Foundations: Pretraining and scaling laws

Sean Welleck

Neural Code Generation
Carnegie Mellon University
January 18, 2024

Course Part I

Part I: Foundations

- **Learning**
- Evaluation
- Inference
- Data

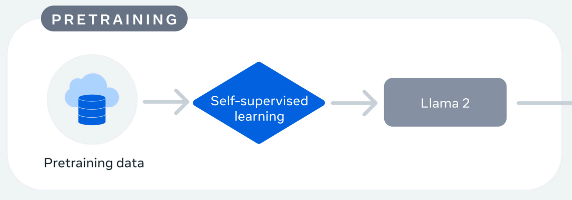
Language models

Language model learning pipeline

- Pretraining
  - Gives a “foundation model”
- Adaptation
  - Continued pretraining
  - Fine-tuning
  - Learning from feedback
  - In-context learning / prompting

Language models

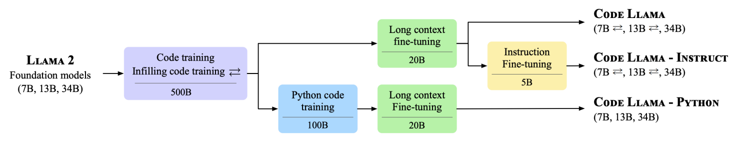
Example: CodeLlama [6]

- Pretraining
  - 2 trillion (T) tokens of mixed data (web, code, etc.)
  - 

Language models

Example: CodeLlama [6]

- **Pretraining**
  - 2 trillion (T) tokens of mixed data (web, code, etc.)
- Adaptation



  - Continued pretraining
    - 500 billion (B) tokens of mostly code data
  - Finetuning
    - Long sequences, Python code, and/or instructions

Outline

- Recap of language models and pretraining objective
- Scaling laws for understanding pretraining
- What do these scaling laws not capture?

Recap: Language models

A language model is a probability distribution over sequences:

$$p_\theta(\mathbf{y}) \tag{1}$$

- $\mathbf{y} = (y_1, \ldots, y_T)$
- θ: parameters

Recap: Autoregressive neural language models

Typical language models are autoregressive, and are parameterized by a transformer:

$$p_\theta(\mathbf{y}) = \prod_{t=1}^{T} p_\theta(y_t|y_{<t}) \tag{2}$$

- θ: transformer[1]

[1] For a review of transformers, see Chapter 12 of Bishop, *Deep Learning* *https://www.bishopbook.com/*.

Recap: Autoregressive neural language models

Autoregressive distributions allow for easy sampling:

- $\hat{y}_1 \sim p_\theta(\emptyset)$
- $\hat{y}_2 \sim p_\theta(\cdot|\hat{y}_1)$
- $\ldots$
- $\rightarrow \hat{\mathrm{y}} \sim p_\theta(\mathrm{y})$

Recap: Autoregressive neural language models

Autoregressive distributions allow for easy sampling:

- $\hat{y}_1 \sim p_\theta(\emptyset)$
- $\hat{y}_2 \sim p_\theta(\cdot|\hat{y}_1)$
- $\ldots$
- $\rightarrow \hat{\mathbf{y}} \sim p_\theta(\mathbf{y})$

Next: how do we learn the parameters θ?

Learning: maximum likelihood

Make observed data likely under the model; *maximum likelihood*:

$$\arg\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathsf{y} \in \mathcal{D}} \log p_{\theta}(\mathsf{y}) \tag{3}$$

Learning: maximum likelihood

Make observed data likely under the model; *maximum likelihood*:

$$\arg\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathsf{y} \in \mathcal{D}} \log p_{\theta}(\mathsf{y}) \tag{3}$$

- Example: $\mathcal{D}$ is 2 trillion tokens for Llama 2

Learning: next-token

Equivalently, learn to 'predict the next token':

$$\arg\max_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{y}\in\mathcal{D}} \log p_\theta(\mathbf{y}) \tag{4}$$

$$\equiv \arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{y}\in\mathcal{D}} \sum_{t=1}^{T} \underbrace{-\log p_\theta(y_t|y_{<t})}_{L_t} \tag{5}$$

Learning– Distribution matching

Equivalently, **match a target distribution**:

$$\arg\min_{\theta} \mathrm{KL}(q \| p_\theta), \tag{6}$$

where the dataset $\mathcal{D} \sim q$ is sampled from a *target distribution* q.[2]

[2] KL: Kullback-Leibler divergence

Learning– Distribution matching

Equivalently, **match a target distribution**:

$$
\begin{aligned}
\min_\theta \mathrm{KL}(q \| p_\theta) &= \min_\theta -\sum_{\mathsf{y} \in \mathcal{Y}} q(\mathsf{y}) \log \frac{p_\theta(\mathsf{y})}{q(\mathsf{y})} \\
&\equiv \min_\theta -\sum_{\mathsf{y} \in \mathcal{Y}} q(\mathsf{y}) \log p_\theta(\mathsf{y}) + \text{constant} \\
&\equiv \min_\theta -\mathbb{E}_{\mathsf{y} \sim q} \log p_\theta(\mathsf{y}) \\
&\approx \min_\theta -\frac{1}{|\mathcal{D}|} \sum_{\mathsf{y} \in \mathcal{D}} \log p_\theta(\mathsf{y}) \\
&\equiv \underbrace{\max_\theta \sum_{\mathsf{y} \in \mathcal{D}} \log p_\theta(\mathsf{y})}_{\text{Maximum likelihood!}}
\end{aligned}
$$

Recap

Next-token prediction has a nice interpretation: it fits the language model p_θ to a target distribution q represented by the dataset $\mathcal{D}$.

The Bitter Lesson

We want to fit the distribution better by “adding more compute”:

- *“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”*[3]

[3] *The Bitter Lesson*, Richard Sutton 2019

We want to fit the distribution better by "adding more compute":

- *"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"*[3]

What is "compute"?

[3] *The Bitter Lesson*, Richard Sutton 2019

Compute

We spend **compute** by performing forward and backward passes using our model on token sequences.

Compute

We spend **compute** by performing forward and backward passes using our model on token sequences.

A rough approximation for transformer language models is [4]:

$$C \approx 6ND \tag{7}$$

- N: number of model parameters
- D: number of tokens
- C: compute; floating point operations (FLOPs)

Compute

We spend **compute** by performing forward and backward passes using our model on token sequences.

For example, LLama 2:

$$
\begin{aligned}
C &\approx 6 * 7 \text{ billion } * 2 \text{ trillion} && (8)\\
&= 8.4 \times 10^{22} FLOPs && (9)
\end{aligned}
$$

Compute

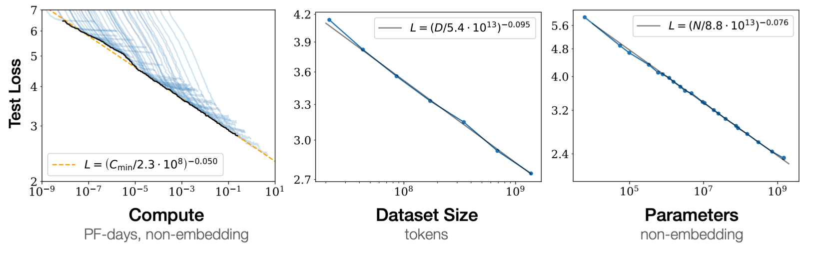
We spend **compute** by performing forward and backward passes using our model on token sequences.

For example, LLama 2:

$$C \approx 6 * 7 \text{ billion } * 2 \text{ trillion} \tag{8}$$
$$= 8.4 \times 10^{22} FLOPs \tag{9}$$

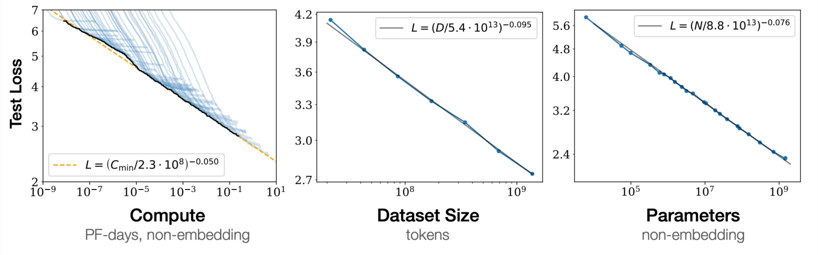
We can **increase compute** by increasing the **number of parameters** ($\uparrow N$), training on **more tokens** ($\uparrow D$), or a **combination** thereof.

Good news: cross entropy loss gets better with more compute



Test loss predictably improves with more compute [Kaplan et al 2020 [4]].

Good news: cross entropy loss gets better with more compute

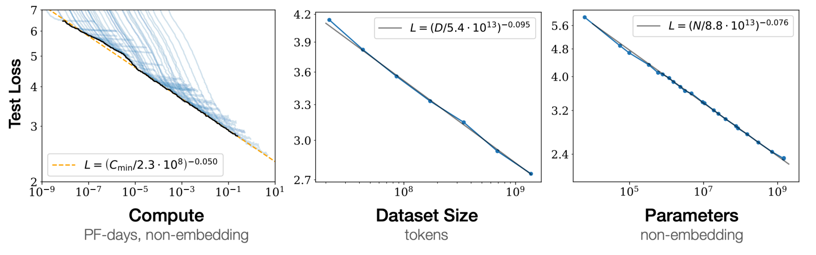


Specifically, loss scales as a power-law with the amount of compute:

$$\underbrace{L(X) \propto 1/X^{\alpha_X}}_{\text{scaling law}}, \tag{10}$$

where X is compute C, dataset size D, or parameters N.

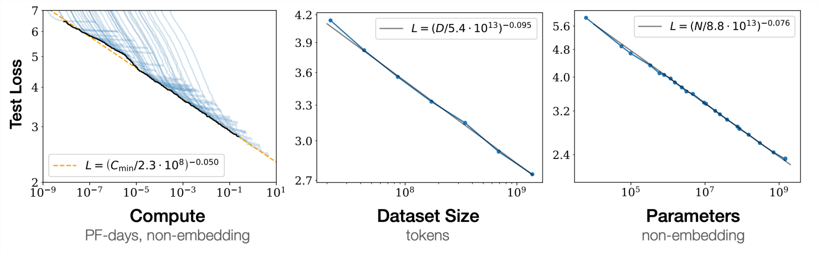
Good news: cross entropy loss gets better with more compute



Example:

$$L(C) \propto 1/C^{0.05} \tag{11}$$

Good news: cross entropy loss gets better with more compute



Basic idea:

- Train models of size $N_1, \ldots, N_n$ for $D_1, \ldots, D_d$ tokens.
- Plot loss at each step (light blue lines)
- Pick the minimum loss at each amount of compute (black line)
- Run linear regression on the resulting $(\log L, \log C)$ pairs

Typically translates to better task performance

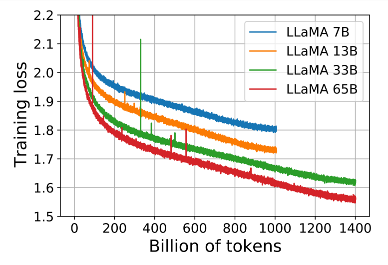


Figure 1: Llama training loss

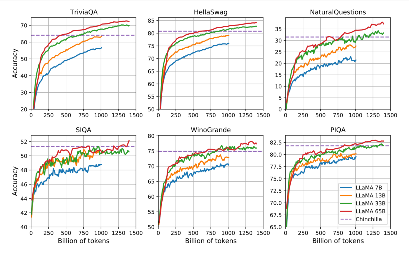


Figure 2: Llama task performance

Good news: it appears to hold for code

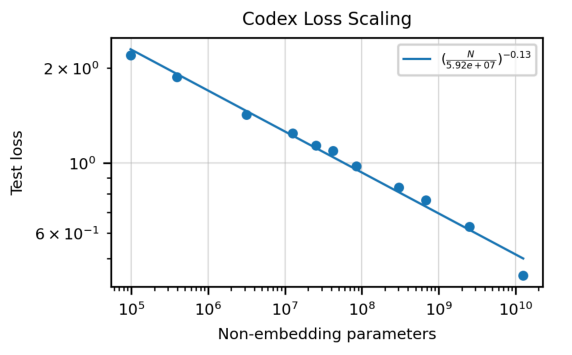


Figure 3: Codex test loss scaling in number of parameters N

Good news: it appears to hold for code

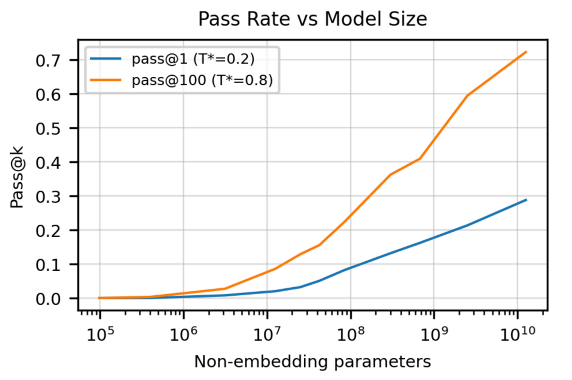


Figure 4: Codex pass rate on HumanEval as a function of parameters N

Recap

- Pretraining is equivalent to fitting a target distribution
- The fit predictably gets better as we increase compute, as described by a scaling law

Recap

- Pretraining is equivalent to fitting a target distribution
- The fit predictably gets better as we increase compute, as described by a scaling law

Should I spend my compute on a larger model, or on more data?

Scaling laws: allocation

Allocation:

For compute budget C, choose number of parameters N and tokens D that minimizes loss.

Scaling laws: allocation

Allocation:

For compute budget C, choose number of parameters N and tokens D that minimizes loss.

$$\arg\min_{N,D} L(N, D)$$
$$\text{subject to } 6ND \leq C$$

Investigated in “the Chinchilla paper” [Hoffmann et al 2022 [3]]

Allocation: Chinchilla

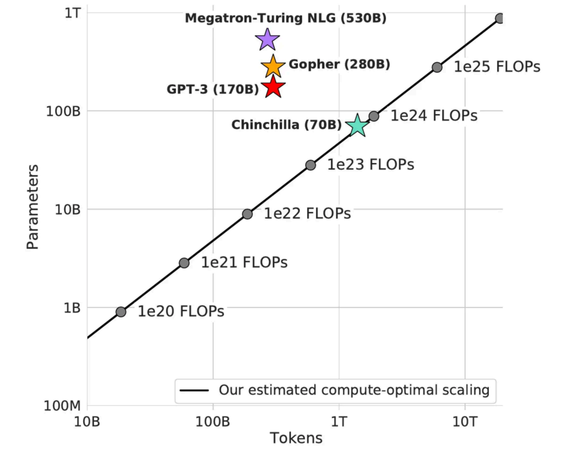


Figure 5: Previous models (e.g. Gopher) allocate a large portion of compute to model size. Chinchilla is a smaller model trained on more tokens that outperforms Gopher.

Allocation: Chinchilla

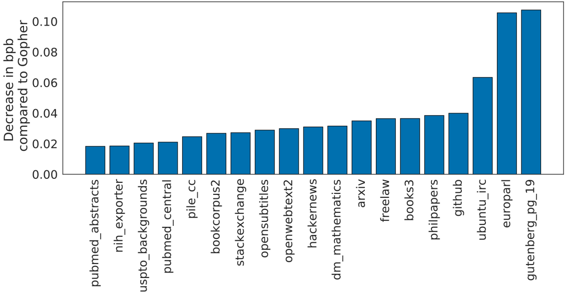


Figure 5 | **Pile Evaluation.** For the different evaluation sets in The Pile (Gao et al., 2020), we show the bits-per-byte (bpb) improvement (decrease) of *Chinchilla* compared to *Gopher*. On all subsets, *Chinchilla* outperforms *Gopher*.

Allocation: Chinchilla

To choose Chinchilla's allocation, the authors fit scaling laws on runs with smaller amounts of compute. They used three 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) [23]	0.73	0.27

$a \approx b$: parameters and tokens should be scaled at the same rate.

Allocation: Chinchilla

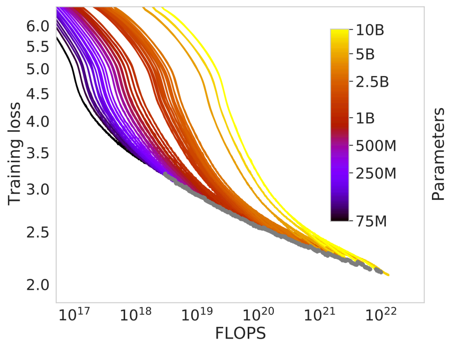
To choose Chinchilla's allocation, the authors fit scaling laws on runs with smaller amounts of compute. They used three approaches.

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) [23]	0.73	0.27

$a \approx b$: parameters and tokens should be scaled at the same rate.

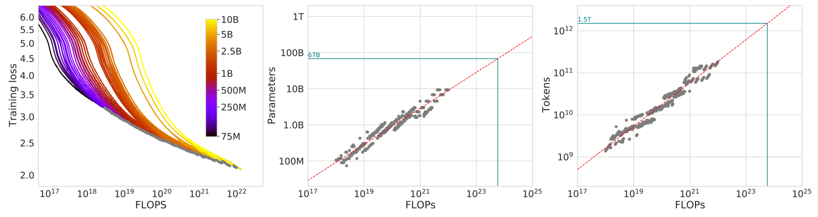
To understand this kind of analysis, we will look at Approach 1

Approach 1: fix N and vary D



- For each size N, train 4 models with different number of tokens D
- For each compute C, pick the model with the lowest loss L
- We now have (C, N, D, L) examples (grey points)

Approach 1: fix N and vary D



- Fit power laws using the (C, N, D, L) examples.
  - Middle: $N_{\text{opt}} \propto C^a$ (optimal model size)
  - Right $D_{\text{opt}} \propto C^b$ (optimal number of tokens)

Allocation: scale parameters and data equally

As a recap, the slope of the lines appears in the table: scale parameters and tokens at similar rates.

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) [23]	0.73	0.27

Post-Chinchilla

- The Chinchilla scaling law arguably led to a focus on scaling data
- Trend: train on ***even more tokens*** than suggested by the compute-optimal scaling law.[4]

[4] Training a smaller model on more tokens may be compute optimal when *inference-time compute* is factored in; smaller models require less inference compute.

Post-Chinchilla

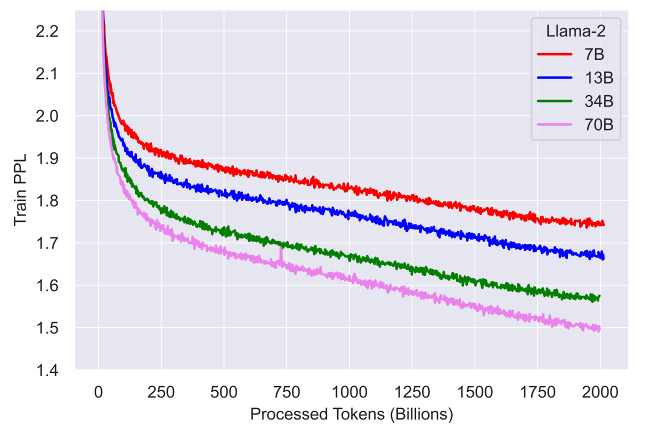


Figure 6: Example: Llama 2 – more tokens than Chinchilla, equal size (70B)

Scaling laws as a tool in the toolbox

deepseek

DeepSeek LLM
Scaling Open-Source Language Models with Longtermism

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y.K. Li, Wenfeng Liang, Fangyun Lin, A.X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R.X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, Yuheng Zou [*]

[*]**DeepSeek-AI**

.CL] 5 Jan 2024

Scaling laws as a tool in the toolbox

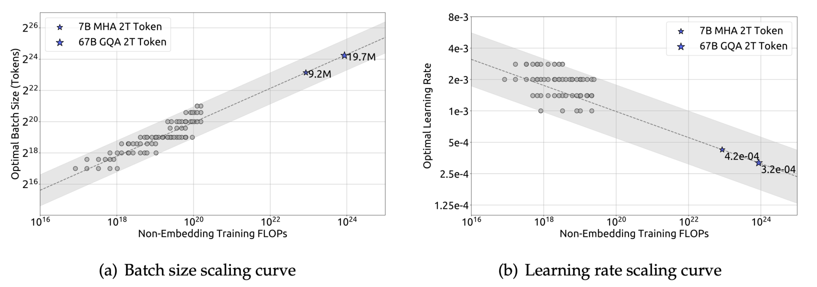


Figure 7: Scaling laws for batch size and learning rate

Scaling laws as a tool in the toolbox

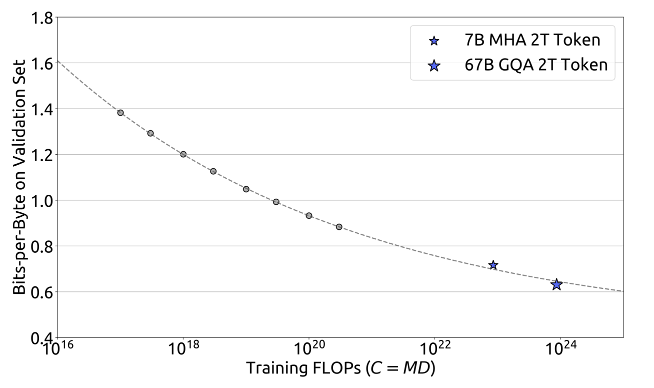


Figure 8: Predicting performance of larger models

Recap

- Scaling laws can determine "compute-optimal training"
    - I.e., the choice of N and D that minimizes loss at compute budget C.
- Scaling the amount of data is important!!

Data constraints

What if we run out of data?

Data-constrained scaling

Data-constrained setting

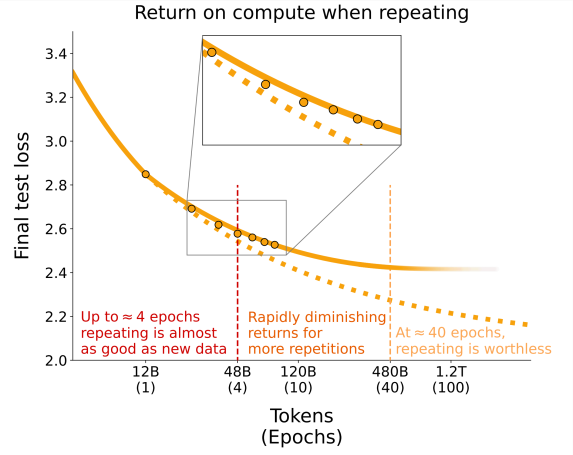
- We might want to train on much more than 2 trillion tokens
- Some programming languages have less tokens
  - E.g. Starcoder pretraining data: $\approx$ 300 billion code tokens
  - E.g. Lean has $\approx$ 300 million tokens [1]

Data-constrained scaling

Option 1: repeat the data

- Studied in *Scaling Data-Constrained Language Models* [5]

Data-constrained scaling

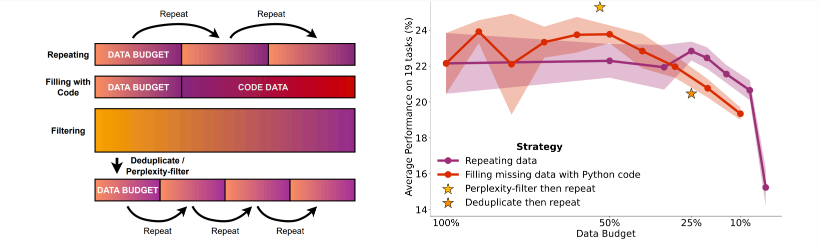


Finding: repeating can be good

- 4 epochs is nearly as good as 1 epoch with 4x the data

Data-constrained scaling

Option 2: mix in other data



- N_1 web tokens + N_2 code tokens $\approx$ repeating N_1 web tokens

Data-constrained scaling

Option 3: transfer

- Pretrain on $\mathcal{D} \sim q$ (e.g. web)
- Continue training on $\mathcal{D}' \sim q'$ (e.g. code)

Scaling laws of transfer

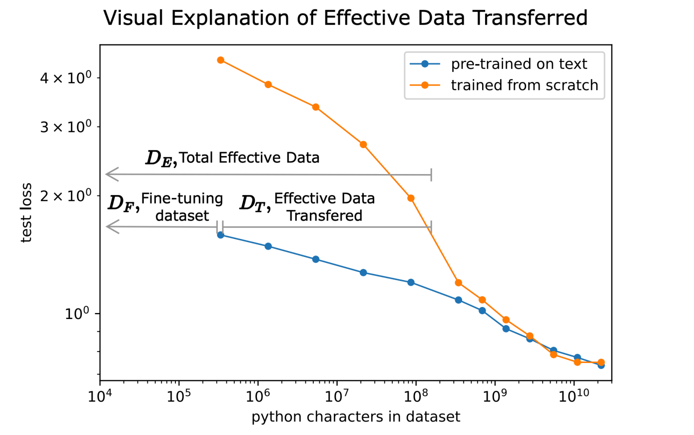


Figure 9: *Scaling Laws for Transfer* [2]

Effective data transfer: code tokens saved by pretraining on text

Scaling laws of transfer

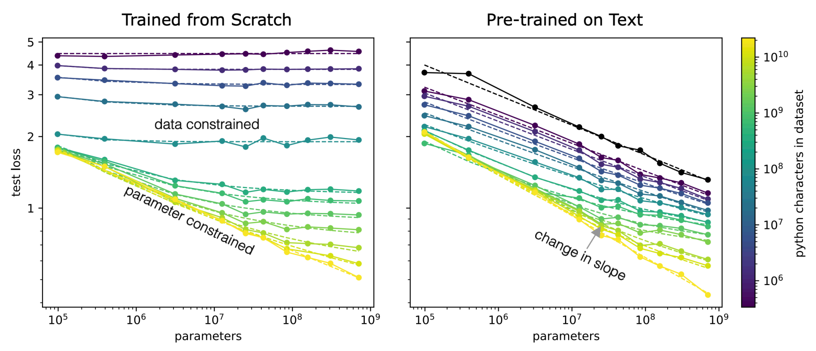


Figure 10: *Scaling Laws for Transfer* [2]

Low-data setting: without pretraining on text, we get no benefit from increasing parameters.

Data-constrained scaling: Llemma

Llemma [1]:

- Pretrain on web and code
  - Initialize with $\theta_{\text{codellama}}$
- Transfer to specialized programming languages and math
  - Continue training on $\mathcal{D}'$: 55 billion token Proofpile II

Data-constrained scaling: Llemma

LLEMMA [1]:

- Pretrain on web and code
  - Initialize with $\theta_{\text{codellama}}$
- Transfer to specialized programming languages and math
  - Continue training on $\mathcal{D}'$: 55 billion token PROOFPILE II
    - Mathematical code (e.g., Lean)
    - Mathematical web data
    - Scientific papers

Data-constrained scaling: Llemma

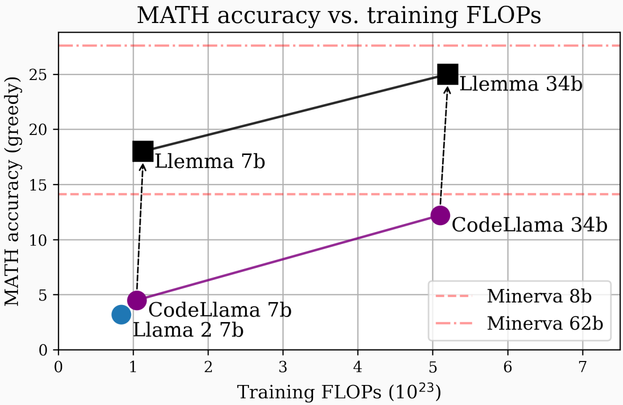


Figure 11: Llemma improves with a modest amount of math-specific compute

Recap

To keep reducing loss, we need many tokens. What if we run out?

- Repeating tokens can be a useful allocation of compute
- Leverage tokens from a data-rich distribution (e.g. web text)

Summary

- Pretraining fits the distribution of pretraining data
- Scaling laws let us forecast performance, allocate compute, and choose hyperparameters
- In low-data settings: repeat data, mix in other data, transfer

Looking ahead

What do these scaling laws **not** cover?

Looking ahead

What do these scaling laws **not** cover?

- **Data quality**: ‘better’ data may be more compute efficient

We will discuss all of these during the semester!

Looking ahead

What do these scaling laws **not** cover?

- **Data quality**: ‘better’ data may be more compute efficient
- **Training objective**: next-token may not be optimally efficient

We will discuss all of these during the semester!

Looking ahead

What do these scaling laws **not** cover?

- **Data quality**: 'better' data may be more compute efficient
- **Training objective**: next-token may not be optimally efficient
- **Distribution mismatch**: what if we perfectly fit q, but want q'
  - q: code on the internet
  - q': code that satisfies a user's intent

We will discuss all of these during the semester!

Looking ahead

What do these scaling laws **not** cover?

- **Data quality**: 'better' data may be more compute efficient
- **Training objective**: next-token may not be optimally efficient
- **Distribution mismatch**: what if we perfectly fit q, but want q'
  - q: code on the internet
  - q': code that satisfies a user's intent
- Many others: architecture, inference cost, performance metric,...

We will discuss all of these during the semester!

Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. McAleer, A. Q. Jiang, J. Deng, S. R. Biderman, and S. Welleck.
Llemma: An open language model for mathematics.
ArXiv, abs/2310.10631, 2023.

D. Hernandez, J. Kaplan, T. Henighan, and S. McCandlish.
Scaling laws for transfer, 2021.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. W. Rae, and L. Sifre.
Training Compute-Optimal Large Language Models.
In *Advances in Neural Information Processing Systems*, 2022.

J. Kaplan, S. McCandlish, T. J. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei.
Scaling laws for neural language models.
ArXiv, abs/2001.08361, 2020.

N. Muennighoff, A. M. Rush, B. Barak, T. L. Scao, A. Piktus, N. Tazi, S. Pyysalo, T. Wolf, and C. Raffel.
Scaling data-constrained language models.
arXiv preprint arXiv:2305.16264, 2023.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. P. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. D'efossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve.
Code llama: Open foundation models for code.

ArXiv, abs/2308.12950, 2023.

Appendix

Approach 3: parametric fit

Step 1: hypothesize a scaling law

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta} \tag{12}$$

Allocation

Step 1: hypothesize a scaling law

$$L(N, D) = \underbrace{E} + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} \tag{13}$$

"Entropy term": with infinite parameters and infinite data ($N, D \to \infty$), we should approach the minimum achievable loss (entropy).

Allocation

Step 1: hypothesize a scaling law

$$L(N, D) = E + \underbrace{\frac{A}{N^{\alpha}}} + \frac{B}{D^{\beta}} \tag{14}$$

"Modeling cost": with infinite data ($D \to \infty$), we should incur a cost from using a transformer with N parameters.

Allocation

Step 1: hypothesize a scaling law

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \underbrace{\frac{B}{D^{\beta}}} \qquad (15)$$

"Optimization cost": with infinite parameters ($N \to \infty$), we should incur a cost from using only D tokens with gradient descent.

Allocation: scale parameters and data equally

Step 2: fit constants E, A, α, B, β using losses from training runs

$$L(N, D) = E + \underbrace{\frac{A}{N^{0.34}}} + \underbrace{\frac{B}{D^{0.28}}} \tag{16}$$

Allocation: scale parameters and data equally

Step 3: derive the optimal parameters and tokens from L, plug in α, β:

$$N_{opt}(C) = G\left(\frac{C}{6}\right)^{a}, \quad D_{opt}(C) = G^{-1}\left(\frac{C}{6}\right)^{b}, \quad \text{where} \quad G = \left(\frac{\alpha A}{\beta B}\right)^{\frac{1}{\alpha+\beta}}, \quad a = \frac{\beta}{\alpha+\beta}, \text{ and } b = \frac{\alpha}{\alpha+\beta}$$

Result: $a = 0.46$, $b = 0.54$