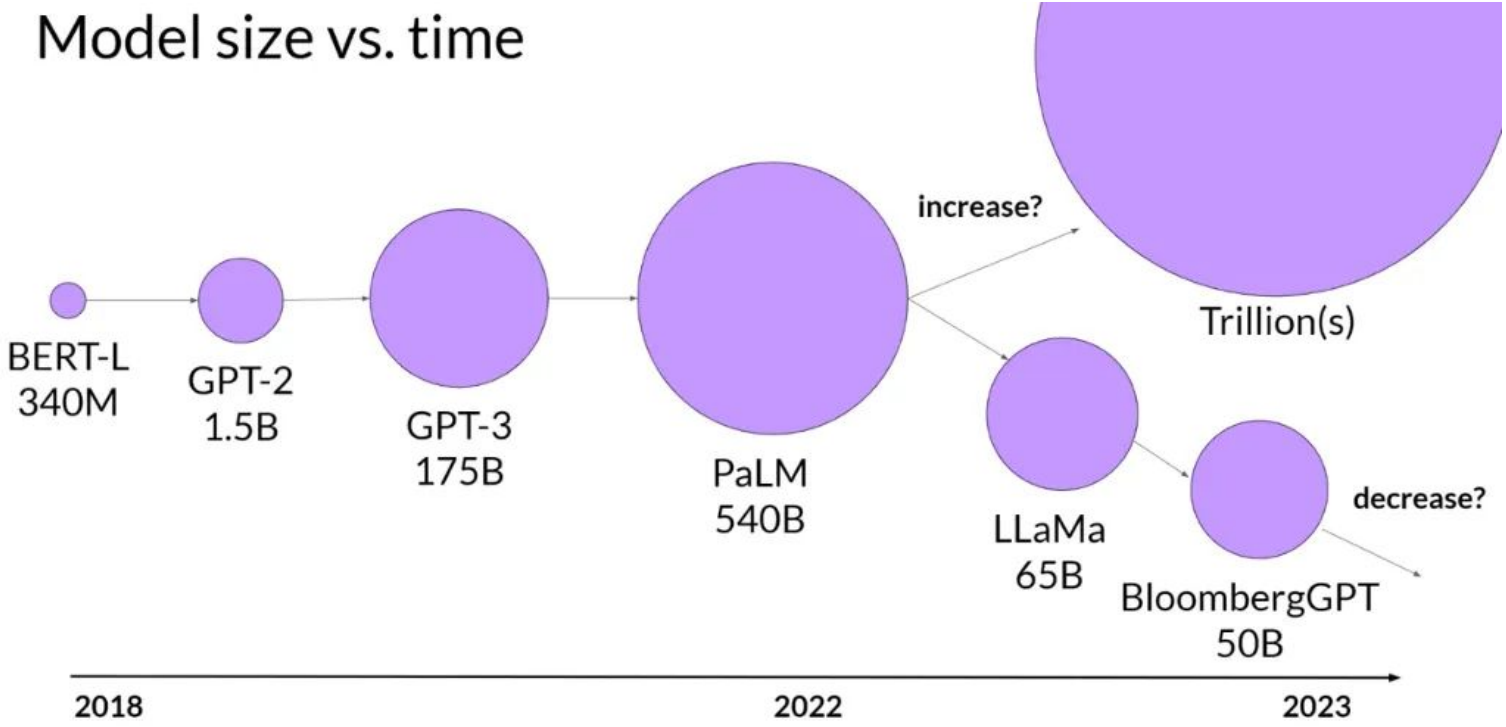
Large Language Models

Scaling Laws

ELL881 · AIL821

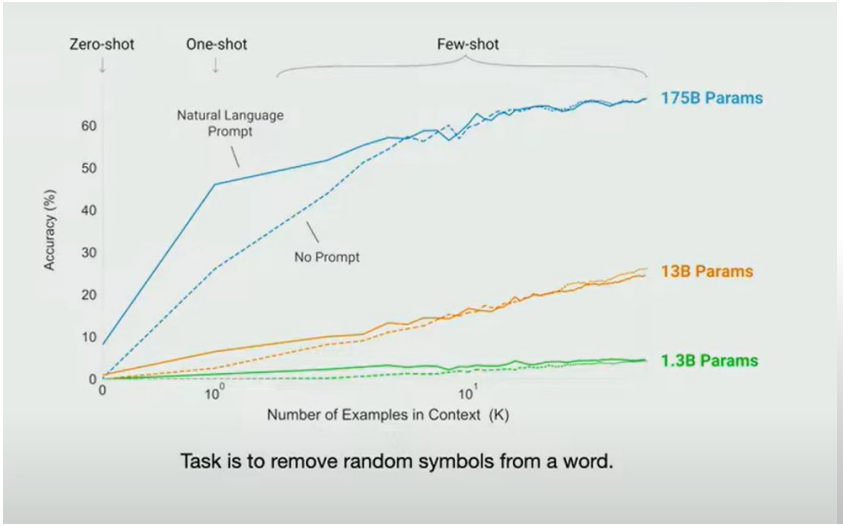
Sourish Dasgupta
Assistant Professor, DA-IICT, Gandhinagar
https://daiict.ac.in/faculty/sourish-dasgupta

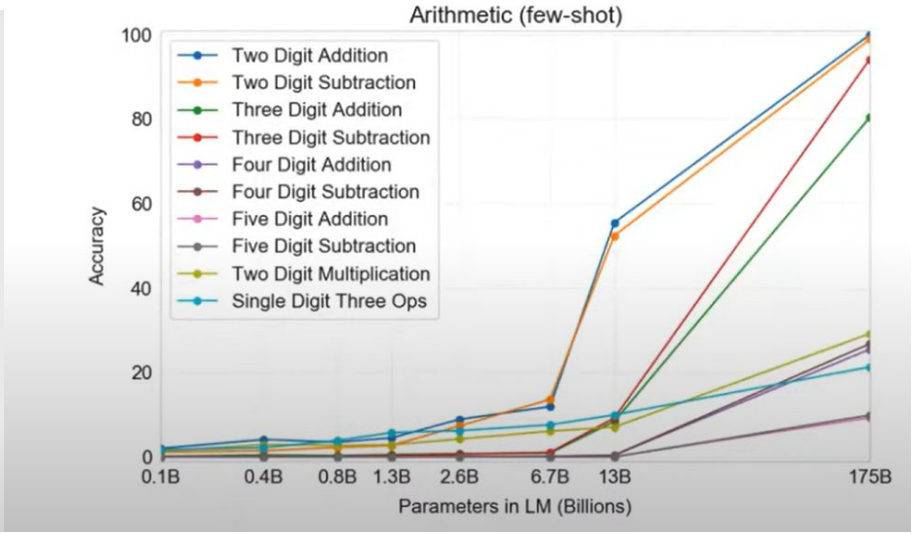
Model size vs. time



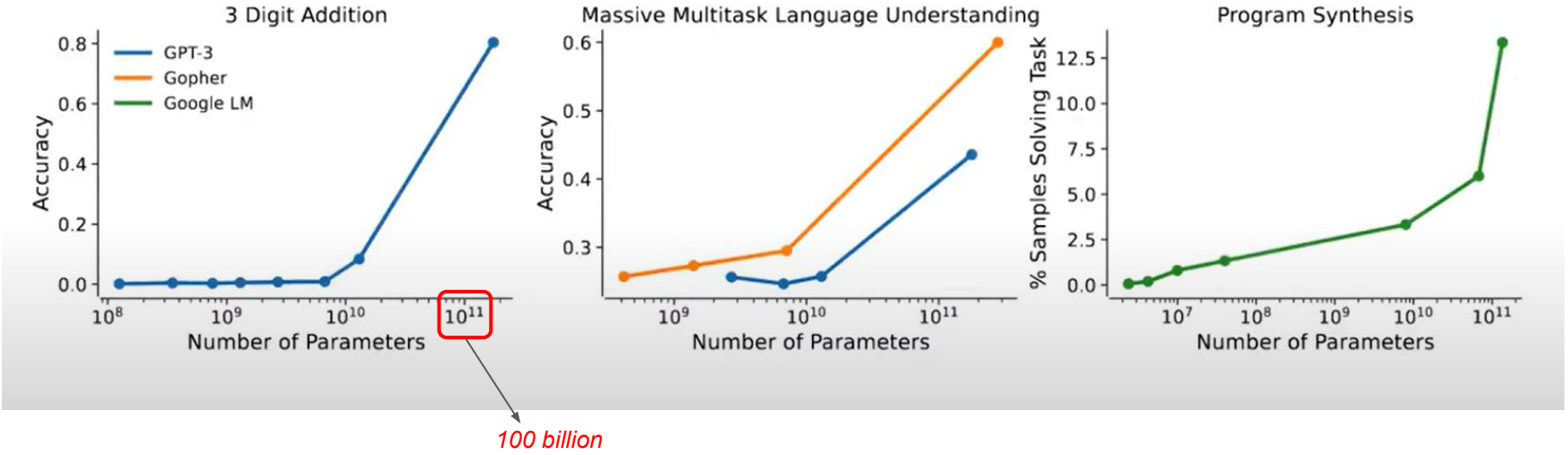
“*Emergent*” abilities in LLM

LLMs are few-shot learners (*and larger is better!*)

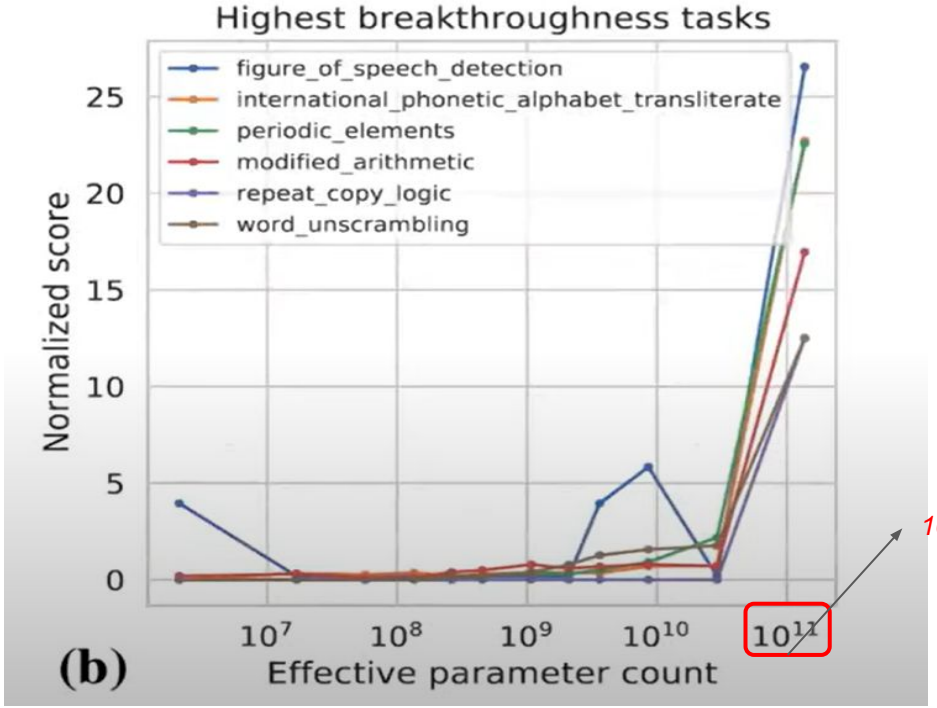




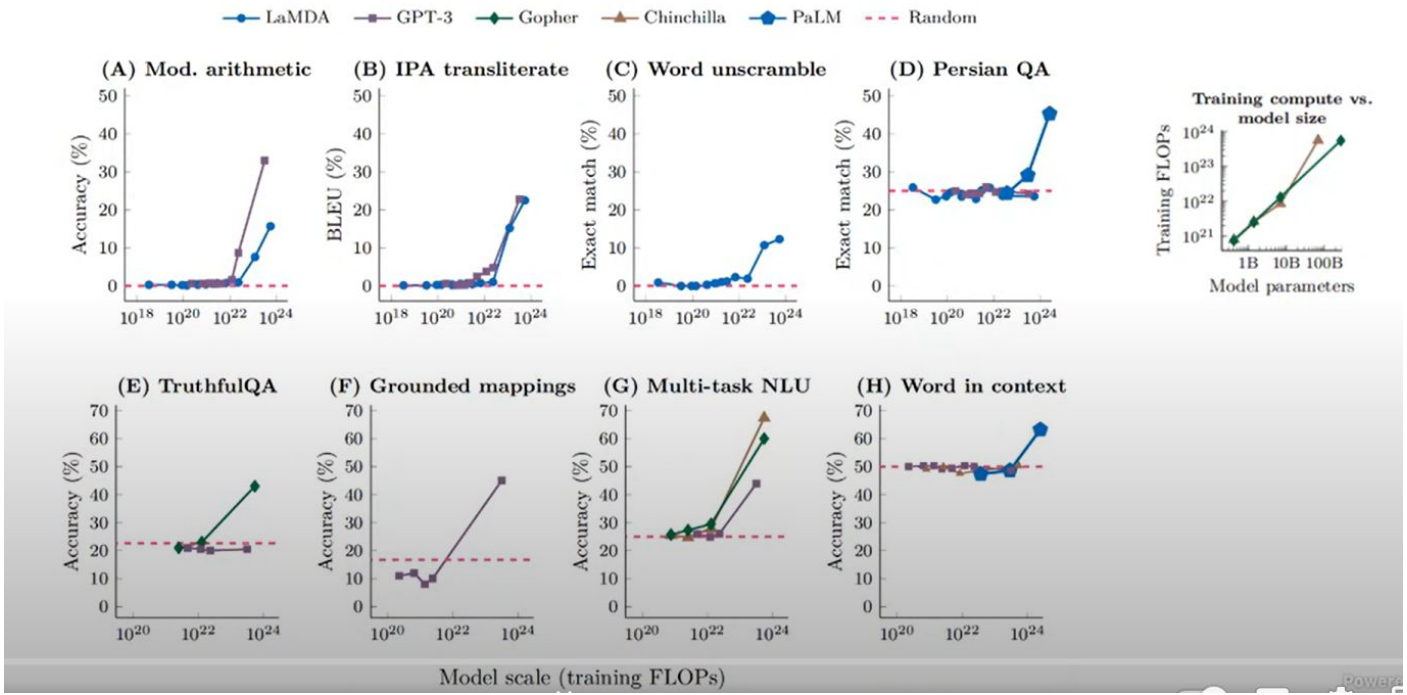
LLMs are few-shot learners (*and larger is better!*)



LLMs are few-shot learners (*and larger is better!*)



LLMs are few-shot learners (*more training is better too!*)



Google BIG-BENCH benchmark

Consider a single Model Family e.g. PaLM

Let x_n be the scale of one family member e.g. PaLM-540B

Let y_n be the family member's score on some Task and Metric

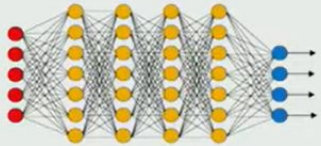
Sort the pairs (x_n, y_n) by model scale x_n, smallest to largest

$$\text{Emergence Score}\left(\left\{(x_n, y_n)\right\}_{n=1}^{N}\right) \overset{\text{def}}{=} \frac{\text{sign}(\arg\max_i y_i - \arg\min_i y_i)(\max_i y_i - \min_i y_i)}{\sqrt{\text{Median}(\{(y_i - y_{i-1})^2\}_i)}}$$

LLMs “*seems*” to get more intelligent with the following:

NT: scale *w/o bottleneck*

Amount of training data



Model size (parameters)

Amount of compute (or time)

α x Model size x Training data = Training compute

~ 6 FLOPs (Floating-point Operations)

Recap on Parameter size & FLOPs

Operation	Parameters	FLOPs per Token
Embed	$(n_{\text{vocab}} + n_{\text{ctx}})\, d_{\text{model}}$	$4d_{\text{model}}$
Attention: QKV	$n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$	$2n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$
Attention: Mask	—	$2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$
Attention: Project	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2n_{\text{layer}} d_{\text{attn}} d_{\text{embd}}$
Feedforward	$n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$	$2n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$
De-embed	—	$2d_{\text{model}} n_{\text{vocab}}$
Total (Non-Embedding)	$N = 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}})$	$C_{\text{forward}} = 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$

one PF-day $= 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ floating point operations.

Emergent abilities are *unpredictable*

If that's true, we never get to know the following:

- *Which* abilities (and *when*) exactly will emerge?
- *What* controls the trigger?
- Can we make desirable abilities to emerge *faster*?
- Can we make undesirable abilities to be suppressed?

Is the value of scaling laws only in predicting?

- How much *return* for a given compute (resource) budget?
- How to *allocate* the compute budget - model size vs. dataset size?

Can there be a curve that fits “emergence”? *Intuition*

Input: $x_1 \ \ldots x_n \ \sim N(\mu, \sigma^2)$

Task: estimate the average as $\hat{\mu} = \frac{\sum_i x_i}{n}$

What’s the error? By standard arguments..

$$\mathrm{E}[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$$

This is a scaling law!!

$$\log(Error) = -\log n + 2\log \sigma$$

More generally, any polynomial rate $1/n^{\alpha}$ is a scaling law

Source: CS-324, Stanford University

What about fitting “emergence” in *non-parametric* setting

Neural nets can approximate arbitrary functions. Lets turn that into an example.

Input: $x_1 \dots x_n$ uniform in 2D unit box. $y_i = f(x_i) + N(0,1)$
Task: estimate f(x)

Approach: cut up the 2D space into boxes with length $n^{-\frac{1}{4}}$, average in each box

What’s our estimation error?

Informally, we have $\sqrt{n}$ boxes, each box gets $\sqrt{n}$ samples.

$$Error \approx \frac{1}{\sqrt{n}} + (other\ smoothness\ terms)$$

In d-dimensions, this becomes $Error = n^{-1/d}$ - **This means scaling is $\boldsymbol{y = -\frac{1}{d}x + C}$**

Takeaway: flexible ‘nonparametric’ learning has dimension dependent scaling laws.

Source: CS-324, Stanford University

Notations

$C \approx 6N$ floating point operators per training token.

$C = 6NBS$ estimates the (non-embedding) compute used at batch size B

S is the number of training steps (ie parameter updates)

$S_{\min}$ – an estimate of the minimal number of training steps needed to reach a given value of the loss. This is also the number of training steps that would be used if the model were trained at a batch size much greater than the critical batch size.

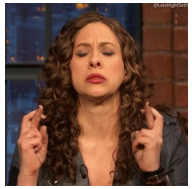
one PF-day $= 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ floating point operations.

B_{crit} – the critical batch size [MKAT18], defined and discussed in Section 5.1. Training at the critical batch size provides a roughly optimal compromise between time and compute efficiency.

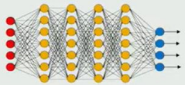
$C_{\min}$ – an estimate of the minimum amount of non-embedding compute to reach a given value of the loss. This is the training compute that would be used if the model were trained at a batch size much less than the critical batch size.

What if the loss-drop (i.e., emergence) follows power-law?

$y = ax^k$ ⟶ Power Law: *can be -ve* ⟶ $L(X) \propto 1/X^{\alpha x}$



$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} \quad \alpha_N \sim 0.076, \quad N_c \sim 8.8 \times 10^{13} \text{ (non-embedding parameters)}$$



$$L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D} \quad \alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13} \text{ (tokens)}$$

Kaplan Laws

$$L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C} \quad \alpha_C = 0.057 \quad C_c = 1.6 \times 10^7 \text{ PF-days}$$

$$L(C_{\min}) = \left(C_c^{\min}/C_{\min}\right)^{\alpha_C^{\min}} \quad \alpha_C^{\min} \sim 0.050, \quad C_c^{\min} \sim 3.1 \times 10^8 \text{ (PF-days)}$$

A more reliable version:

$$L(C_{\min}) = \left(C_c^{\min}/C_{\min}\right)^{\alpha_C^{\min}}$$

$$\alpha_C^{\min} \sim 0.050, \quad C_c^{\min} \sim 3.1 \times 10^8 \text{ (PF-days)}$$

$$C_{\min}(C) \equiv \frac{C}{1 + B/B_{\text{crit}}(L)} \quad \text{(minimum compute, at } B \ll B_{\text{crit}})$$

$$B_{\text{crit}}(L) \approx \frac{B_*}{L^{1/\alpha_B}}$$

$B_* \sim 2 \cdot 10^8$ tokens, $\alpha_B \sim 0.21$

$$L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C}$$

Parameters	Data	Compute	Batch Size
Optimal	∞	C	Fixed

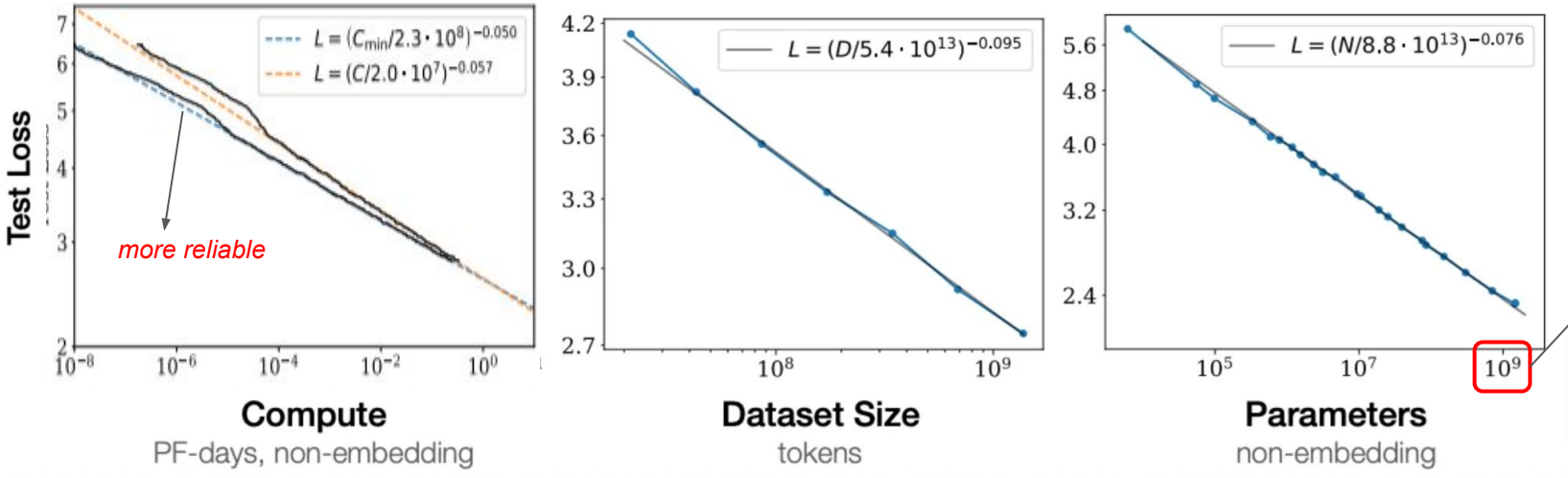
VS.

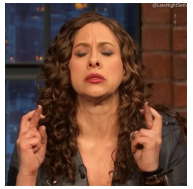
N_{opt}	D_{opt}	$C_{\min}$	$B \ll B_{\text{crit}}$

Compute-Efficient Value	Power Law	Scale
$N_{\text{opt}} = N_e \cdot C_{\min}^{p_N}$	$p_N = 0.73$	$N_e = 1.3 \cdot 10^9$ params
$B \ll B_{\text{crit}} = \frac{B_*}{L^{1/\alpha_B}} = B_e C_{\min}^{p_B}$	$p_B = 0.24$	$B_e = 2.0 \cdot 10^6$ tokens

Kaplan Laws

We are in luck! Turns out that scale ***is*** predictable

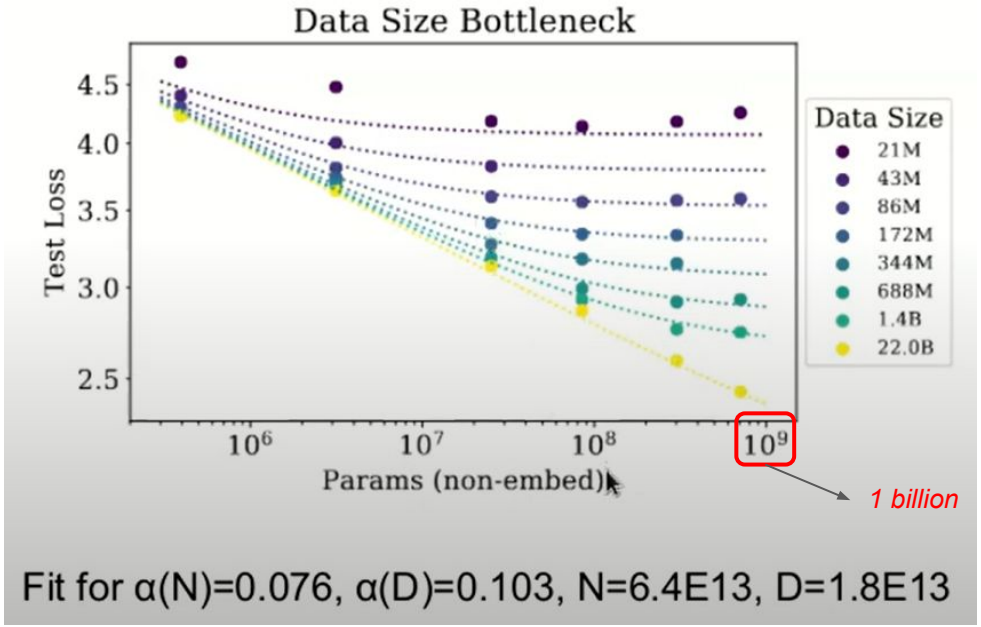


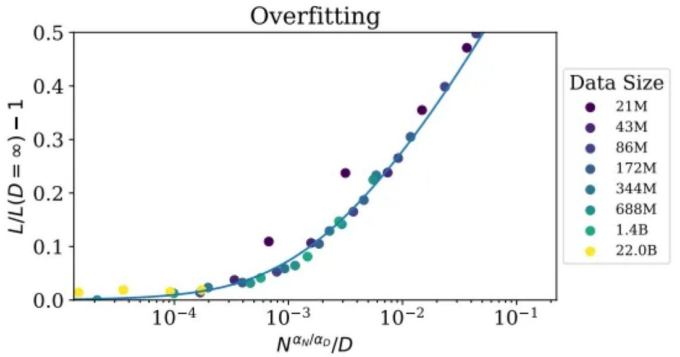


1 billion

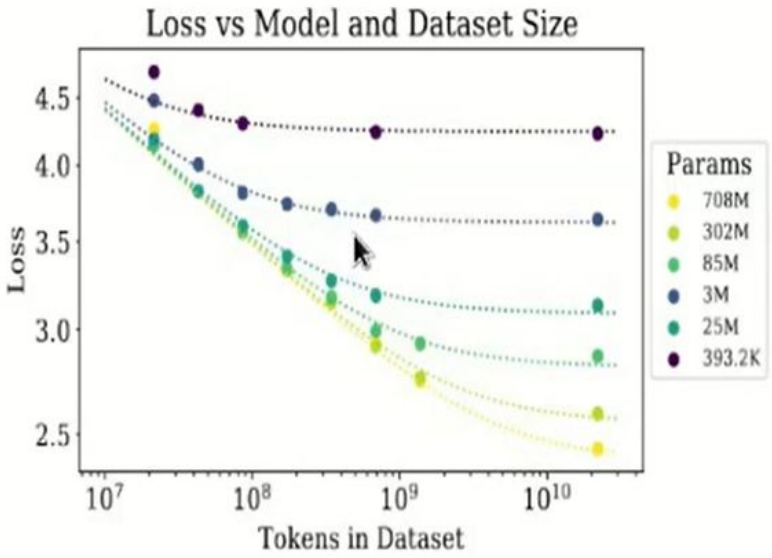
(NT: log-log plotting) → Now that's smooth!!

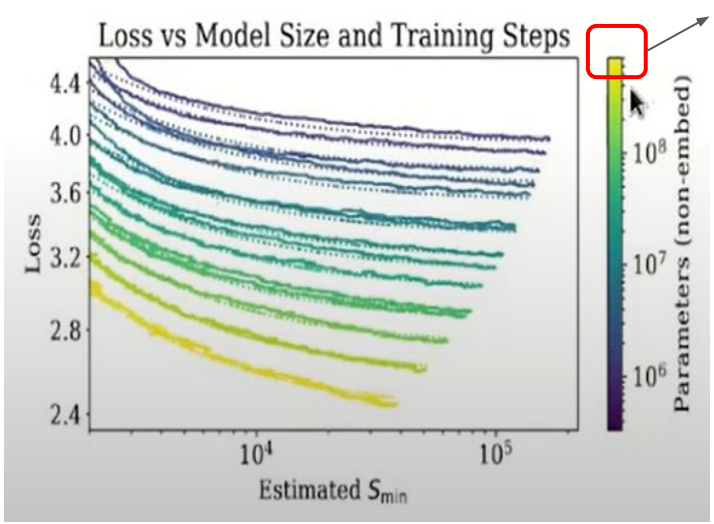
Observation 1a: *Universality of Overfitting*



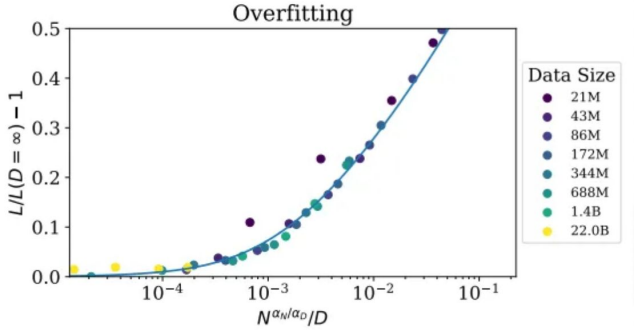


Observation 1b: *Sample Efficiency*





Key takeaway 1: *Both* parameter and dataset to be scaled



$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N}$$

$$L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D}$$

$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

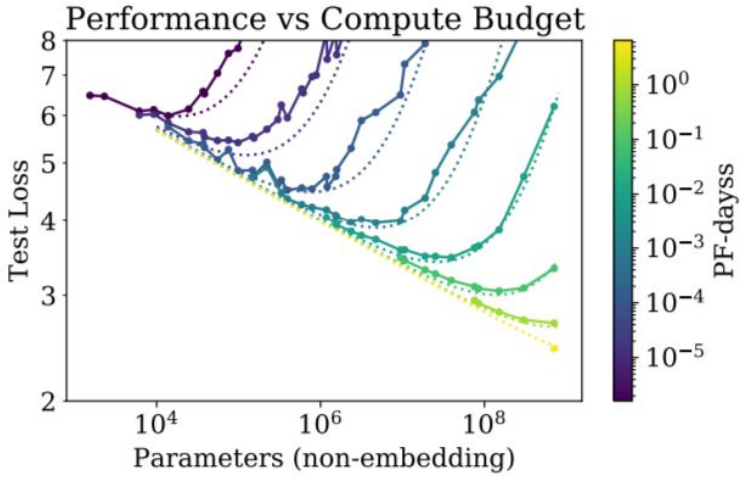
$$D \propto N^{\frac{\alpha_N}{\alpha_D}} \sim N^{0.74}$$

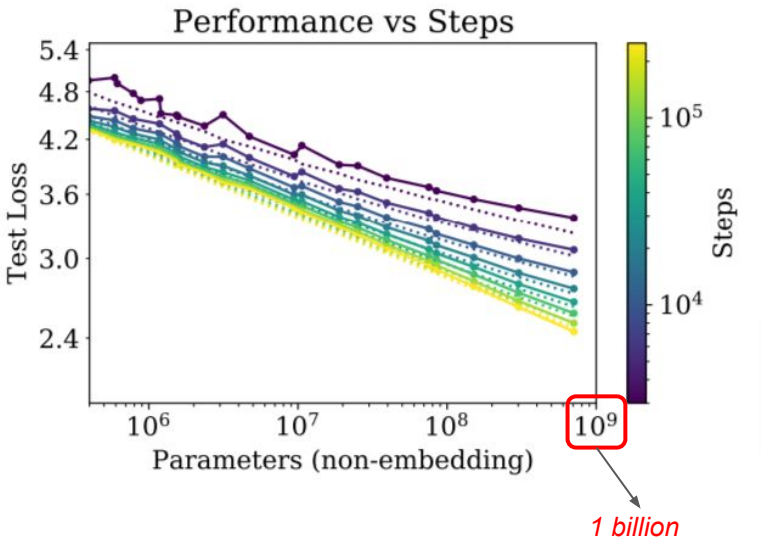
Performance penalty is $N^{0.75}$ / D

- if model increases 8x, dataset must increase 5x

Both needs to be scaled together

Observation 2: What about training time (*steps & FLOPs*)?





Key takeaway 2: *Universality of training*

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)}\right)^{\alpha_S}$$

$$L(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

$S_c \approx 2.1 \times 10^3$ and $\alpha_S \approx 0.76$, and $S_{\min}(S)$ is the minimum possible number of optimization steps

$$S_{\min}(S) \equiv \frac{S}{1 + B_{\text{crit}}(L)/B} \qquad \text{(minimum steps, at } B \gg B_{\text{crit}}\text{)}$$

$$B_{\text{crit}}(L) \approx \frac{B_*}{L^{1/\alpha_B}}$$

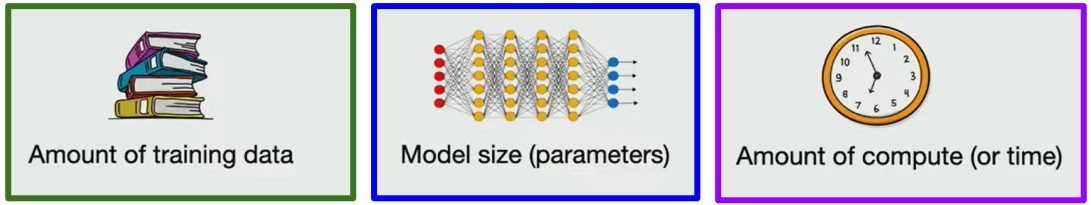
$B_* \sim 2 \cdot 10^8$ tokens, $\alpha_B \sim 0.21$

Parameter	α_N	α_D	N_c	D_c
Value	0.076	0.103	6.4×10^{13}	1.8×10^{13}

Parameter	α_N	α_S	N_c	S_c
Value	0.077	0.76	6.5×10^{13}	2.1×10^3

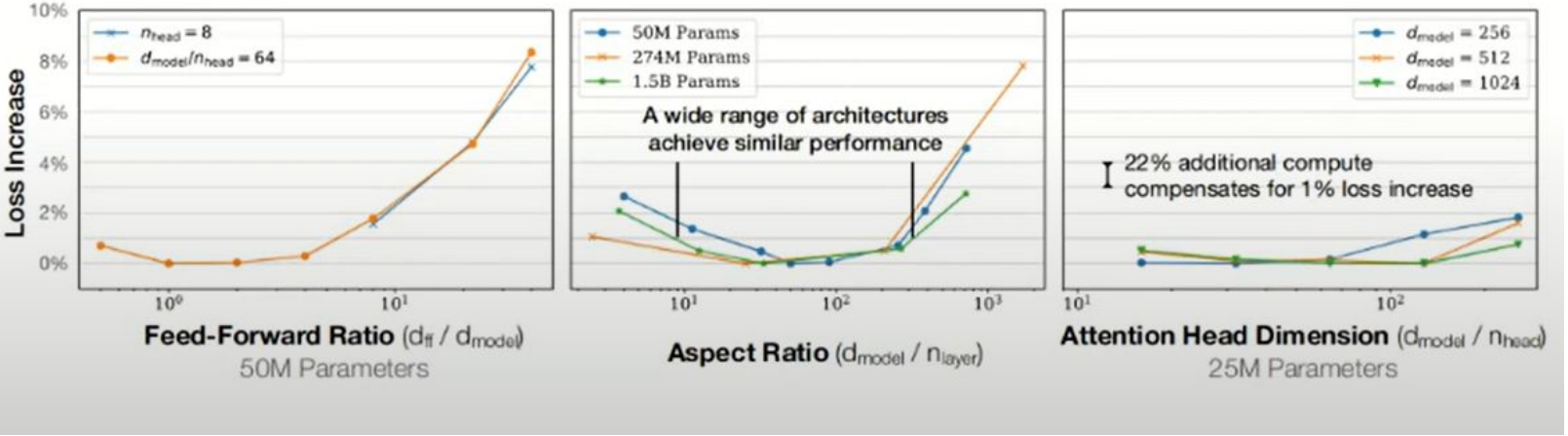
Kaplan Laws

Are we only to worry about

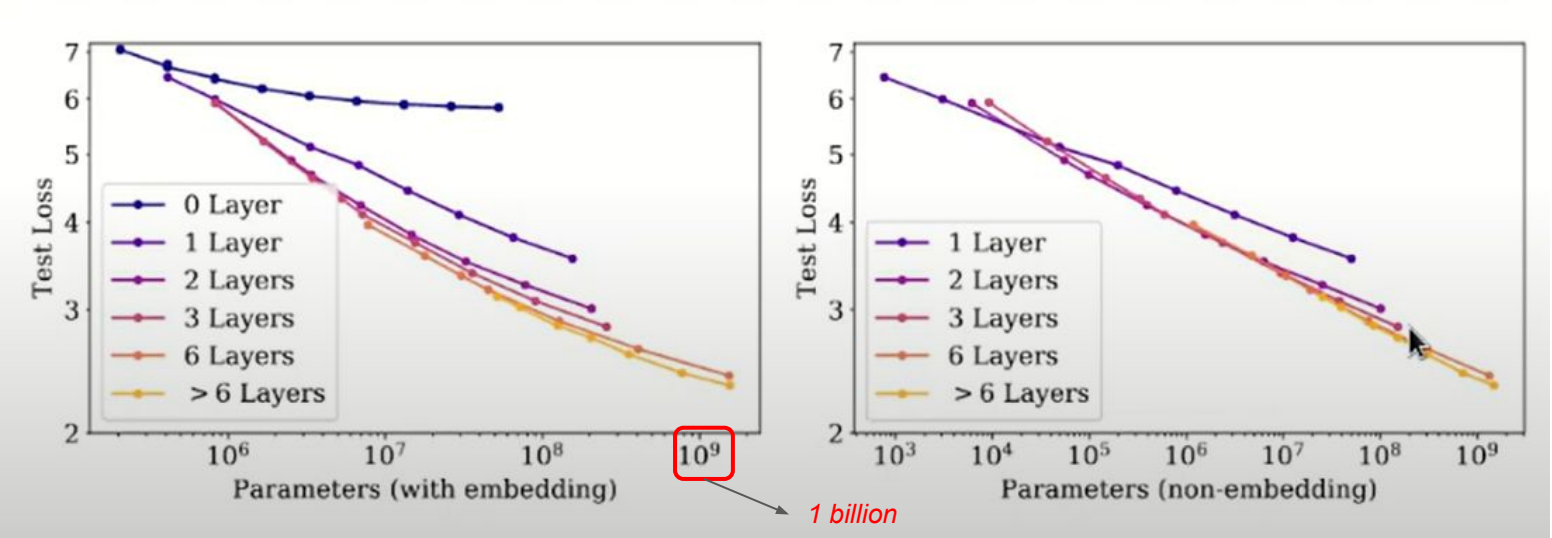


???

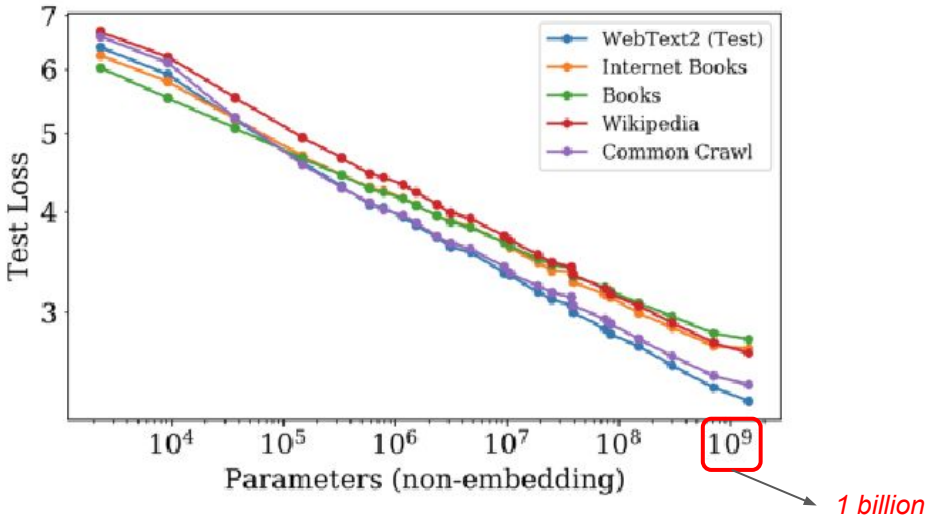
Key Takeaway 3: Model shape does not matter!



Key Takeaway 4: Embedding matrix does not matter!



Key Takeaway 5: Dataset *composition* does not matter!



Kaplan Scaling Laws at a glance:

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
$\alpha_C^{\min} = 0.050$	$C_c^{\min} = 3.1 \times 10^{8}$ 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

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}	$C_{\min}$	$B \ll B_{\text{crit}}$	$L(C_{\min}) = \left(C_c^{\min}/C_{\min}\right)^{\alpha_C^{\min}}$
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	$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S,B)}\right)^{\alpha_S}$