

FALL 2024 COS597R: DEEP DIVE INTO LARGE LANGUAGE MODELS

Danqi Chen, Sanjeev Arora



Lecture 4: Scaling Laws for Pre-training

<https://princeton-cos597r.github.io/>

The Bitter Lesson

Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great. The two methods that seem to scale arbitrarily in this way are *search* and *learning*.

Scaling up Deep Learning (history)

Simple question

Suppose you take a deep net and you multiply its size by C_1 and its dataset size by C_2 .

By how much does the compute requirement (in FLOPs) increase?

FLOP= Floating point operation (addition/multiplication/ division)

Scaling up Deep Learning

ConvNets drove initial successes, esp for vision datasets (CIFAR, ImageNet etc.)

What's a recipe to scale them up for arbitrary image tasks?

[Tan & Le '19] “Efficient (Conv)-Nets” : If you have 2^N factor more compute, scale up width, depth, image-size by $\alpha^N, \beta^N, \gamma^N$ where α, β, γ are determined by grid search on smaller conv-nets for the same task

Compute requirement for forward pass (transformer)

- Embeddings
 - $2 \times \text{seq_len} \times \text{vocab_size} \times \text{d_model}$ (Factor 2 for multiply accumulate)
- Attention (Single Layer)
 - **Key, query and value projections:** $2 \times 3 \times \text{seq_len} \times \text{d_model} \times (\text{key_size} \times \text{num_heads})$
 - **Key @ Query logits:** $2 \times \text{seq_len} \times \text{seq_len} \times (\text{key_size} \times \text{num_heads})$
 - **Softmax:** $3 \times \text{num_heads} \times \text{seq_len} \times \text{seq_len}$
 - **Softmax @ query reductions:** $2 \times \text{seq_len} \times \text{seq_len} \times (\text{key_size} \times \text{num_heads})$
 - **Final Linear:** $2 \times \text{seq_len} \times (\text{key_size} \times \text{num_heads}) \times \text{d_model}$
- Dense Block (Single Layer)
 - $2 \times \text{seq_len} \times (\text{d_model} \times \text{ffw_size} + \text{d_model} \times \text{ffw_size})$
- Final Logits
 - $2 \times \text{seq_len} \times \text{d_model} \times \text{vocab_size}$

Total forward pass FLOPs: $\text{embeddings}, \text{num_layers}^1 \text{total_attention}, \text{dense_block}^o + \text{logits}$
(In [Kaplan et al'20] approximated as $6ND$; $N = \# \text{parameters}$, $D = \# \text{tokens}$)

Jared Kaplan *

Johns Hopkins University, OpenAI

Sam McCandlish*

OpenAI

et al, 2021

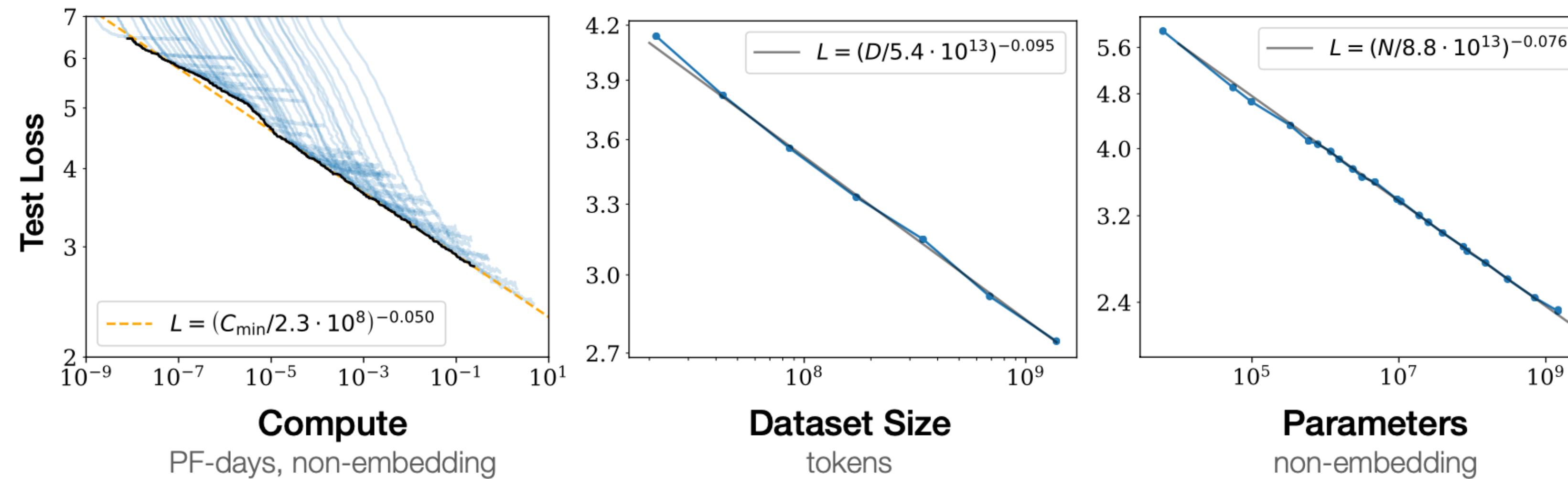


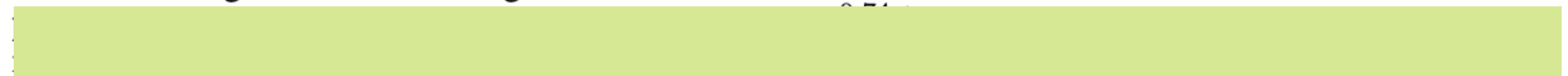
Figure 1 Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute² used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Claims that aged relatively well...

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)

Smooth power laws: Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude (see Figure 1). We observe no signs of deviation from these trends on the upper end, though performance must flatten out eventually before reaching zero loss. (Section 3)

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



Universality of training: Training curves follow predictable power-laws whose parameters are roughly independent of the model size. By extrapolating the early part of a training curve, we can roughly predict the loss that would be achieved if we trained for much longer. (Section 5)

Transfer improves with test performance: When we evaluate models on text with a different distribution than they were trained on, the results are strongly correlated to those on the training validation set with a roughly constant offset in the loss – in other words, transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set. (Section 3.2.2)

Claims that did not....

Sample efficiency: Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps (Figure 2) and using fewer data points (Figure 4).

Convergence is inefficient: When working within a fixed compute budget C but without any other restrictions on the model size N or available data D , we attain optimal performance by training *very large models* and stopping *significantly short of convergence* (see Figure 3). Maximally compute-efficient training would therefore be far more sample efficient than one might expect based on training small models to convergence, with data requirements growing very slowly as $D \sim C^{0.27}$ with training compute. (Section 6)

Optimal batch size: The ideal batch size for training these models is roughly a power of the loss only, and continues to be determinable by measuring the gradient noise scale [MKAT18]; it is roughly 1-2 million tokens at convergence for the largest models we can train. (Section 5.1)

10 \times increase in compute should be allocated to a 5.5 \times increase in model size and a 1.8 \times increase in training tokens.

Brief Era of Undertrained Mega Models (2020-22)

Implication of Kaplan et al. [2020] : 10 \times increase in compute should be allocated to a 5.5 \times increase in model size and a 1.8 \times increase in training tokens.”

Gopher [Rae et al'21, Google]

280B parameters, 300B tokens...

Scaling Language Models: Methods, Analysis & Insights from Training *Gopher*

Model	Layers	Number Heads	Key/Value Size	d_{model}	Max LR	Batch Size
44M	8	16	32	512	6×10^{-4}	0.25M
117M	12	12	64	768	6×10^{-4}	0.25M
417M	12	12	128	1,536	2×10^{-4}	0.25M
1.4B	24	16	128	2,048	2×10^{-4}	0.25M
7.1B	32	32	128	4,096	1.2×10^{-4}	2M
<i>Gopher</i> 280B	80	128	128	16,384	4×10^{-5}	3M → 6M

Table 1 | **Model architecture details.** For each model, we list the number of layers, the key/value size, the bottleneck activation size d_{model} , the maximum learning rate, and the batch size. The feed-forward size is always $4 \times d_{\text{model}}$.

PaLM [Choudhery et al'22]

PaLM: Scaling Language Modeling with Pathways
[PaLM2] was a followup

540 B parameters; 780B tokens

Design tailored for parallelization
in TPU v4 pod client-server
architecture (Pathways)

Later stages use bigger
batch sizes for better
gradient estimate (less noise)

Model	# of Parameters (in billions)	Accelerator chips	Model FLOPS utilization
GPT-3	175B	V100	21.3%
Gopher	280B	4096 TPU v3	32.5%
Megatron-Turing NLG	530B	2240 A100	30.2%
PaLM	540B	6144 TPU v4	46.2%

Model	Layers	# of Heads	d_{model}	# of Parameters (in billions)	Batch Size
PaLM 8B	32	16	4096	8.63	256 → 512
PaLM 62B	64	32	8192	62.50	512 → 1024
PaLM 540B	118	48	18432	540.35	512 → 1024 → 2048

Table 1: Model architecture details. We list the number of layers, d_{model} , the number of attention heads and attention head size. The feed-forward size d_{ff} is always $4 \times d_{\text{model}}$ and attention head size is always 256.

Deterministic batches; “fully bitwise reproducible from any checkpoint”.

PaLM (the hardware)

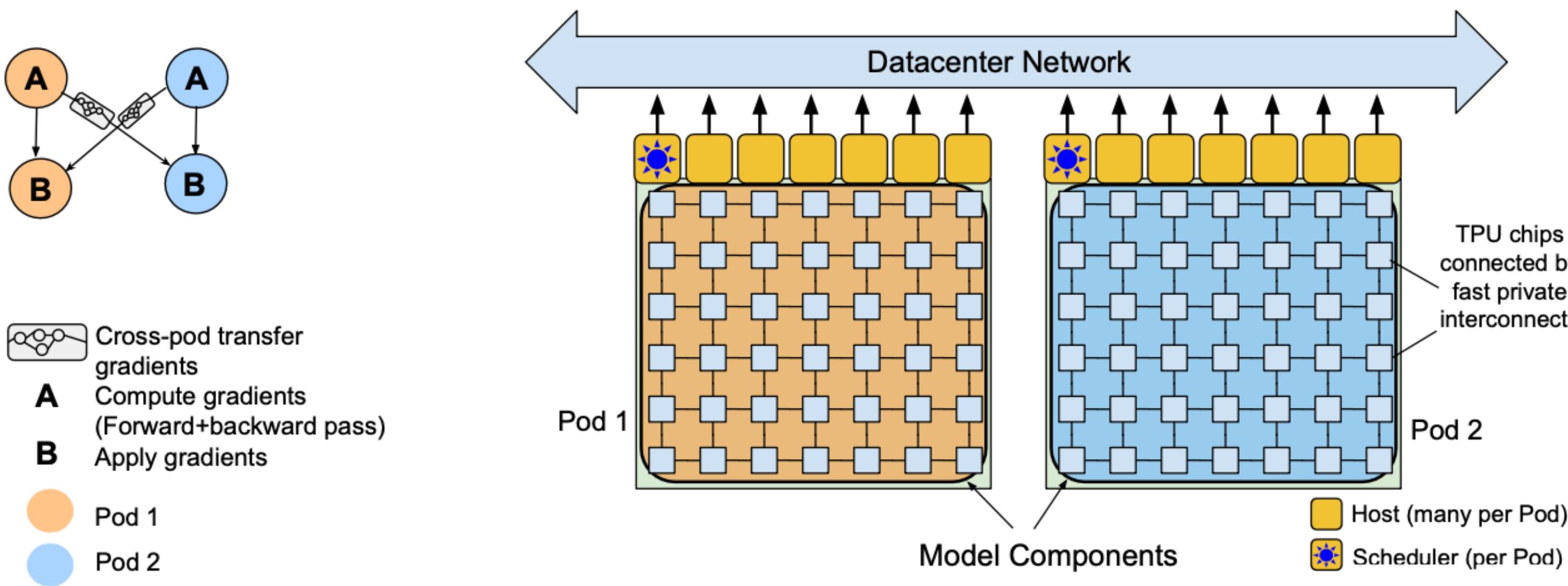


Figure 2: The Pathways system (Barham et al., 2022) scales training across two TPU v4 pods using two-way data parallelism at the pod level.

Figure 2 shows how the Pathways system executes the two-way pod-level data parallelism. A single Python client constructs a sharded dataflow program (shown on the left in Figure 2) that launches JAX/XLA (XLA, 2019) work on remote servers that each comprise a TPU pod. The program contains a component A for within-pod forward+backward computation (including within-pod gradient reduction), transfer subgraph for cross-pod gradient transfer, and a component B for optimizer update (including summation of local and remote gradients).

Megatron Turing NLG

(Nvidia, 2022)

530B parameters, 270B tokens

monolithic (unlike Google PaLM, PaLM2); served to highlight Nvidia's own parallelism solution (NVLink within a node, InfiniBand across nodes)*

In hindsight, a fairly unexceptional effort....

By combining tensor-slicing and pipeline parallelism, we can operate them within the regime where they are most effective. More specifically, the system uses tensor-slicing from Megatron-LM to scale the model within a node and uses pipeline parallelism from DeepSpeed to scale the model across nodes.

Deepmind's effort at finding Scaling Laws



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*

“Chinchilla paper”

“Compute optimal” : Best heldout cross-entropy given total FLOPs budget

- No constraints on # of GPUs and # Tokens
- Ignores communication latencies

$$N_{opt}(C), D_{opt}(C) = \underset{N, D \text{ s.t. } \text{FLOPs}(N, D)=C}{\operatorname{argmin}} L(N, D).$$

Caveat: Minimization
only over architectures,
training, and datasets
that were popular in ‘22

Experiments: 400 models, sizes 70M to 16B

Dataset size 5B to 500B

(other hyper-parameters such as batch size, dimension, etc taken from earlier studies)

“Compute optimal” : Best heldout cross-entropy given total FLOPs budget

- No constraints on # of GPUs and # Tokens
- Ignores communication latencies

$$N_{opt}(C), D_{opt}(C) = \underset{N, D \text{ s.t. } \text{FLOPs}(N, D)=C}{\operatorname{argmin}} L(N, D).$$

Caveat: Minimization
only over architectures,
training, and datasets
that were popular in ‘22

Let's figure out: If $L(N, D) = 2 + \frac{400}{N^{1/3}} + \frac{2000}{D^{1/3}}$ what is the correct scaling recipe?

Table: Scalina Recipe

Parameters	FLOPs	FLOPs (in <i>Gopher</i> unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

“Chinchilla Scaling Law”

($D \approx 20N$ is compute-optimal choice)

Main finding

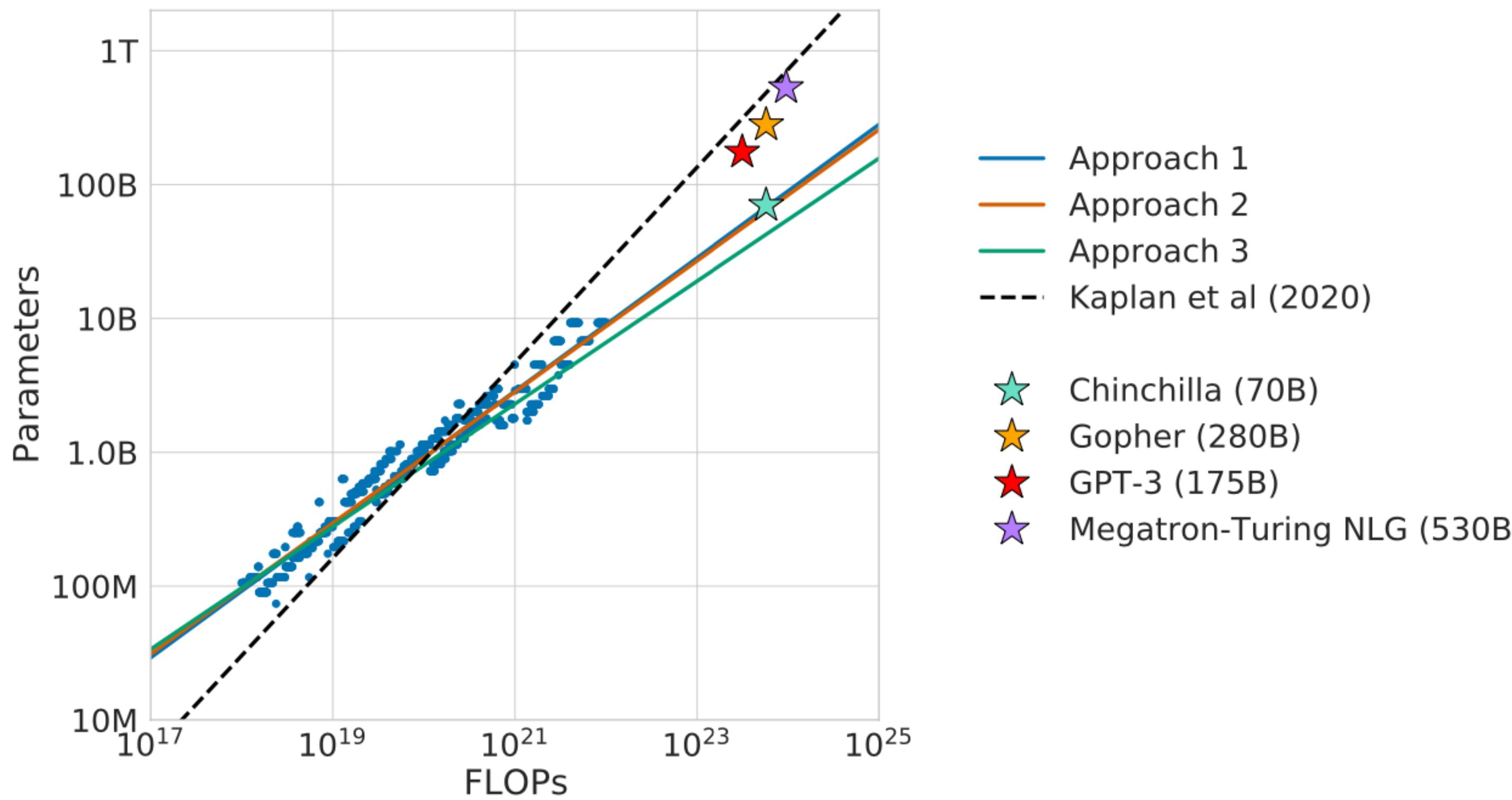


Figure 1 | Overlaid predictions. We overlay the predictions from our three different approaches, along with projections from Kaplan et al. (2020). We find that all three methods predict that current large models should be substantially smaller and therefore trained much longer than is currently done. 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).**

Side-benefit:
Compute optimal
models =>
faster inference

Recap: Cosine LR schedule

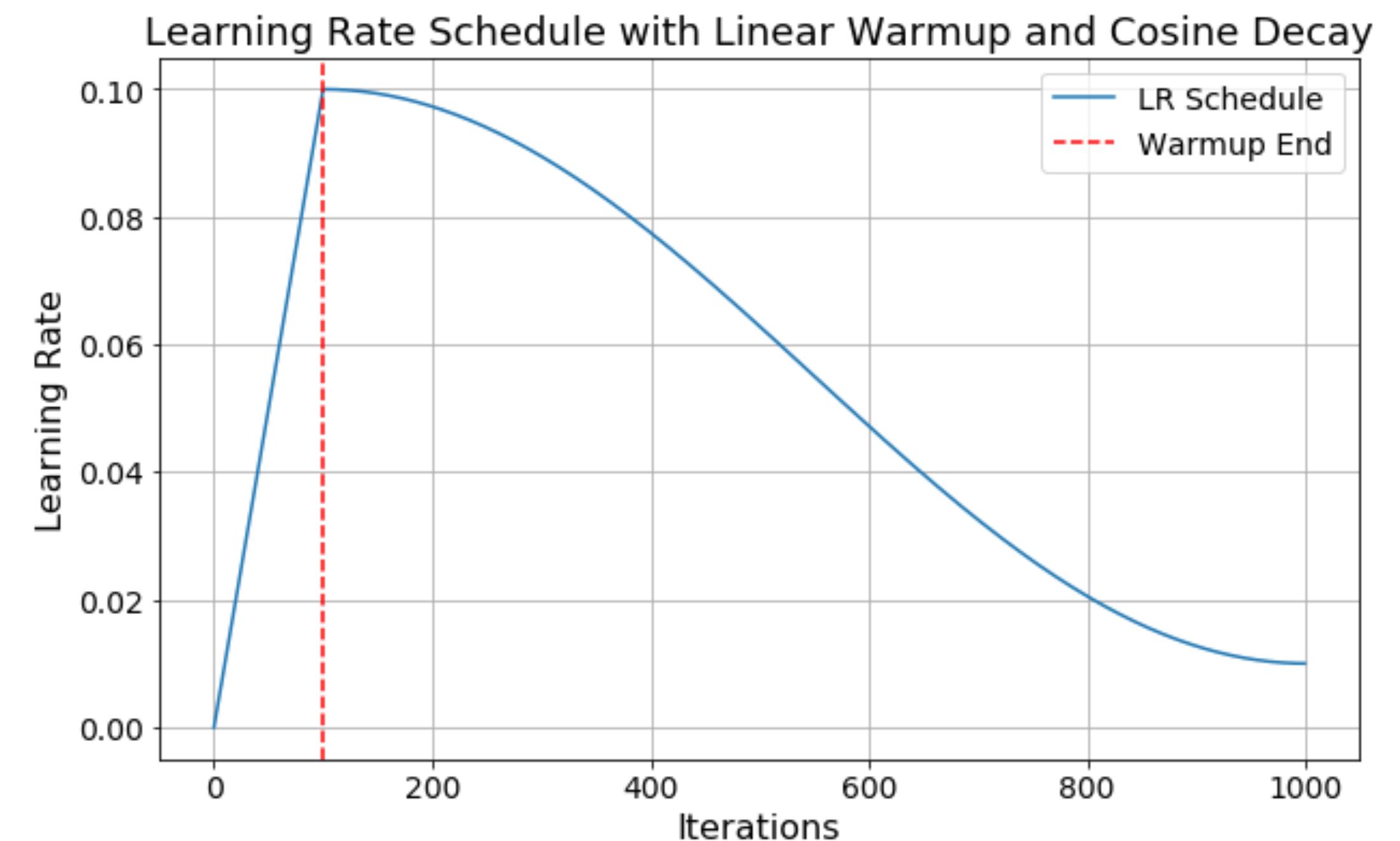
$$\text{LR}(t) = \ell + \frac{1}{2} (L - \ell) \left(1 + \cos \left(\frac{(t - t_w)\pi}{(T - t_w)} \right) \right).$$

L = Max LR

ℓ = min LR

T = total # of iterations

t_w = # of warmup iterations



Key Finding: When # of tokens (hence T) changes, use LR schedule for this new T

(DON'T finish training before hitting the end of the cosine schedule.)

This partly explains why OpenAI's Scaling Law [Hoffman et al'20] was off..

Finding how loss scales with compute and data

IsoFlop Curves

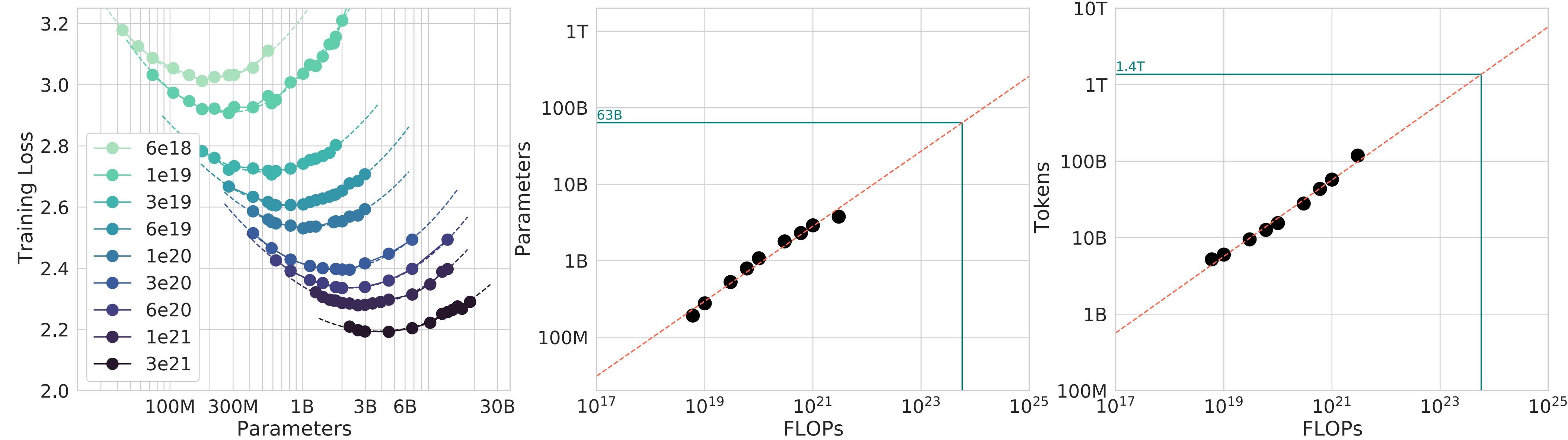


Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (left). Using the location of these valleys, we project optimal model size and number of tokens for larger models (center and right). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

Qs for class: What functional form does Fig 3 imply for scaling #params and #tokens?

Method to find Scaling Law

$N = \# \text{ parameters}$ $D = \# \text{ tokens}$ $C = \text{total compute}$

$$N_{opt}(C), D_{opt}(C) = \underset{N, D \text{ s.t. FLOPs}(N, D)=C}{\operatorname{argmin}} L(N, D).$$

Empirical finding from prev. slide : $N = KC^\alpha, D = K^{-1}C^\beta$ for
some α, β (functional form confirmed by all 3 approaches..)

Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

Estimated held-out c-e loss given D, N

$$L(N, D) = E + \frac{A}{N^{0.34}} + \frac{B}{D^{0.28}},$$

with $E = 1.69$, $A = 406.4$, $B = 410.7$.

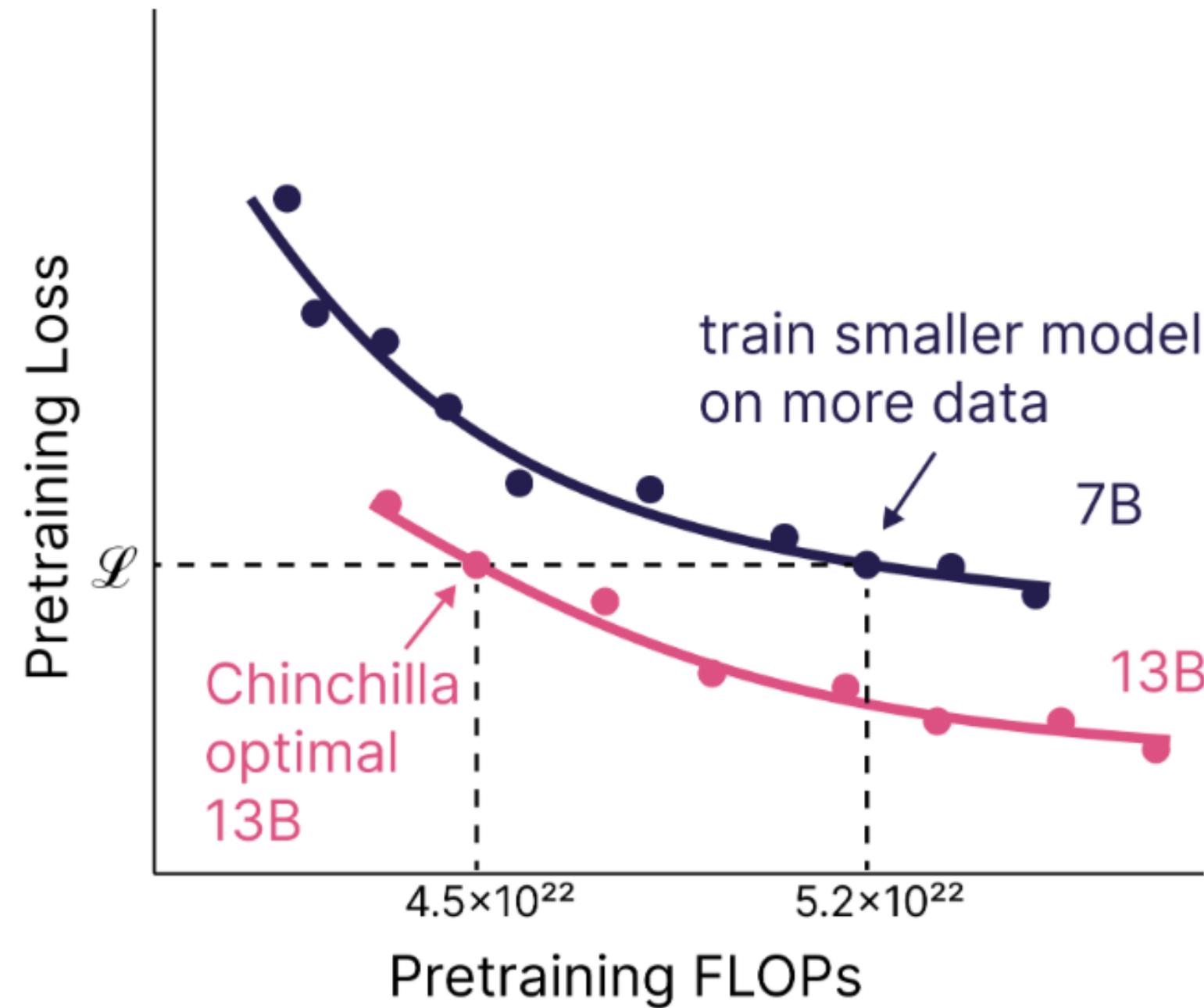
Fitting the decomposition to data. We effectively minimize the following problem

$$\min_{a,b,e,\alpha,\beta} \sum_{\text{Run } i} \text{Huber}_\delta \left(\text{LSE}(a - \alpha \log N_i, b - \beta \log D_i, e) - \log L_i \right),$$

where LSE is the log-sum-exp operator. We then set $A, B, E = \exp(a), \exp(b), \exp(e)$.

“Chinchilla Law”: Amended

v1: Accounting for inference cost



Depending on how many tokens are extracted in inference, higher cost of overtrained 7B model may be worth it

**Beyond Chinchilla-Optimal:
Accounting for Inference in Language Model Scaling Laws**

Nikhil Sardana¹ Jacob Portes¹ Sasha Doubov¹ Jonathan Frankle¹

e.g., Llama1 7B trained on 1.4T tokens (Chinchilla recipe) in Feb'23,
but a year later Llama3 8B was trained on 5T tokens

Correcting Mistakes in Chinchilla Paper

[Epoch AI, 2024]

Motivation: Accurate prediction on models that are not compute-optimal

We reconstruct a subset of the data in Hoffmann et al.'s paper by extracting it from their plots and fit the same parametric model. Our analysis reveals several potential issues with Hoffmann et al.'s estimates of the parameters of their scaling law:

1. Hoffmann et al.'s estimated model fits the reconstructed data poorly, even when accounting for potential noise in the data reconstruction and excluding outlier models.
2. The confidence intervals reported by Hoffmann et al. are implausibly tight given the likely number of data points they had (~400). Obtaining such tight intervals would require hundreds of thousands of observations.
3. The scaling policy implied by Hoffmann et al.'s estimated model is inconsistent with their other approaches and the 20-tokens-per-parameter rule of thumb used to train their Chinchilla model.

$$L(N, D) = 1.8172 + \frac{482.01}{N^{0.3478}} + \frac{2085.43}{D^{0.3658}}$$

Another major Chinchilla Amendment (data-constrained training)

Scaling Data-Constrained Language Models

Niklas Muennighoff¹

Alexander M. Rush¹

Boaz Barak²

Teven Le Scao¹

Aleksandra Piktus¹

Nouamane Tazi¹

Sampo Pyysalo³

Thomas Wolf¹

Colin Raffel¹

Motivation : Not enough data

e.g., Chinchilla law suggests training 530B model on 11T tokens

Assembling a dataset of 11T tokens may involve too many compromises (ie low-quality)

Specialized corpora (law, medicine, wikipedia etc.) are small; essentially fixed size.

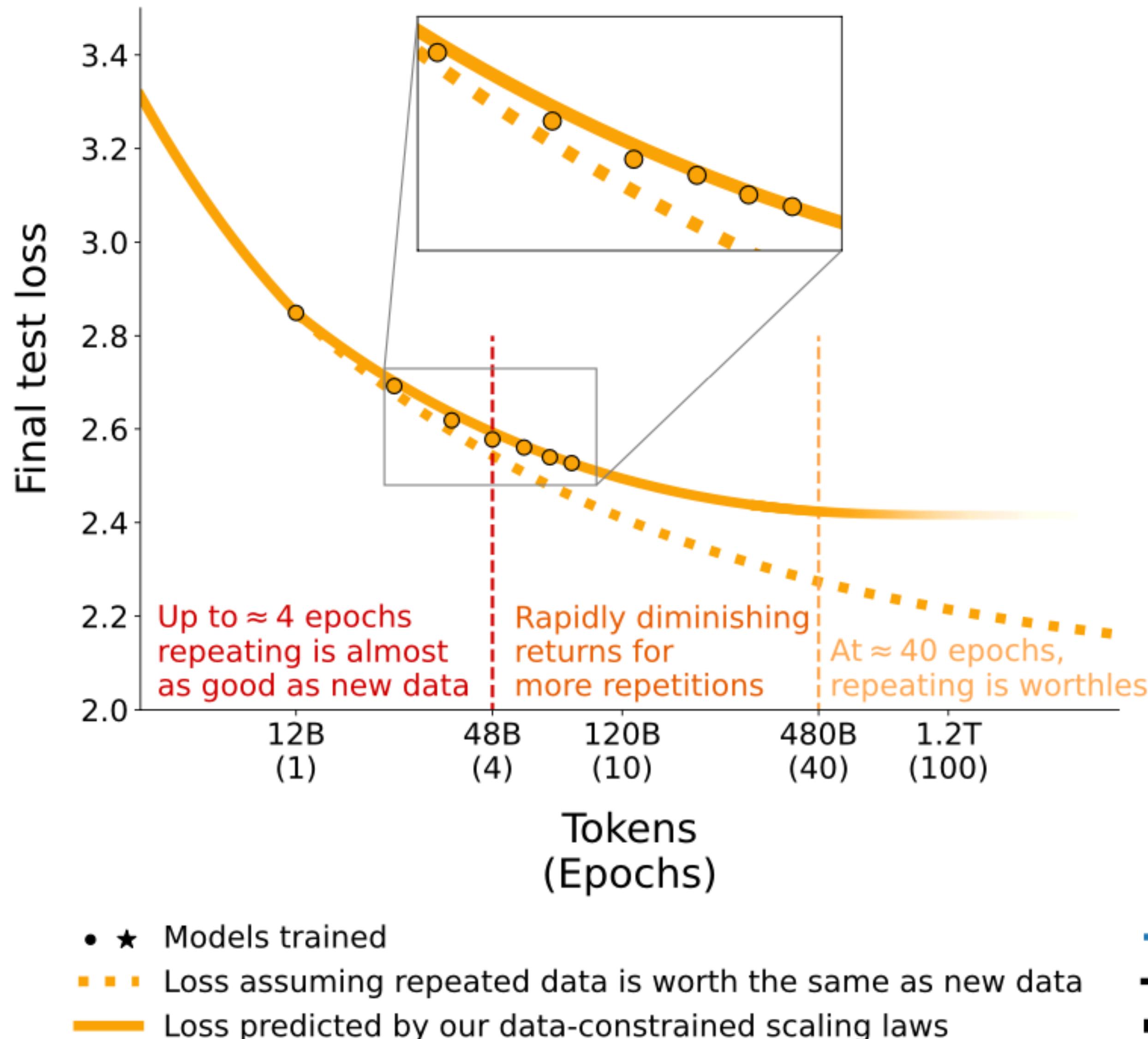
Scaling up with web data throws off the data-mix proportions

From paper abstract [Meunnighoff et al'23]

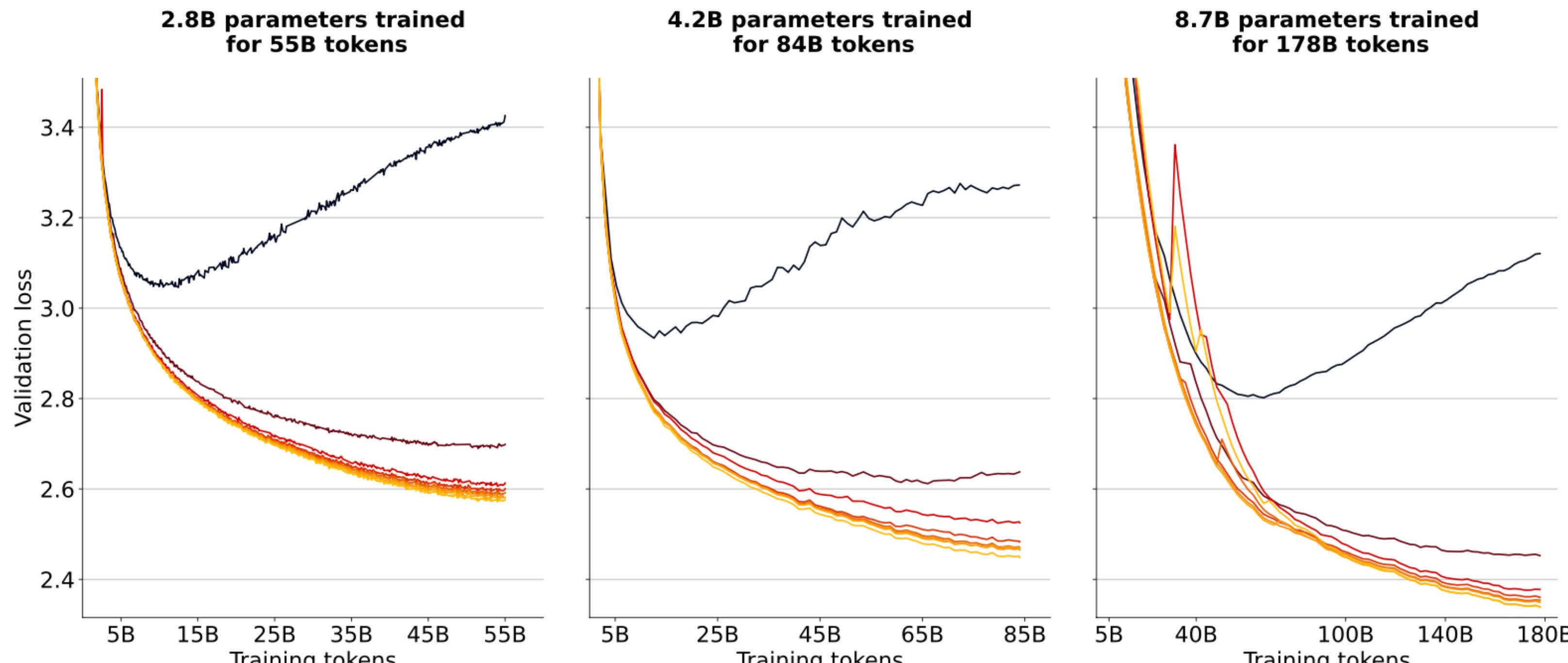
regimes. Specifically, we run a large set of experiments varying the extent of data repetition and compute budget, ranging up to 900 billion training tokens and 9 billion parameter models. We find that with constrained data for a fixed compute budget, training with up to 4 epochs of repeated data yields negligible changes to loss compared to having unique data. However, with more repetition, the value of adding compute eventually decays to zero. We propose and empirically validate a scaling law for compute optimality that accounts for the decreasing value of repeated tokens and excess parameters. Finally, we experiment with approaches mitigating data scarcity, including augmenting the training dataset with code data or removing commonly used filters. Models and datasets from our 400 training runs

Experiments with 4.2B model

Return on compute when repeating



What is your takeaway from this figure if you're training a model that will be widely used? (eg Llama3)



Epochs

— 1 — 2 — 3 — 4 — 5 — 7 — 14 — 44

FLOP budget (C)	Parameters (N)	Training tokens (D)	Data budget (D_C)
9.3×10^{20}	2.8B	55B	{ 55, 28, 18, 14, 11, 9, 4, 1.25 }B
2.1×10^{21}	4.2B	84B	{ 84, 42, 28, 21, 17, 12, 6, 1.9 }B
9.3×10^{21}	8.7B	178B	{ 178, 88, 58, 44, 35, 25, 13, 4 }B

Figure 4: Validation Loss for Different Data Constraints (IsoFLOP). Each curve represents the same number of FLOPs spent on an equal size model. Colors represent different numbers of epochs due to repeating because of data constraints. Parameters and training tokens are set to match the single-epoch compute-optimal configurations for the given FLOPs. Models trained on data that is repeated for multiple epochs have consistently worse loss and diverge if too many epochs are used.

Thought process in deriving Chinchilla-like law

$$L(N, D) = \frac{A}{N'^\alpha} + \frac{B}{D'^\beta} + E$$

D' = "effective # of tokens"

N' = "effective # of parameters"

Let D = total # of tokens with repetition. U_D = unique tokens

Let U_N = optimal # parameters for U_D tokens (as per Chinchilla)

Define $R_D = \frac{D}{U_D} - 1$ $R_N = \frac{N}{U_N} - 1$

Hypothesis: There exist learnable parameters R_D^*, R_N^* such that

$$D' = \text{Effective datasize} = U_D + U_D R_D^* \left(1 - e^{-\frac{R_D}{R_D^*}}\right)$$

$$N' = U_N + U_N R_N^* \left(1 - e^{\frac{-R_N}{R_N^*}}\right).$$

Motivation: Exponential drop-off in effectiveness

Fitting this model

$$L(U_N, U_D, R_N, R_D) = \frac{A}{(U_N + U_N R_N^* (1 - e^{\frac{-R_N}{R_N^*}}))^{\alpha}} + \frac{B}{(U_D + U_D R_D^* (1 - e^{\frac{-R_D}{R_D^*}}))^{\beta}} + E$$

Do best fit using Huber loss

$$L_{\delta}(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{if } |a| > \delta \end{cases}$$

(Modification of MSE, less sensitive to outliers)

$R_N^* = 5.31, R_D^* = 15.39$ fit the data quite well to give the expression

$$L(U_D, R_N, R_D) = \frac{521}{(U_N + 5.3 \cdot U_N (1 - e^{\frac{-R_N}{5.3}}))^{0.35}} + \frac{1488}{(U_D + 15.4 \cdot U_D (1 - e^{\frac{-R_D}{15.4}}))^{0.35}} + 1.87$$

where $U_N = U_D \cdot 0.051$

..

Two ways to overcome limited text data

[Meunnighof et al]

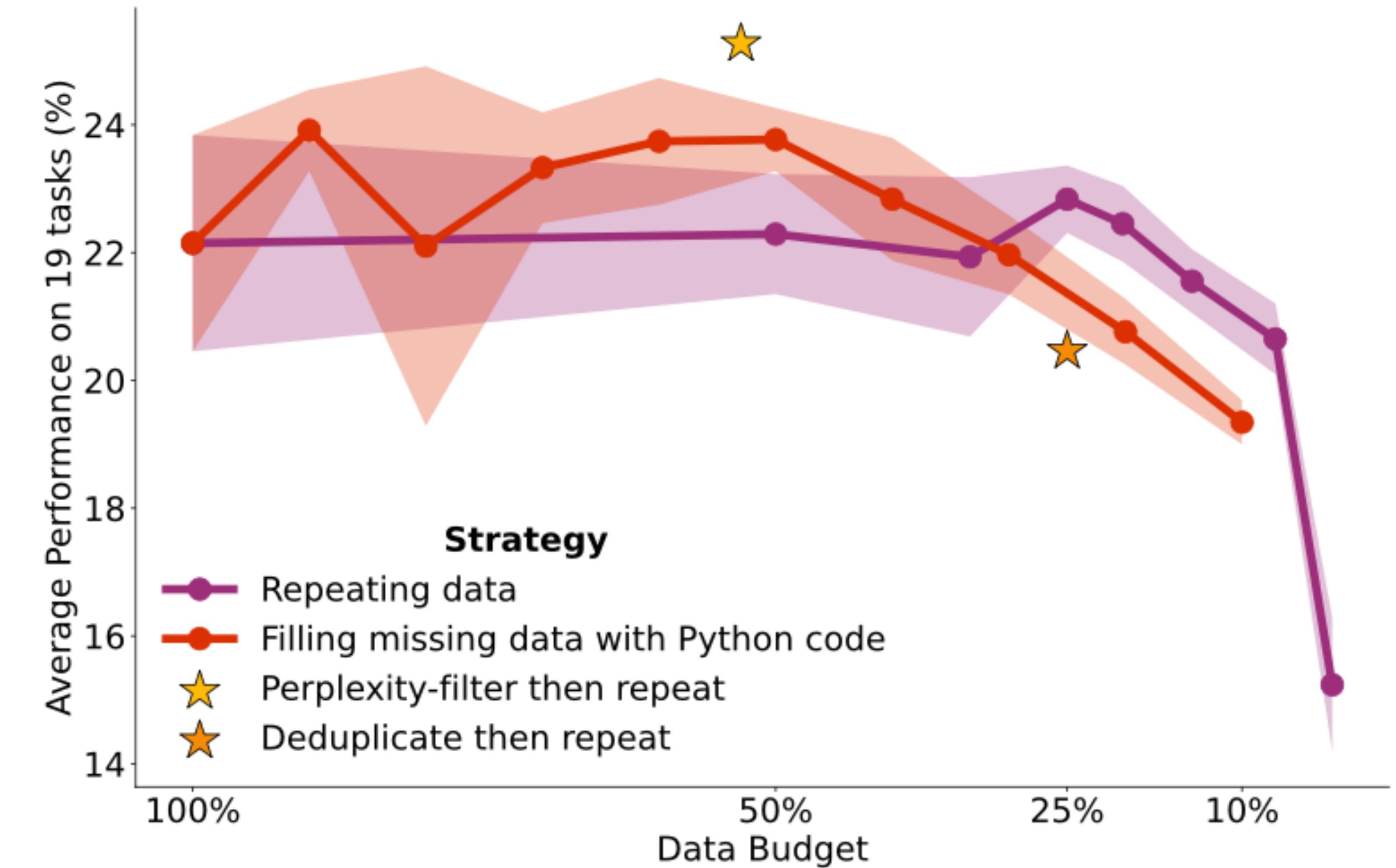
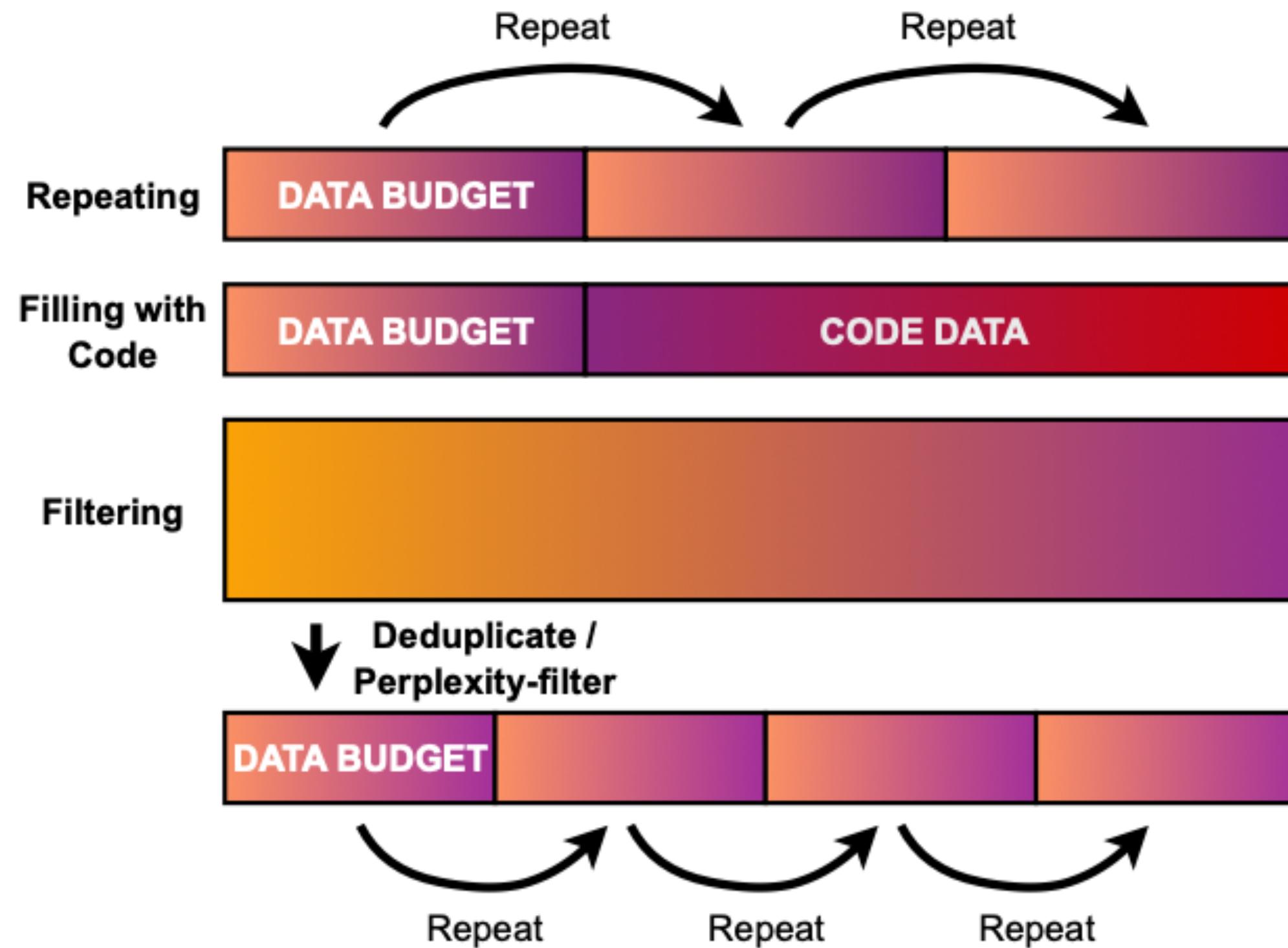
Note: once you play with data mix, held-out perplexity is no longer a good measure. (Why?) Customary to evaluate via performance on **downstream evals**

Setup

4.2B model, trained with 84B tokens. Tokens could be (i) unique (ii) repeated (iii) code tokens (iv) filtered using perplexity

For these experiments, we set a maximum data budget (D_C) of 84 billion tokens. For repetition and code filling, only a subset of D_C is available and the rest needs to be compensated for via repeating or adding code. For both filtering methods, we start out with approximately twice the budget (178 billion tokens), as it is easier to gather noisy data and filter it than it is to gather clean data for training. For perplexity filtering, we select the top 25% samples with the lowest perplexity according to a language model trained on Wikipedia. This results in 44 billion tokens that are repeated for close to two epochs to reach the full data budget. For deduplication filtering, all samples with a 100-char overlap are removed resulting in 21 billion tokens that are repeated for four epochs during training. See **Appendix N** for more details on the filtering procedures.

Note: “lowest perplexity” \implies highest probability (hopefully, “most like wikipedia”)



Interesting Settings: (i) Code + Data (up to 50-50 is good)
(ii) apply perplexity filter to get 42 B tokens, then 2 epochs

Caveat: Code is known to improve reasoning, and they didn't test for this

Next time

“Emergence” phenomenon for LLMs

Controversy whether it is real or illusory

Can we understand it at some level?