Bigger = Better ? Scaling laws

There are many LLM's, with varying choices of

- number of parameters N
- size of training dataset D
- 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.

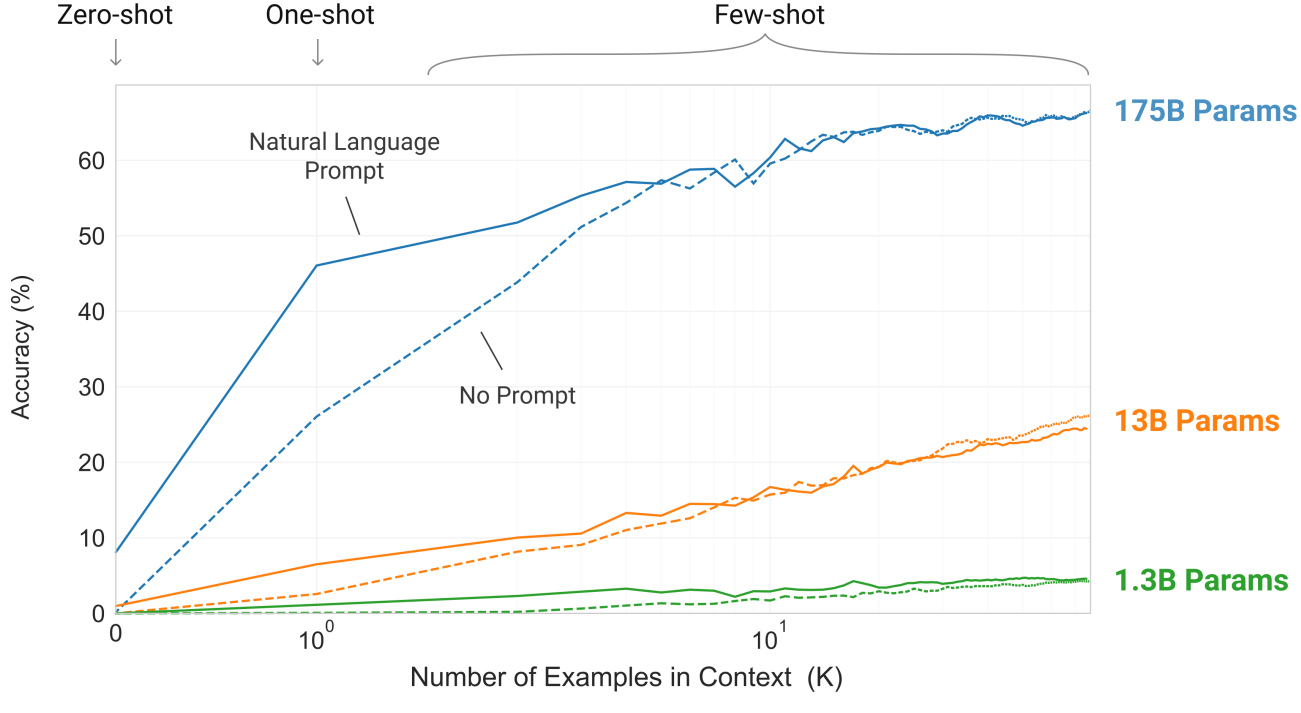
Calculations can be seen in Table D.1 and are explained within the table caption.

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.

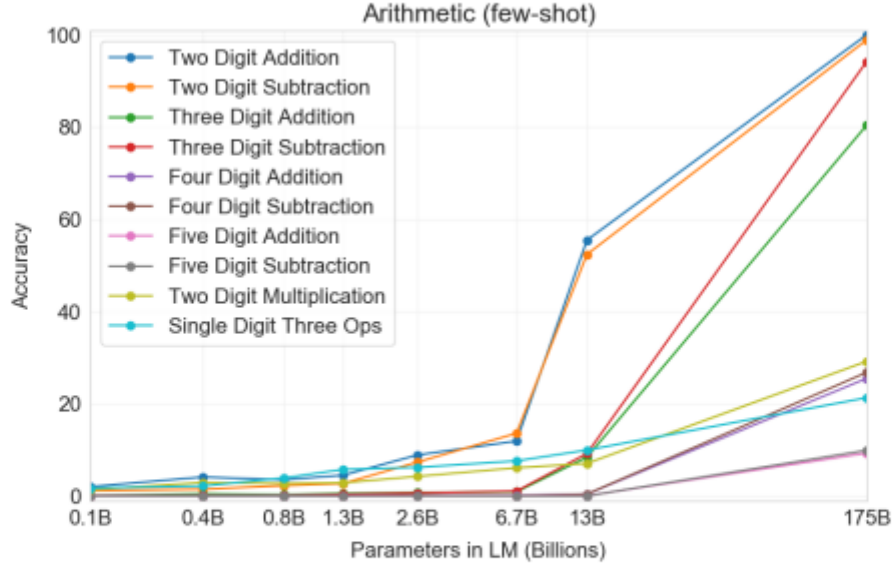


Figure 3.10: Results on all 10 arithmetic tasks in the few-shot settings for models of different sizes. There is a significant jump from the second largest model (GPT-3 13B) to the largest model (GPT-3 175), with the latter being able to reliably accurate 2 digit arithmetic, usually accurate 3 digit 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)]

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)

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

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 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

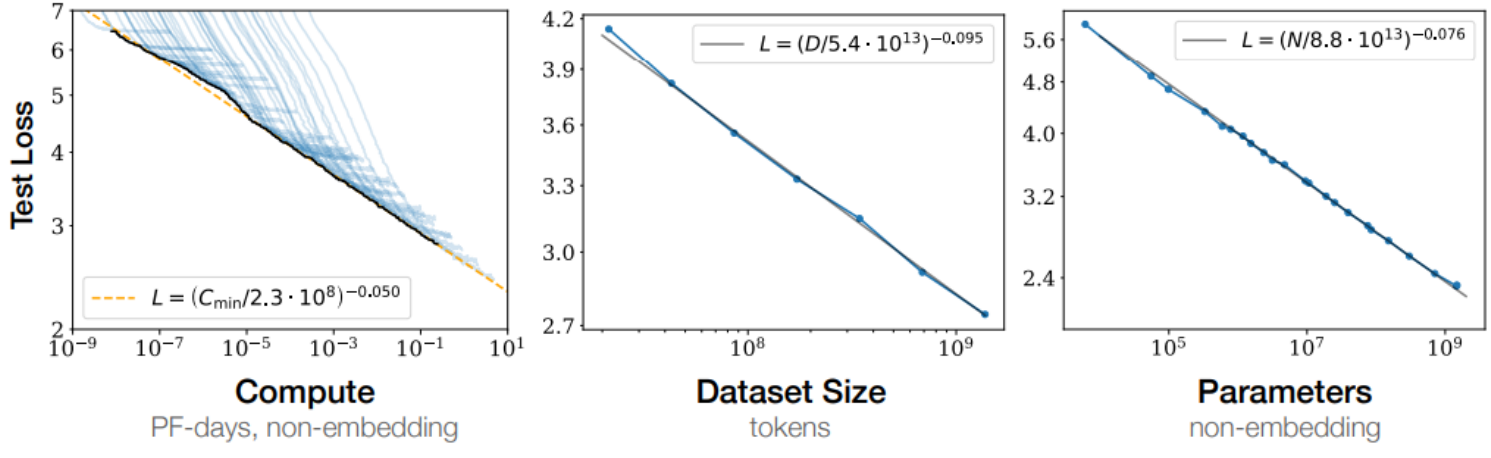
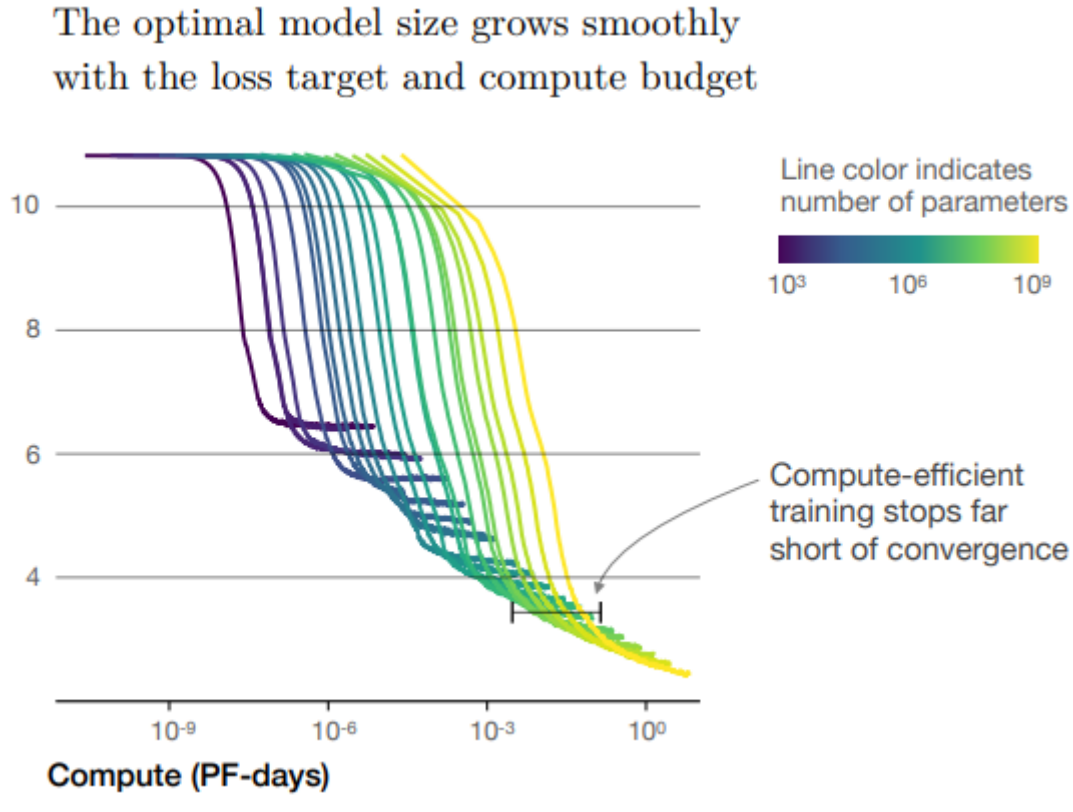


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when 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

For a fixed Compute C

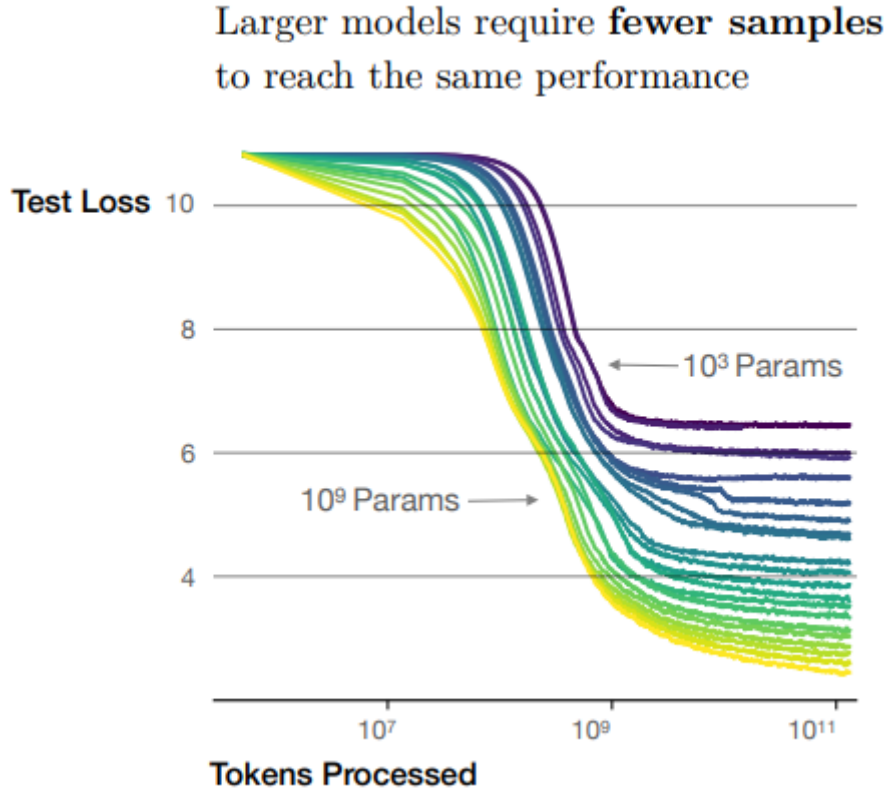
- a smaller model (that has reached its asymptotic minimum) has better performance
- provided that there is enough training data

This is interesting in that more data D may compensate for fewer parameters

- we may be able to create "small" models (fewer parameters)
- with performance equal to larger models
- given sufficient D

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 have better performance
- but at a higher C

Here is one graph that combines N and D

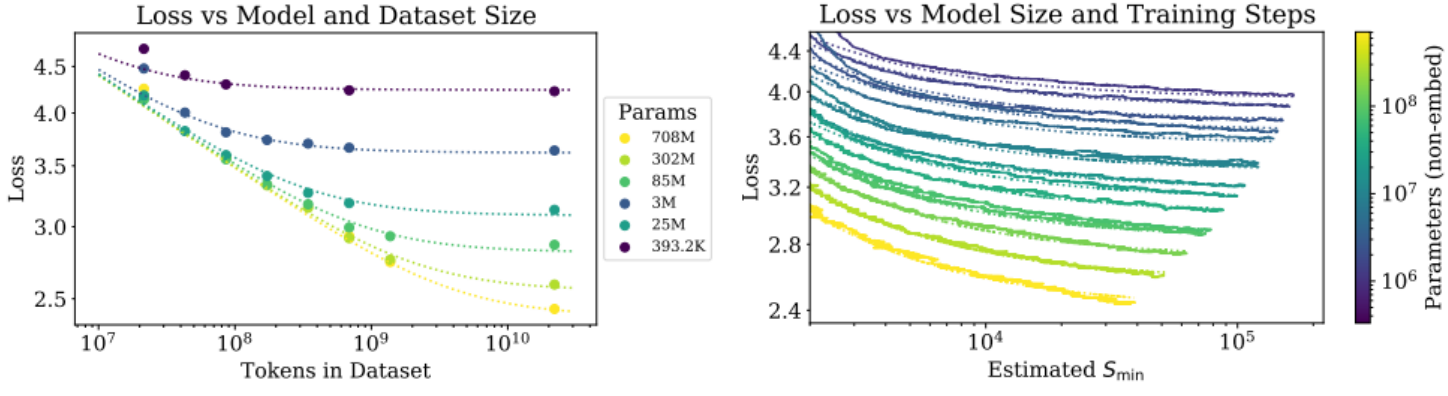


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of $S_{\min}$,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

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

More Recent results

Answering the same question as the original paper (https://arxiv.org/pdf/2001.08361.pdf)

- a more general approach (https://arxiv.org/pdf/2203.15556.pdf) to the same question
- leads to somewhat different conclusions

Stated more directly, the paper proposes an empirical function to estimate the optimal N and D

- for a fixed compute budget C

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

where $L(N, D)$ is the early-stopped loss

- not trained to optimal converged L
- which would require more than the compute budget C

One point of departure between the two papers:

- the second paper uses a *learning rate schedule* that varies with D
    - decay to a fixed fraction of the initial rate, based on length of D
- versus a *fixed* learning rate schedule used by the first paper

The second paper contends that

- failing to use a variable learning rate
- causes an *over-estimate* of L when $D < 130B$

Using the overestimate in fitting an empirical function causes a difference in conclusions.

- The second paper concludes that D should grow linearly with N
- rather than sub-linearly ($D = N^{0.74}$)

In comparing the optimal values for a variable (e.g., C) between paper 1 and paper 2

- we use subscript j to refer to the value in paper $j \in \{, 2\}$

Here are some conclusions (https://arxiv.org/pdf/2203.15556.pdf#page=7) offered in the second paper

- Most LLM's use an N that is *too large* given their *compute budgets*
- For $N = 175B$ (GPT-3), an optimal version of GPT-3
    - needs to be trained longer than the actual
        - $C_2 = 4.4 * 10^{24}$ Flops versus actual $C_1 = 3.1 * 10^{23}$
    - on more tokens D
        - $D_2 = 4.2T$ versus $D_1 = 0.3T$
- For current models much larger than GPT-3 ($N > 175B$)
    - the optimal C_2 and D_2 are not realistic in practical terms

Using the projected optimal values

- the second paper started with a model called Gopher with $N = 280B$
- set a compute budget equal to that used for Gopher
- to derive an optimal $N_2 = 70B$ and $D_2 = 1.4T$
- and trained a smaller model called Chinchilla

Chinchilla, although only 25% as large as Gopher

- outperforms on many benchmarks

So perhaps the future will see

- a trend to smaller models
- with more data

This may be particularly relevant

- with the use of non-parametric knowledge (external knowledge sources, like the Web)
- naturally reduced N

Test-time cost versus Train-time cost

We have been focused on the cost of *training*

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

Post-training, at test time, the cost of prediction is

- cost of a forward pass

The way that N (number of parameters) usually increases in a Transformer Architecture

- is by stacking an increasing number of Transformer blocks.

This increases the path length of a forward path.

So making *predictions* using a bigger model will be more costly than doing so in a smaller model.

If you are running a prediction service at large scale (e.g., ChatGPT)

- you need increased compute
- to support the same number of predictions
- on a bigger model than a smaller one.

So smaller models have test-time as well as train-time advantages.

In [2]:
```
print("Done")
```

```
Done
```