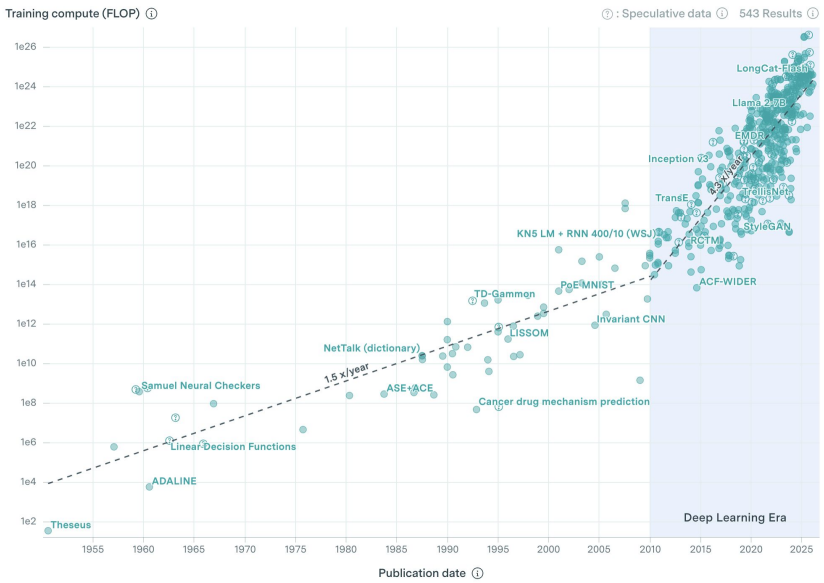
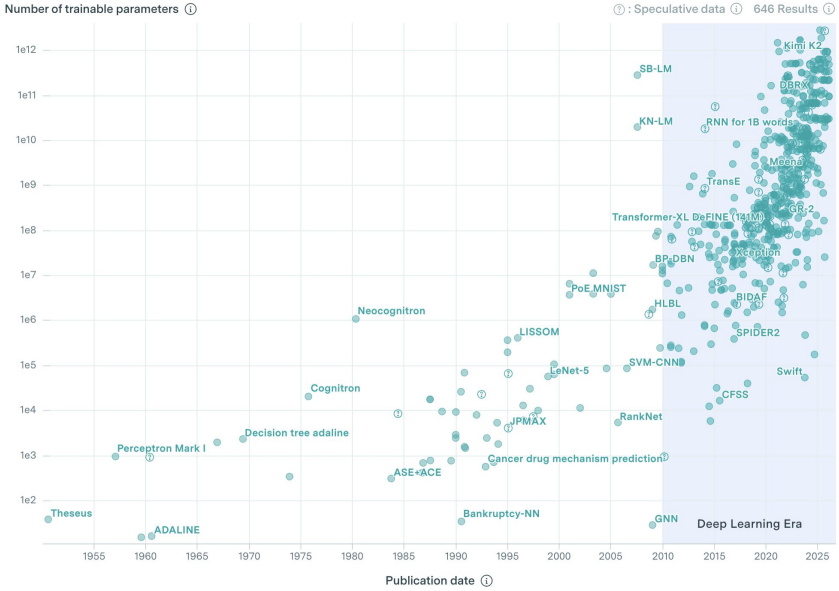


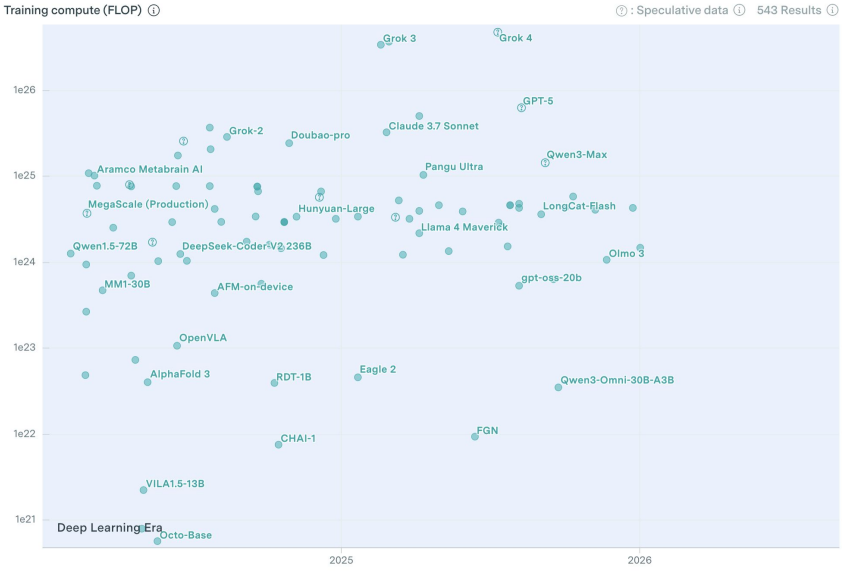
Figure 6: **Strategies for data-constrained settings and their downstream performance.** *(Left):* Schematic showing alternative data use strategies of code filling and filtering. *(Right):* $N = 4.2$ billion parameter models trained for a total of $D = 84$ billion tokens with varying budgets D_C. For repeating and filling with code, five models with different seeds are trained for each dot and the standard deviation is visualized as the shaded area.

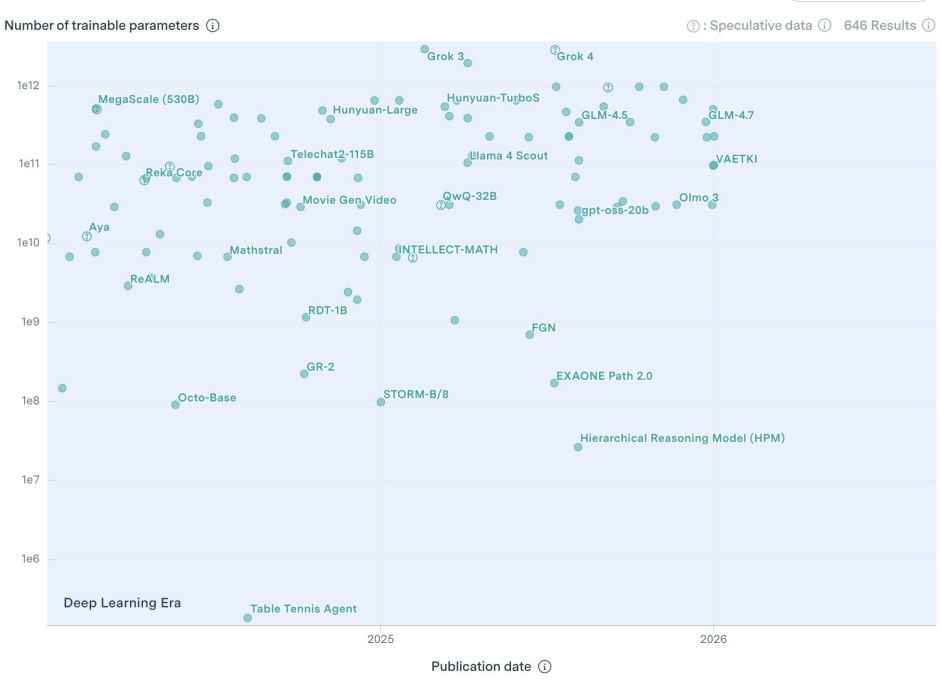
Why have model sizes slowed?





Why have model sizes slowed?





Why have model sizes slowed?

- We stopped knowing how big frontier models are
- Mixture of Expert models emerged
- We maxxed out the amount of pre-training data
- Post-training emerged
- **There is a new optimisation metric: Cost of Inference**
- Test time scaling

Practicalities

- We almost always start from a pretrained model
- We almost always are data constrained
- Bigger models adapt faster to new data
- We are constrained by maximum concurrent GPU mem, not so much by total FLOPS

-> Choose the largest model that you can fit into VRAM and do at least 4 epochs.

If you are lucky enough that 4 epochs and a big model takes > a few days to train, start doing scaling experiments :) Chances are you'd be better filtering your data

Future

- What does scaling look like for RLVR?