Bigger = Better ? Scaling laws

There are many LLM's, with varying choices of

- number of parameters N
- size of training data D
    - number of tokens trained on
    - not *distinct* tokens in dataset
        - same token encountered in each epoch is counted once per epoch
- amount of compute for training C

Here is a table from the GPT-3 paper (https://arxiv.org/pdf/2005.14165.pdf#page=46)

D Total Compute Used to Train Language Models

This appendix contains the calculations that were used to derive the approximate compute used to train the language models in Figure 2.2. As a simplifying assumption, we ignore the attention operation, as it typically uses less than 10% of the total compute for the models we are analyzing.

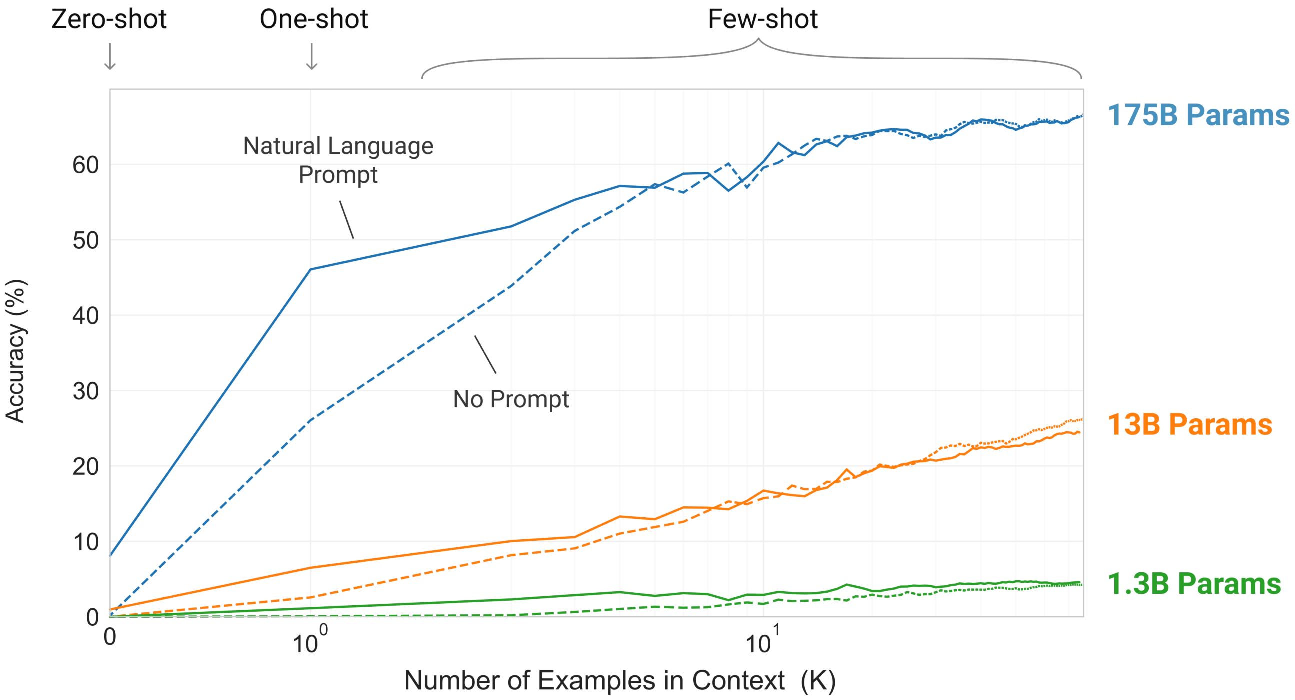
Calculations can be seen in Table D.1 and are explained within the table caption.

Model	Total train compute (PF-days)	Total train compute (flops)	Params (M)	Training tokens (billions)	Flops per param per token	Mult for bwd pass	Fwd-pass flops per active param per token	Frac of params active for each token
T5-Small	2.08E+00	1.80E+20	60	1,000	3	3	1	0.5
T5-Base	7.64E+00	6.60E+20	220	1,000	3	3	1	0.5
T5-Large	2.67E+01	2.31E+21	770	1,000	3	3	1	0.5
T5-3B	1.04E+02	9.00E+21	3,000	1,000	3	3	1	0.5
T5-11B	3.82E+02	3.30E+22	11,000	1,000	3	3	1	0.5
BERT-Base	1.89E+00	1.64E+20	109	250	6	3	2	1.0
BERT-Large	6.16E+00	5.33E+20	355	250	6	3	2	1.0
RoBERTa-Base	1.74E+01	1.50E+21	125	2,000	6	3	2	1.0
RoBERTa-Large	4.93E+01	4.26E+21	355	2,000	6	3	2	1.0
GPT-3 Small	2.60E+00	2.25E+20	125	300	6	3	2	1.0
GPT-3 Medium	7.42E+00	6.41E+20	356	300	6	3	2	1.0
GPT-3 Large	1.58E+01	1.37E+21	760	300	6	3	2	1.0
GPT-3 XL	2.75E+01	2.38E+21	1,320	300	6	3	2	1.0
GPT-3 2.7B	5.52E+01	4.77E+21	2,650	300	6	3	2	1.0
GPT-3 6.7B	1.39E+02	1.20E+22	6,660	300	6	3	2	1.0
GPT-3 13B	2.68E+02	2.31E+22	12,850	300	6	3	2	1.0
GPT-3 175B	3.64E+03	3.14E+23	174,600	300	6	3	2	1.0

We have already seen that some LLM properties

- like in-context learning (zero or few shot)
- "emerge" only when model size passes a threshold

This argues for bigger models.



There is also evidence that the emergence of ability to perform some in-context tasks

- is sudden
- rather than gradual as the number of parameters increase.

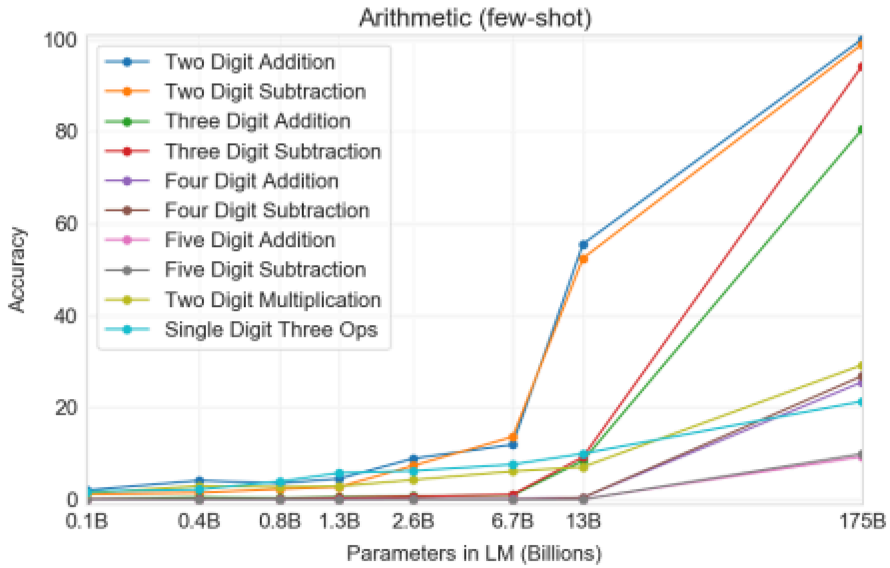


Figure 3.10: Results on all 10 arithmetic tasks in the few-shot settings for models of different sizes. There is a significant jump from the second largest model (GPT-3 13B) to the largest model (GPT-3 175), with the latter being able to reliably accurate 2 digit arithmetic, usually accurate 3 digit arithmetic, and correct answers a significant fraction of the time on 4-5 digit arithmetic, 2 digit multiplication, and compound operations. Results for one-shot and zero-shot are shown in the appendix.

Attribution: [GPT-3 paper (https://arxiv.org/pdf/2005.14165.pdf#page=46)](https://arxiv.org/pdf/2005.14165.pdf#page=46)

Is bigger N always better ?

Consider the costs. Larger N

- entails more computation: larger C
- probably requires more training data: larger D

If we fix a "budget" for one choice (e.g., C) we can explore choices for N, D that meet this budget.

Here are two models with the same C budget

- but vastly different N and D

model	Compute (PF-days)	params (M)	training tokens (B)
RoBERTa-Large	49.3	355	2000
GPT-3 2.7B	55.2	2650	300

Attribution: [GPT-3 paper (https://arxiv.org/pdf/2005.14165.pdf#page=46)](https://arxiv.org/pdf/2005.14165.pdf#page=46)

Given these choices: how do we choose ?

One way to quantify the decision is by setting a goal

- to maximize "performance"
- where this is usually proxied by "minimizing test loss" L
    - Cross Entropy for the "predict the next" token task of the LLM

We can state some basic theories

- Increasing N creates the *potential* for better performance L
- To *actualize* the potential
    - we need increased C
        - more parameters via increasing the number of stacked Transformer Blocks
    - we need increased D

But this still leaves many unanswered questions

- Can L always be reduced ?
    - Does performance hit a "ceiling"
    - For a fixed N: perhaps increasing D or C won't help
- What is the relationship between N and D ?
    - how much must D by increased when N increases
- For a fixed D: what is the best choice for N ?
    - holding performance constant

Scaling Laws: early research

Fortunately, this paper (https://arxiv.org/pdf/2001.08361.pdf) has

- conducted an empirical study of models with varying N, D, C and resulting L
- fit an empirical function (*Scaling Laws*) describing the dependency of L on N, D, C.

We briefly summarize the results.

"Performance" (test loss L) depends on scale.

Scale consists of 3 components

- Compute C
- Dataset size (really: number of training tokens) D
- Parameters N

We can set a "budget" for any of variables L, N, D, C

- and examine trade-offs for the non-fixed variable

The paper shows that

- Increasing your budget for one of the scale factors
- increases performance (decrease loss)
- **provided** the other two factors don't become bottlenecks

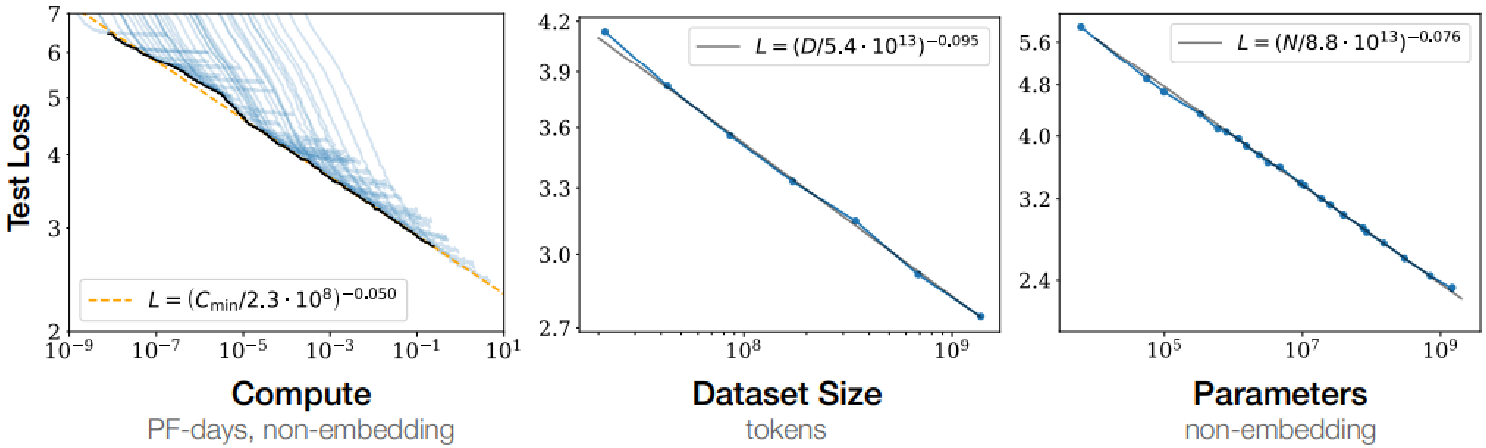
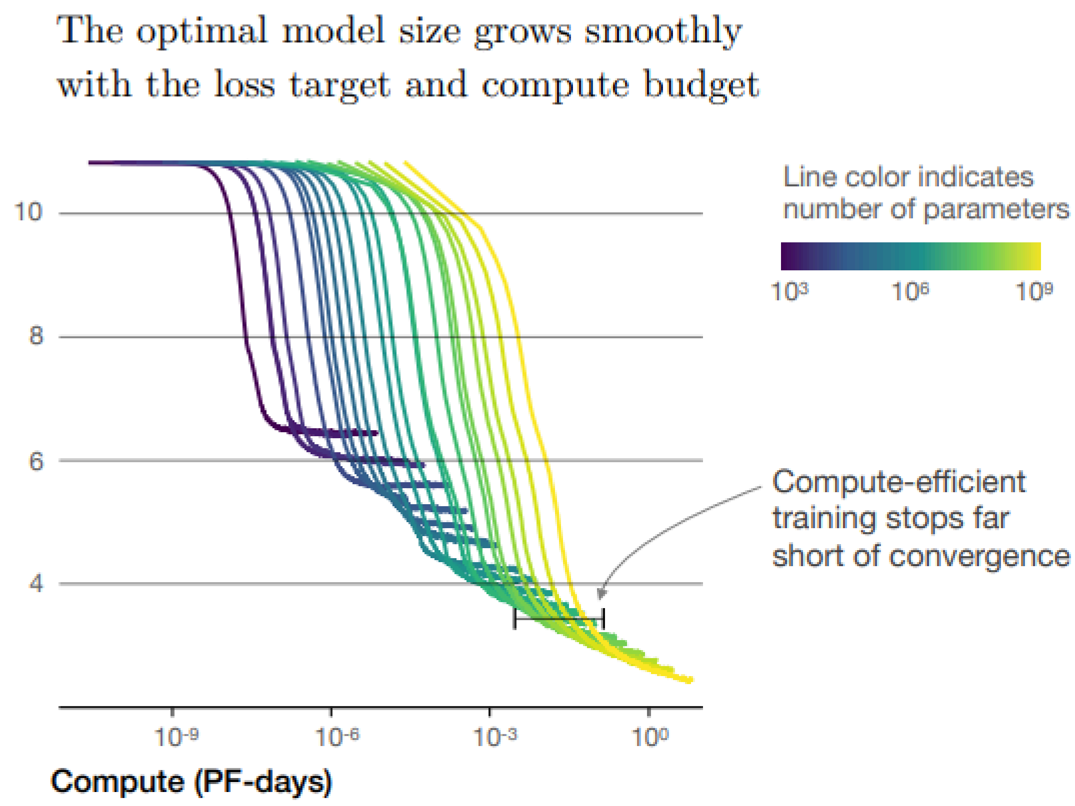


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when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

But bottlenecks are a worry:

- The potential performance of a model of fixed size N hits a "ceiling"
- That can't be overcome by increasing compute C



Observation

Consider fixing the loss at L = 8.

- models of all sizes can reach this performance
- but the *smaller* model reaches it with less compute
- if we are *compute constrained*
    - favor a smaller model

Observation

For a fixed Compute C

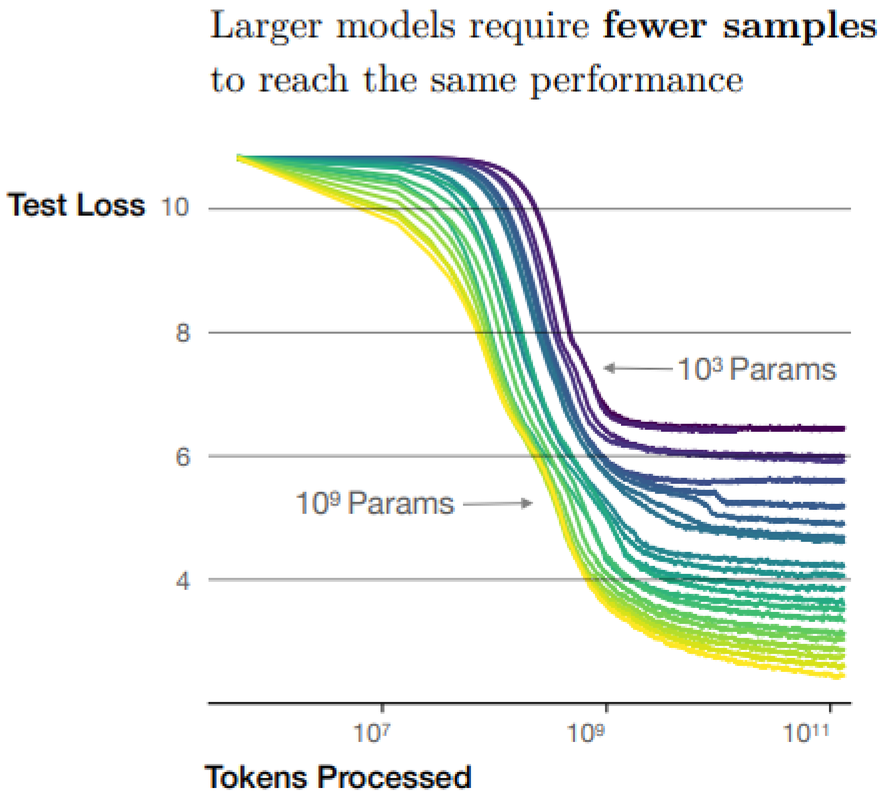
- a smaller model (that has reached its asymptotic minimum) has lower loss
- provided that there is enough training data

For a fixed L

- a smaller model reaches the loss with less compute

We can also set a performance budget L

- and examine the amount of training data D to reach this budget
- as N varies



Observation

For a fixed D

- bigger models are more data efficient
    - for a given level of loss L, a larger model achieves L with fewer tokens
- but at a higher C

But we can also see that more data D may compensate for fewer parameters

For some levels of loss (e.g., $L = 8$)

- the smallest (purple line) and largest (yellow line) models both achieve the Loss
- but the smallest model needs a much larger D

Key observation

- smaller models
- can achieve the same performance (e.g., Loss) as larger models
- **if** they are trained on more data

Given the *same performance*, a smaller models may be preferred

- because their training compute *may* still be smaller than the larger model
- their *inference* compute will be much smaller

Here is one graph that combines N and D

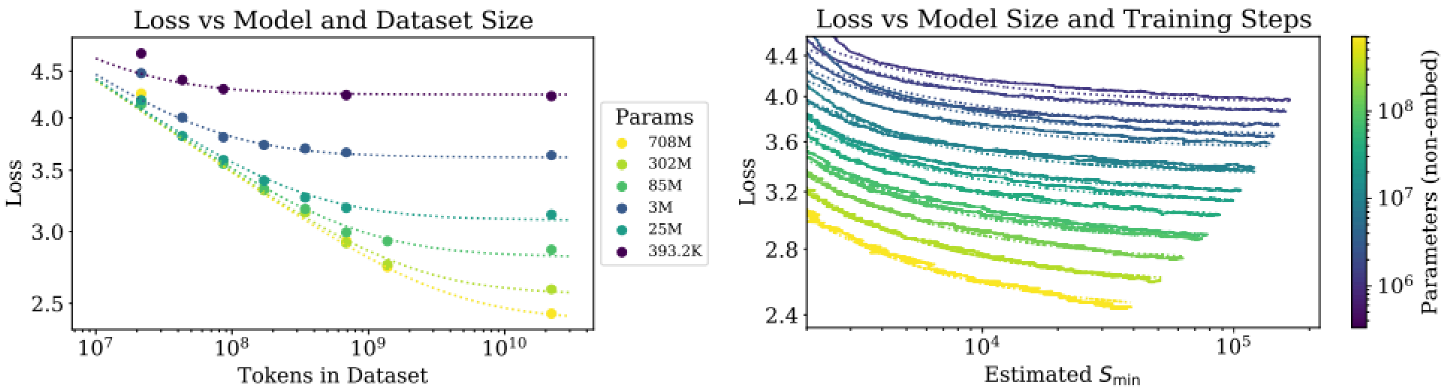


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of $S_{\min}$, the number of steps when training at large batch size (details in Section 5.1).

Key result: how must D scale with N ?

The major contribution of the research on Scaling

- fits a model of
  - Loss L as a function of number of parameters N and number of training tokens D
  - given a compute budget C

Given this function, one can find optimal N and D for a fixed compute budget C

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

This is a very interesting result.

- For someone on a fixed compute budget C
- One can find optimal values for model and data size

Note that

- because of fixed compute budget C
- L is an *early-stopped* loss, not necessarily the minimal L

From this result, we can vary compute resource C

- and solve for the optimal N and D
- and derive the relationship between N and D
- to stay on the compute-optimal frontier

$$\frac{N^{0.74}}{D} = \text{constant}$$

For example

- increasing N by a factor of 8
- requires D to increase by a factor of $8^{0.74} \approx 5$

Failure to scale N and D together causes performance to flatten.

The Scaling Laws (https://arxiv.org/pdf/2001.08361.pdf#page=4) show that Loss follows a Power Law as a function of N, C, D.

Here (https://arxiv.org/pdf/2001.08361.pdf#page=20) is a summary of the Scaling Laws.

Appendices

A Summary of Power Laws

For easier reference, we provide a summary below of the key trends described throughout the paper.

Parameters	Data	Compute	Batch Size	Equation
N	∞	∞	Fixed	$L(N) = (N_c/N)^{\alpha_N}$
∞	D	Early Stop	Fixed	$L(D) = (D_c/D)^{\alpha_D}$
Optimal	∞	C	Fixed	$L(C) = (C_c/C)^{\alpha_C}$ (naive)
N_{opt}	D_{opt}	C_{min}	$B \ll B_{\text{crit}}$	$L(C_{\text{min}}) = \left(C_c^{\text{min}}/C_{\text{min}}\right)^{\alpha_C^{\text{min}}}$
N	D	Early Stop	Fixed	$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$
N	∞	S steps	B	$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\text{min}}(S,B)}\right)^{\alpha_S}$

Table 4

The empirical fitted values for these trends are:

Power Law	Scale (tokenization-dependent)
$\alpha_N = 0.076$	$N_c = 8.8 \times 10^{13}$ params (non-embed)
$\alpha_D = 0.095$	$D_c = 5.4 \times 10^{13}$ tokens
$\alpha_C = 0.057$	$C_c = 1.6 \times 10^7$ PF-days
$\alpha_C^{\text{min}} = 0.050$	$C_c^{\text{min}} = 3.1 \times 10^8$ PF-days
$\alpha_B = 0.21$	$B_* = 2.1 \times 10^8$ tokens
$\alpha_S = 0.76$	$S_c = 2.1 \times 10^3$ steps

Table 5

Scaling laws: newer research

[Continuing research (https://arxiv.org/pdf/2203.15556.pdf)](https://arxiv.org/pdf/2203.15556.pdf) in the area of scaling

- confirms the need to scale N and D together
- but with a different scaling relationship

Key result: how must D scale with N ?

$$\frac{N}{D} = \text{constant}$$

Contrast this result with the original paper's relationship of the constant ratio as

$$\frac{N^{0.74}}{D} = \text{constant}$$

A key difference between the two papers is the *learning rate schedule*.

Recall: a learning rate moderates the rate a which gradient updates affect the model's weights during Gradient Descent

$$\mathbf{W}_{(t+1)} = \mathbf{W}_{(t)} - \alpha_t * \frac{\partial \mathcal{L}_{(t)}}{\partial \mathbf{W}}$$

where $\alpha_{(t)}$ is the rate used at epoch t.

In the original paper, the learning rate schedule is *fixed* (constant across epochs)

$$\alpha_{(t)} = c$$

This is not ideal

- slowing the learning rate
- as the number of epochs increases
- is more common
    - avoids catastrophic forgetting

The newer paper shows that a fixed learning rate *over-estimates* $L(N, D)$ when $D < 130B$

- leading to mis-fitting the empirical relationship

This can be avoided via a *variable learning rate* that decays $\alpha_{(t)}$

- to a fixed fraction of the initial rate $\alpha_{(0)}$
- as epoch number t increases

Hence the optimal relationship changes from $\frac{N}{D} = \text{constant}$ to $\frac{N^{0.74}}{D} = \text{constant}$

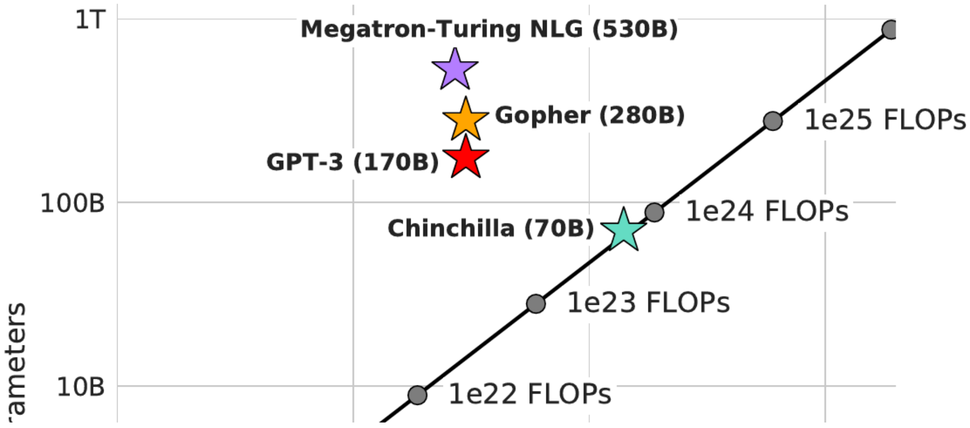
The future of large models as seen through Scaling laws

The Scaling laws suggest

- given a fixed compute budget C
- there is an optimal number of parameters N and number of training tokens D

The line in the graph below shows the N and D for each level of C.

Chinchilla Optimal Training



1T
100B
10B
rameters
Megatron-Turing NLG (530B)
Gopher (280B)
GPT-3 (170B)
Chinchilla (70B)
1e25 FLOPs
1e24 FLOPs
1e23 FLOPs
1e22 FLOPs

Let's evaluate GPT-3, which pre-existed the scaling laws, in terms of what we now know.

- $N_{\text{GPT}} = 175B$
- $D_{\text{GPT}} = 0.3T$

According to the Scaling Laws, GPT-3 is *under-trained* in both time and size of training data

$$
\begin{aligned}
C^*(N_{\text{GPT}}, D_{\text{GPT}}) &= 4.4 * 10^{24} \text{ Flops} \\
D^*(N_{\text{GPT}}, D_{\text{GPT}}) &= 4.2 \text{ TB tokens}
\end{aligned}
$$

GPT-3 is *under-trained* in time by a large factor

$$\frac{C^*(N_{\text{GPT}}, D_{\text{GPT}})}{C_{\text{GPT}}} = \frac{4.4 * 10^{24}\ \text{Flops}}{3.1 * 10^{23}\text{Flops}} > 10$$

GPT-3 is *under-trained* in number of tokens by a large factor

$$\frac{D^*(N_{\text{GPT}})}{D_{\text{GPT}}} = \frac{4.2TB}{0.3TB} > 10$$

One implication of these results is

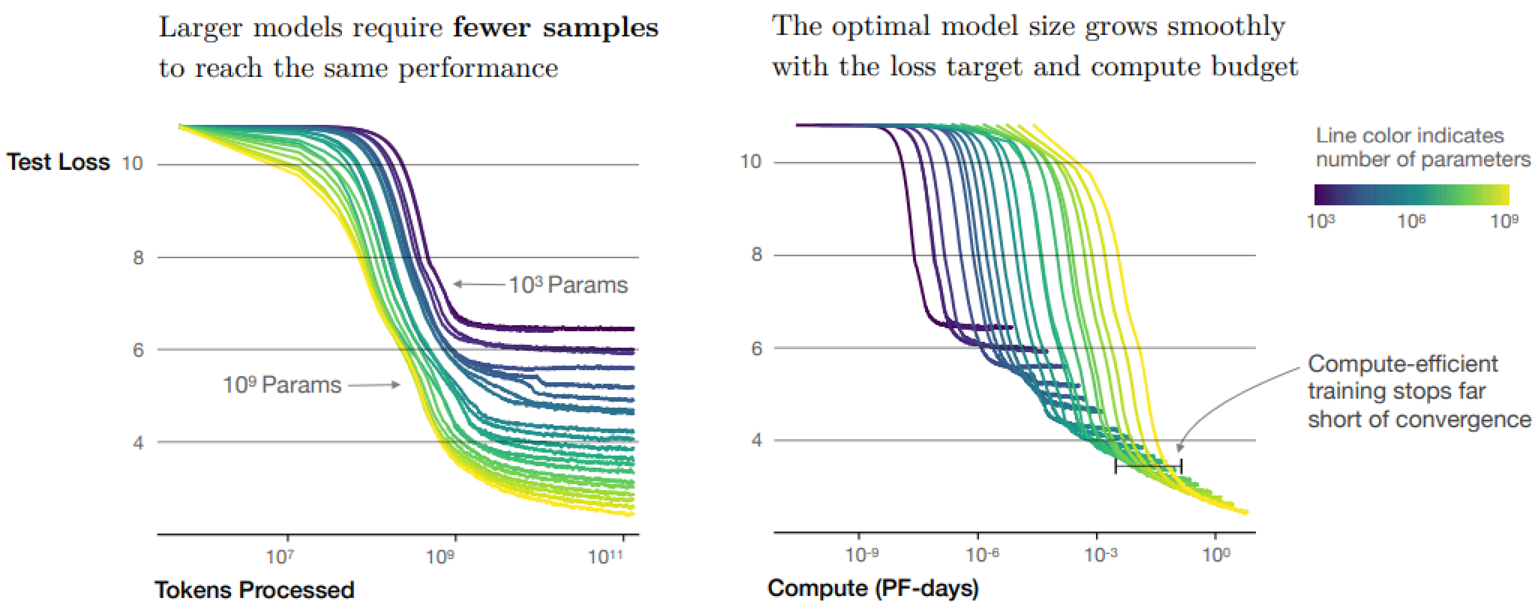
- it may not be practical (in terms of compute budget) to *optimally* train models with $N > N_{\text{GPT}}$
- A 10 trillion parameter model needs 100 times the compute used for GPT-3

Referring to the chart below, we compare a smaller model (purple line) to a larger one (yellow line)

- for a fixed performance $L = 8$

The Scaling Laws show

- a smaller model (purple line), compared to a larger one (yellow line)
  - may achieve L, but needs a bigger D
  - even though D is bigger, the smaller N may result in a smaller C



Given that reality, a likely future world is one of

- smaller N
- trained on *a lot* of data (to the point of non-decreasing loss)
- resulting in *better* performance L than a larger model

The authors validated this hypothesis by comparing two models

- started with a large model called Gopher with $N_{\text{Gopher}} = 280B$
- trained a smaller model called Chinchilla with $N_{\text{Chinchilla}} = 70B$
- using the same compute $C_{\text{Chinchilla}} = C_{\text{Gopher}}$
- but optimal D: $D_{\text{Chinchilla}} = 1.4T$

Chinchilla, although only 25% as large as Gopher

- outperforms on many benchmarks

The Scaling Laws are driving the design of new models.

- There are "clones" of GPT-3 with similar (or better) performance and many fewer parameters
    - at a greatly reduced compute budget.
    - LLaMA (https://arxiv.org/pdf/2302.13971v1.pdf): 13B parameters
        - From Meta. Model weights *are not* freely available
    - BLOOM (https://huggingface.co/docs/transformers/model_doc/bloom)
        - family of models from the BigScience Workshop (https://bigscience.huggingface.co/); Open-source
- The successor (PaLM 2 (https://ai.google/static/documents/palm2techreport.pdf)) to Google's $540B$ parameter PaLM model
    - has only $16B$ parameters (in its largest configuration) but performs at a similar levelmm

Another trend (unrelated to scaling) is to incorporate *non-parametric* knowledge into models

- e.g., the Web as a source of "world" knowledge

With an external knowledge source, a model's parameters

- can be fewer
- encode "procedural" knowledge rather than factual knowledge

Thus, the trend towards models of ever increasing size is probably over.

Inference budget

We have been focused on the *training budget*

- cost of a forward pass
- cost of a backward pass
- summed over many training examples

But larger models also require more compute (and greater latency) at *inference* time

- typical way of increasing model size is stacking more Transformer blocks
- deeper stack
    - greater latency
    - increased compute

Since we train a model once

- but perform inference many times
- perhaps we should focus on the *inference budget* rather than the *training budget*

A detailed examination of [inference cost (https://kipp.ly/transformer-inference-arithmetic/#flops-counting)](https://kipp.ly/transformer-inference-arithmetic/#flops-counting) approximates the number of Flops F for once inference at

$$F = n_{\text{layers}} * 24 * d_{\text{model}}^2$$

Since N is proportional to n_{layers}

- inference cost is proportional to model size N

So a smaller model size N can reduce our inference budget.

Recall

- a larger model achieves a given loss level *on fewer tokens* then a smaller model

But, fixing the loss level

- a smaller model can achieve the same loss
- at the cost of training on more tokens

The smaller model's inference cost is lower

- so trading off more tokens for fewer parameters
- may be *inference time* optimal

This idea was explored in (LlaMa (https://arxiv.org/pdf/2302.13971.pdf))

- optimizes inference
- by training smaller models
- on *more data* than required for a larger model to achieve the same loss

LIAMA loss vs training tokens

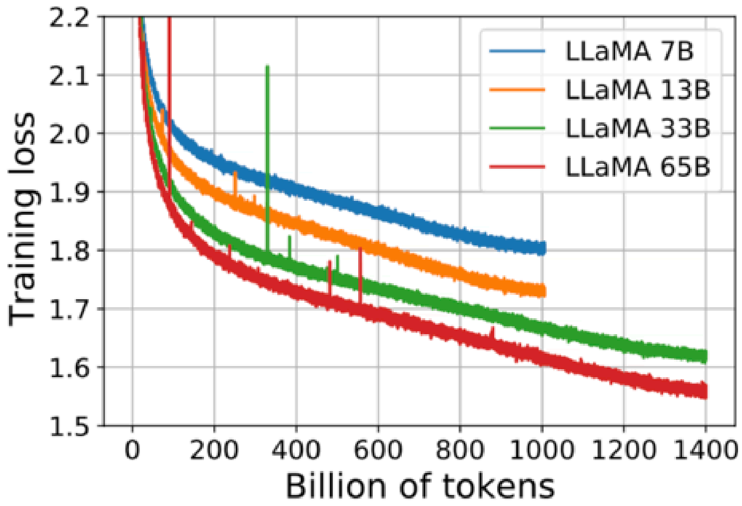


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

Attribution: https://arxiv.org/pdf/2302.13971.pdf#page=3

From the above graph: consider a Loss Level of 1.7

- the $N = 65$B model requires $D = 600$B tokens
- the $N = 33$B model requires $D = 800$B tokens

The author suggests that, for the same loss level, we can trade off

- doubling N
- for a decrease of D by 40%

Thus, when we are **not trying to optimize for training compute**, it make sense

- to train a small model
- on greater than "compute-optimal" D
- because the loss will continue to decrease

This leads us to

- small models
- with loss as good as a larger model
- with better **inference** time speed
- at the cost of "excessive" (relative to compute optimal) training compute and data

The result is that the current trend in LLM's is to

- small models
- trained on many tokens
- so as to optimize the inference budget

Summary

The Scaling laws demonstrate that

- Larger models have the *potential* for smaller loss than smaller models
    - but require more (compared to a smaller model) data and compute to achieve their potential
- For a fixed Loss level, compared to a smaller model
    - larger models achieve the loss with fewer tokens (more data efficient)
    - **but** with a larger compute requirement (higher **training budget**)
    - a smaller model can achieve the same Loss
        - using more training tokens
        - **and** is more cost effective at inference time

Thus, the trend is toward smaller models.

We recommend Chinchilla's Death (https://espadrine.github.io/blog/posts/chinchilla-s-death.html) for a comprehensive understanding of the Scaling Laws.

In [2]:

```
print("Done")
```

```
Done
```