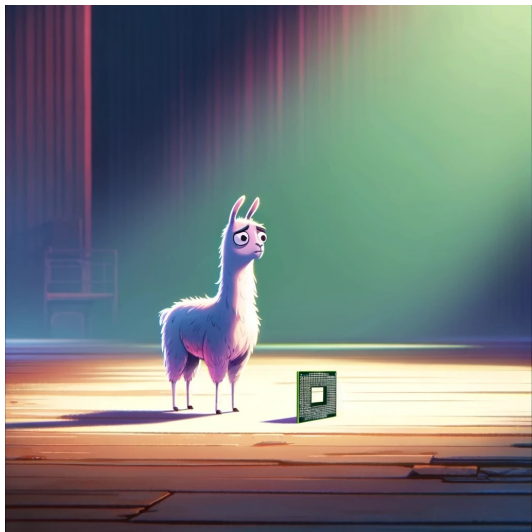
Deep Learning at Scale for the GPU poor - Part 1

Harry Coppock



Plan

- Tools for measuring scale – FLOPs + memory requirements
- Motivation – why scale matters
- How to get the most out of your resources
- Finetuning vs Prompt Engineering
- Foundation model safety/evals

Foundation model parameter count

- Transformers are typically described by the #parameters.
- But how does this translate to the computational requirements to store and run these models?

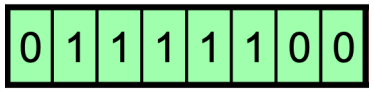
Model Name	# Parameters (Billion)
GPT3	175
Gopher	280
Llama 1 + 2	6-70
Mixtral	8*7
PaLM	540
GPT4	1700 (rumour)

Side note: A Fruit Fly has 100 million synapses
Side note: The Human brain has 100 trillion synapses

Floating Point System - Motivation

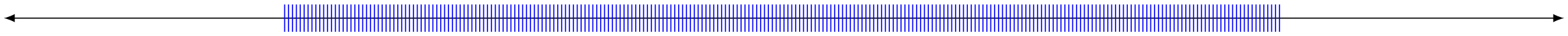
- There are in infinite amount of real numbers, $\mathbb{R}$.
- There are an infinite amount of numbers between any two numbers in $\mathbb{R}$.
- There are an infinite set of digits to describe each number in $\mathbb{R}$.
- This poses a great challenge in a world of finite memory and processing power.
- This is addressed through finite approximation

Floating Point System - Motivation

- bit – single binary digit (0 or 1)
- byte – 8 bit string
- Note: why only 0 or 1? – all computation occurs based on whether a current can flow through a transistor or not. It is on or off.
- Solution: represent a subset of real numbers with finite precision.
- Int8, each number represented by 8 bits. 

Pros: Each value takes up 1 byte of space.

Cons: Only 2^8 integers to be represented. We can only represent the integers between -127 and 127….

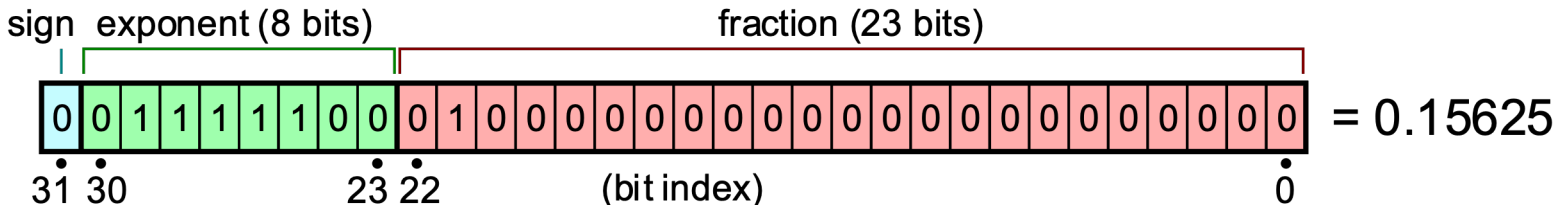


-128 0 127

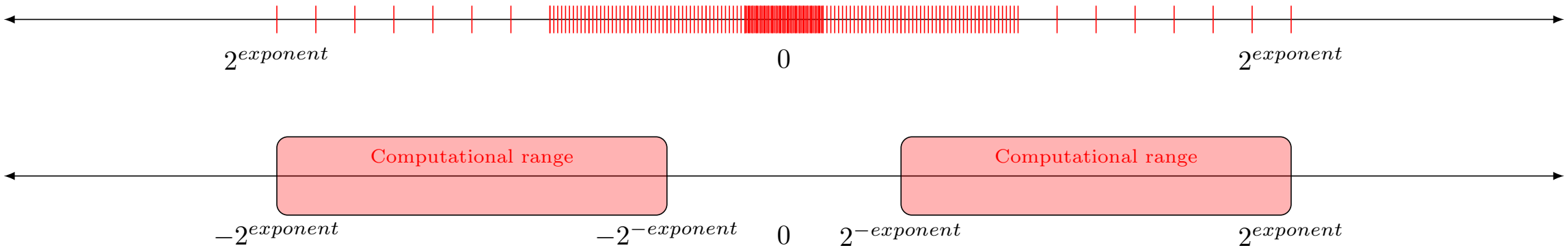
Floating point system

- Allow the point to ‘float’
- Float32 → 32 bits → 4 bytes

1.2 3 4 5



$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

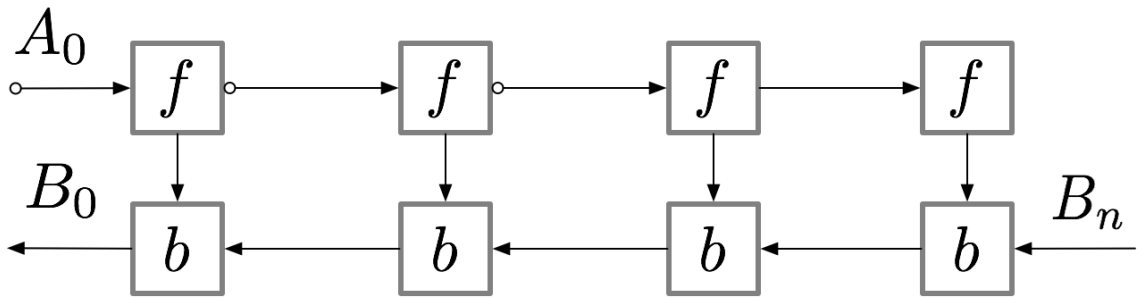


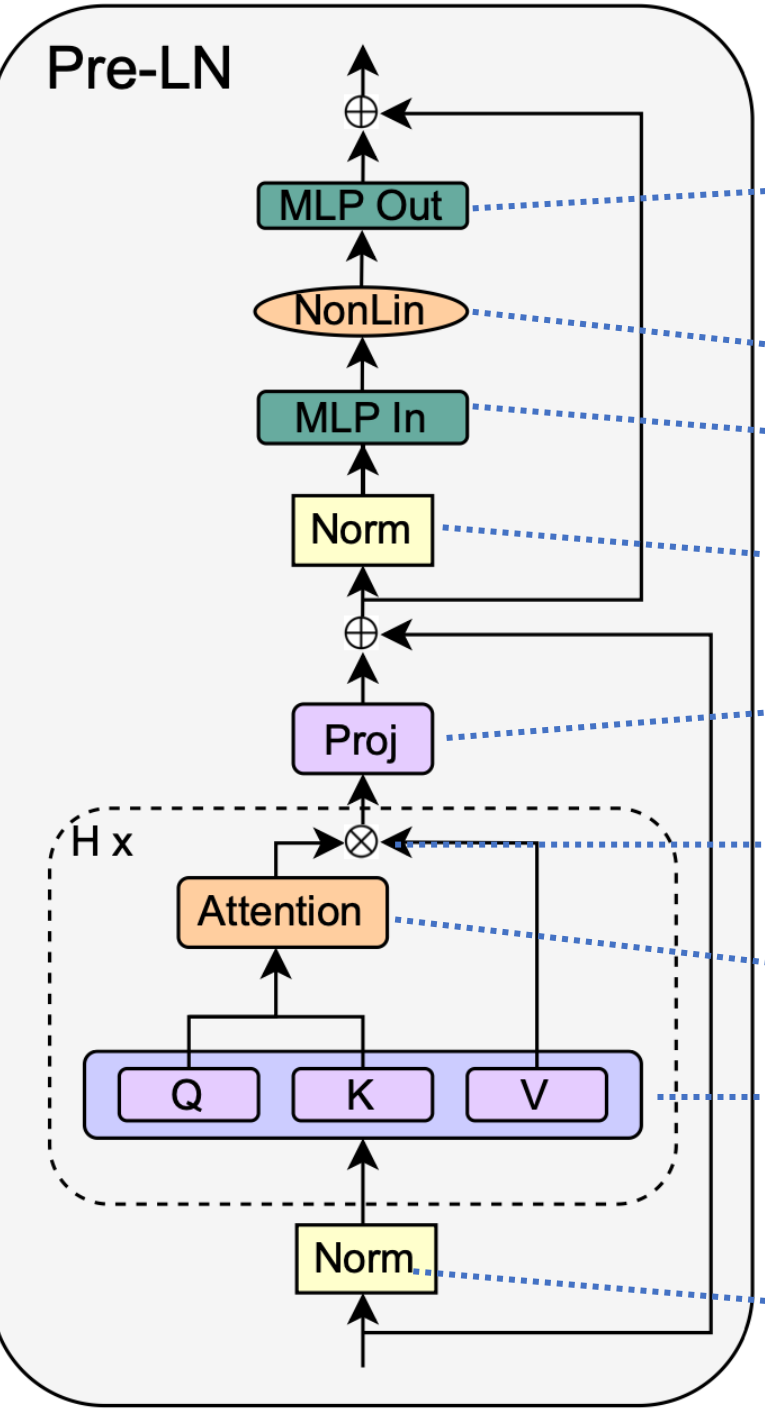
GPU Memory footprint for Llama 65B @32float

- Inference time:

  32 (bits)*65*10^9(params) = 2.08*10^{12}(bits)/8 = 260GB

- Training time
  - Model parameters = 260GB (same at inference time)
  - Gradients! 4*65*10^9 = 260GB
  - Optimiser: 2*4*65*10^9 (Adam coefficients momentum + variance) = 520GB
  - Activations for backward…





4*BND

4*BND *Note: typically, the hidden dim of the MLP is 4*D*

BND

BND

Projection: BND

- Output from softmax: BHN^2
- Values: BND

- QK^T need to store both Q and K: 2BND
- Attention matrix stored for softmax op: BHN^2

BND *Note: only need to store their shared input!*

BND

Llama 1 65B		
D	Model dimension	8192
N	Sequence Length	2048
B	Batch size	1
L	Number of layers	80
H	Number of attention heads	64

Total:

$L(16BND + 2BHN^2)$ parameters

$LB(16ND + 2HN^2)*4$ bytes

For batch size 1 for Llama 65B:

~250GB

GPU Memory footprint for Llama 65B @32float

- Inference time:

  32 (bits)*65*10^9(params) = 2.08*10^{12}(bits)/8 = 260GB
- Training time
  - Model parameters = 260GB (same at inference time)
  - Gradients! 4*65*10^9 = 260GB
  - Optimiser: 2*4*65*10^9 (Adam coefficients momentum + variance) = 520GB
  - Activations for backward assuming batch size of 1 = 250GB
- Activations for backward pass are strongly dependent on sequence length and batch size
- Model params + Gradients + optimizer + activations = 1290GB
- 25 RTX 6000s ! – just for a batch size of 1….

Floating point operations (FLOPs)

- Basic unit of computation
- e.g. addition, subtraction, multiplication, divide of 2 floating point numbers

Examples

Given $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^n$

- $\boldsymbol{w} + \boldsymbol{x}$:

  n FLOPs

- $\boldsymbol{w}^T \boldsymbol{x}$:

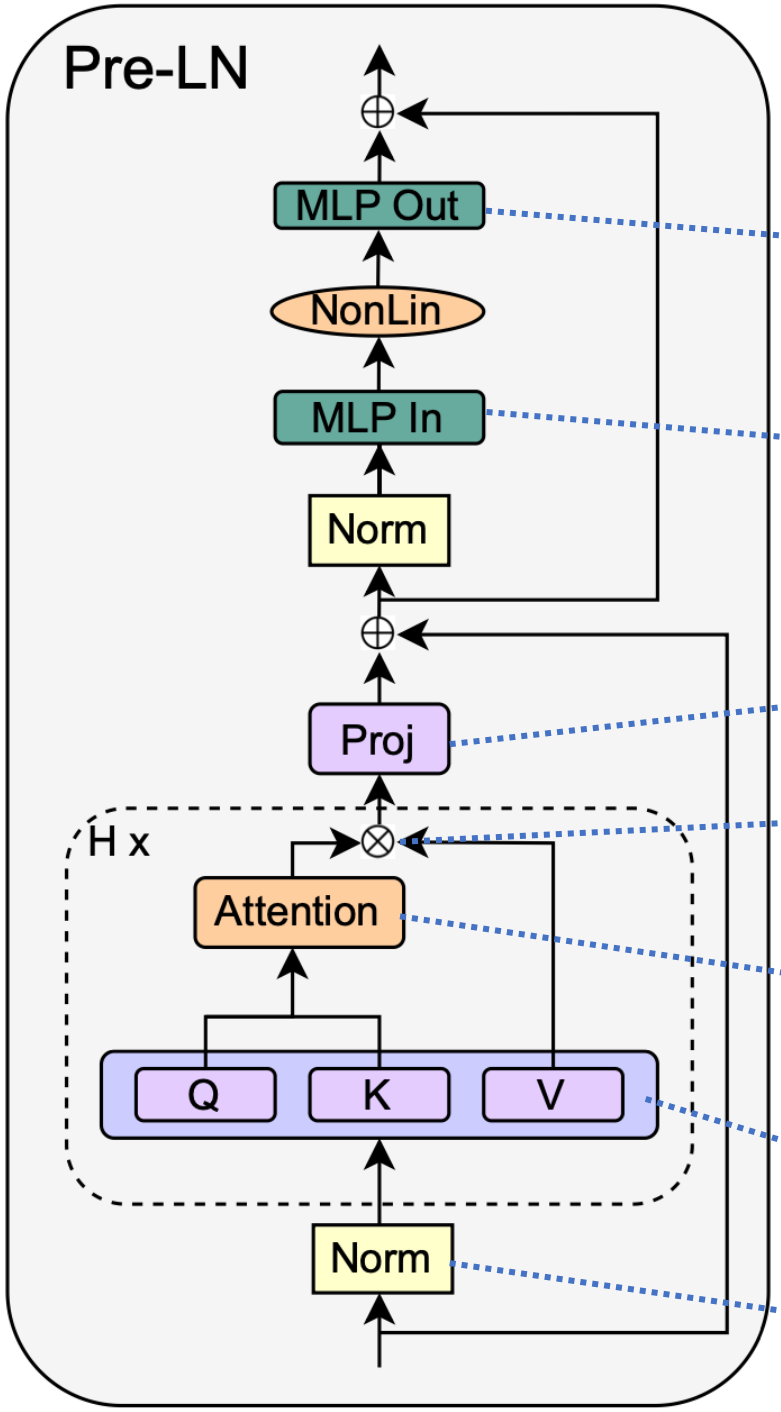
  2n FLOPs (multiplication + (n-1) addition)

Given $\boldsymbol{W} \in \mathbb{R}^{m \times p}$, $\boldsymbol{X} \in \mathbb{R}^{p \times n}$

- $\boldsymbol{W}\boldsymbol{X}$:

  2nmp

Llama FLOPs



Llama 1 65B		
D	Model dimension	8192
N	Sequence Length	2048
B	Batch size	1
L	Number of layers	80
H	Number of attention heads	64

MLP Out: $2*N4DD$

MLP In: $2*ND4D$

Proj: $2*ND^2$

Softmax@V: $2*N^2D$

Attention:
- QK^T : $2*N^2D$
- Softmax: $3*H*N^2$

Q, K, V: $2*3ND^2$ *Note: equivalent to $2*3*H*D*(D/H)$*

Norm: ~20ND *Note: we will drop this as minor compared to following steps*

Total for the forward pass:

$BL(24ND^2 + 4N^2D + 3HN^2)$

$\sim 2.7*10^{14}$

Total for the backward pass:

2*forward pass

Total for batch size 1:

$8*10^{14}$ FLOPs

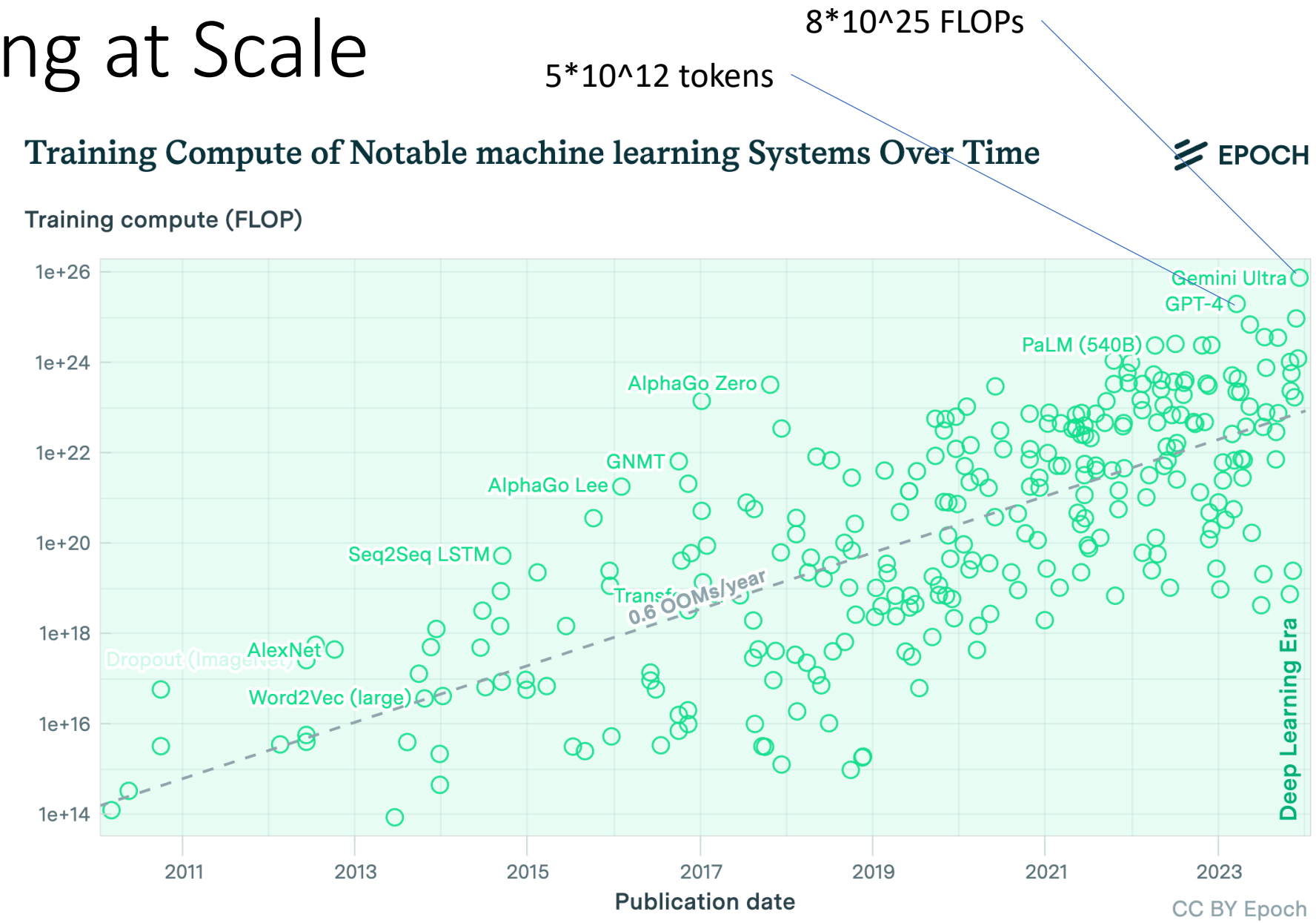
Total training flops ≈

6*dataset size in tokens*number of parameters

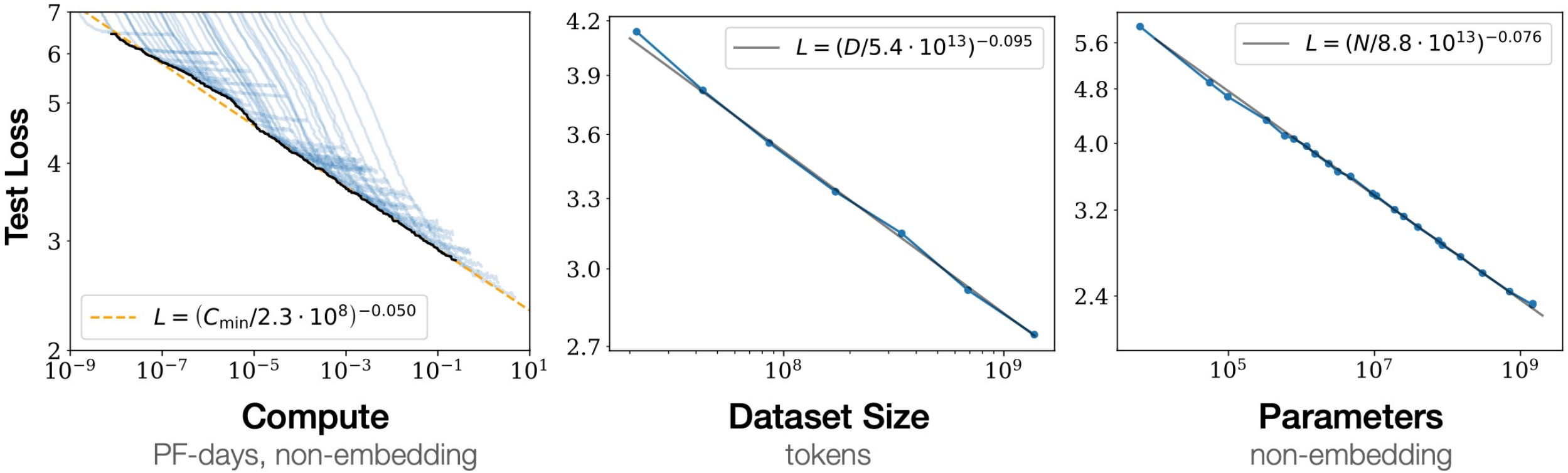
Hoffman et al. Training Compute-Optimal Large Language Models 2022

Deep Learning at Scale

- What 3 main things to we need for Deep Learning?
- Model + Data + Compute!
- Here the task is next token prediction (this token can represent text or image/video/audio patches).
- As we scale these ingredients how much better at the task do we get?
- How to allocate our precious £ to each of the 3 ingredients?



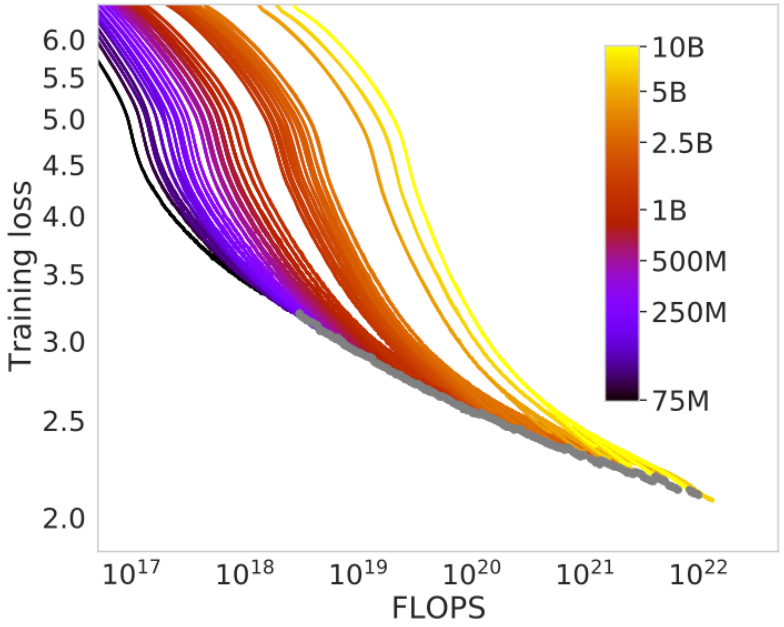
https://epochai.org/data/epochdb

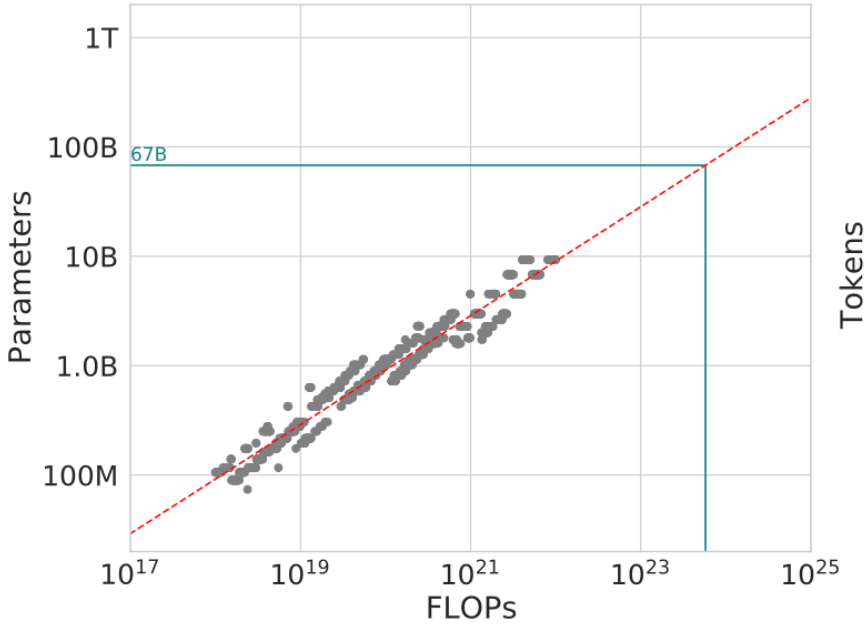


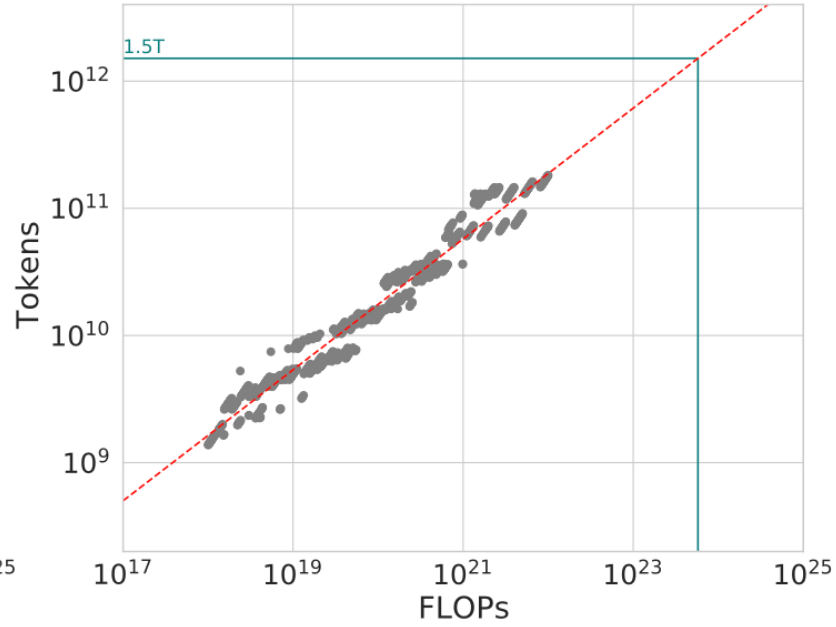
Note1: PF-day = Peta FLOP days = 10^15 * 24 * 3600
Note2: A100 @FP32 19*10^12 FLOPs (per second)

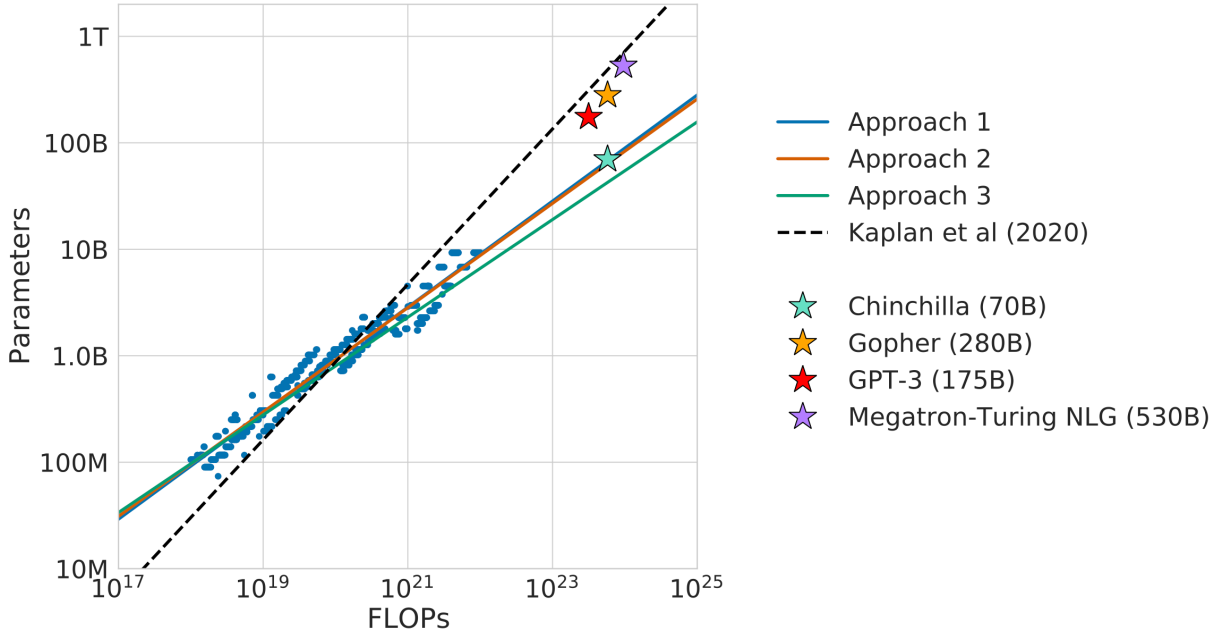
Kaplan et al. Scaling Laws for Neural Language Models 2020

https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

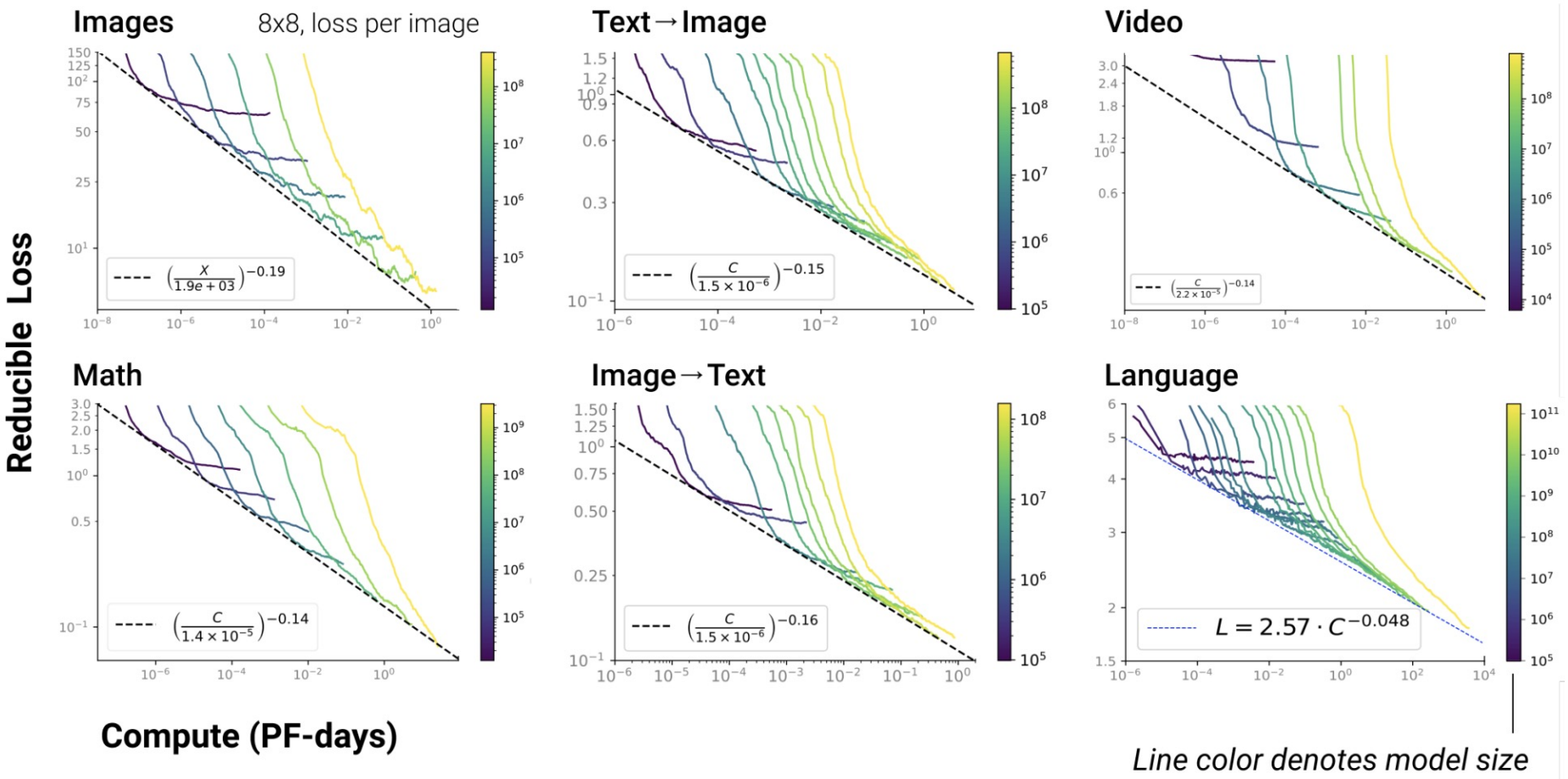








Hoffmann et al. Training Compute-Optimal Large Language Models 2022



Reducible Loss = Loss – Irreducible Loss

Irreducible loss can be viewed as the entropy of the true data distribution
Reducible loss is an estimate of the KL divergence between the data and model distribution

Henighan et al. Scaling Laws for Autoregressive Generative Modeling 2020

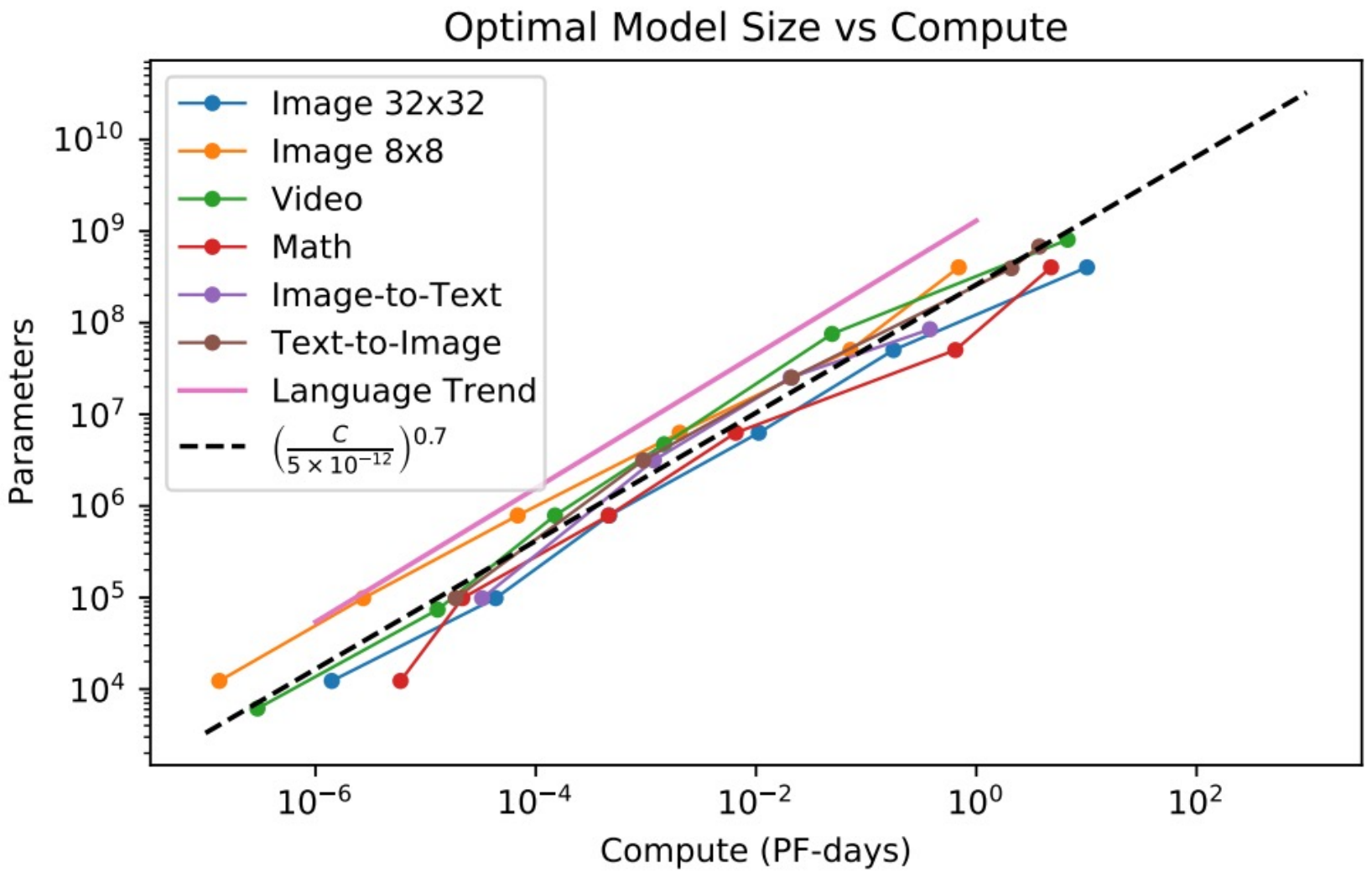
SORA

Base compute

4*compute

16*compute

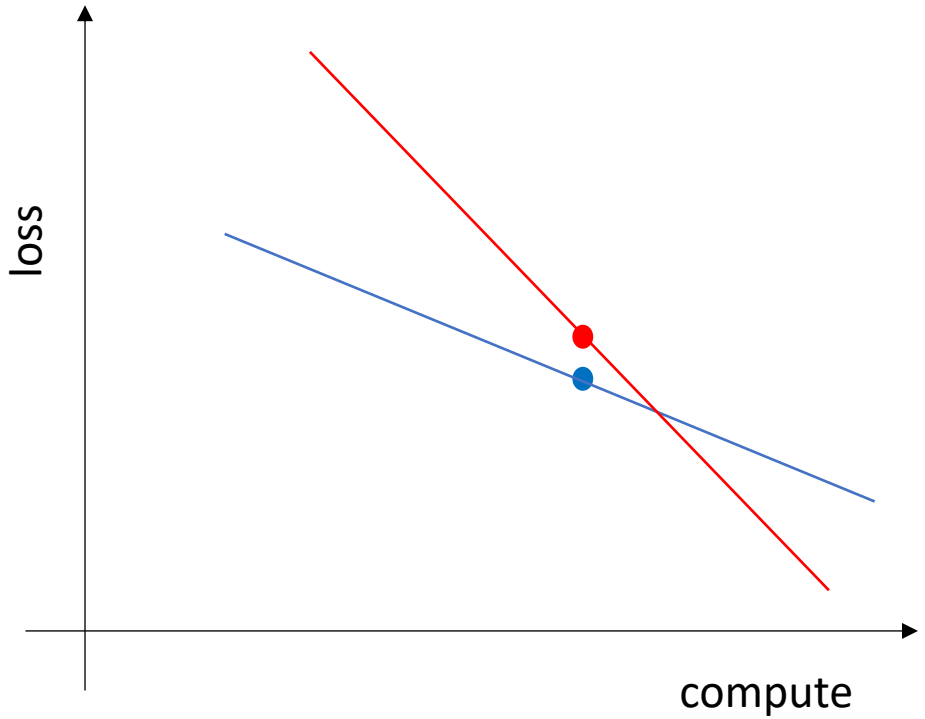
https://openai.com/research/video-generation-models-as-world-simulators



Henighan et al. Scaling Laws for Autoregressive Generative Modeling 2020

Take homes

- Scanning over dataset and model size (2N) compute
  - Bottle necks
  - Improvement gain?
- No barriers to limitless improvement?
- More than 1 epoch?
  - [Hoffman et al. Training Compute-Optimal Large Language Models 2022]

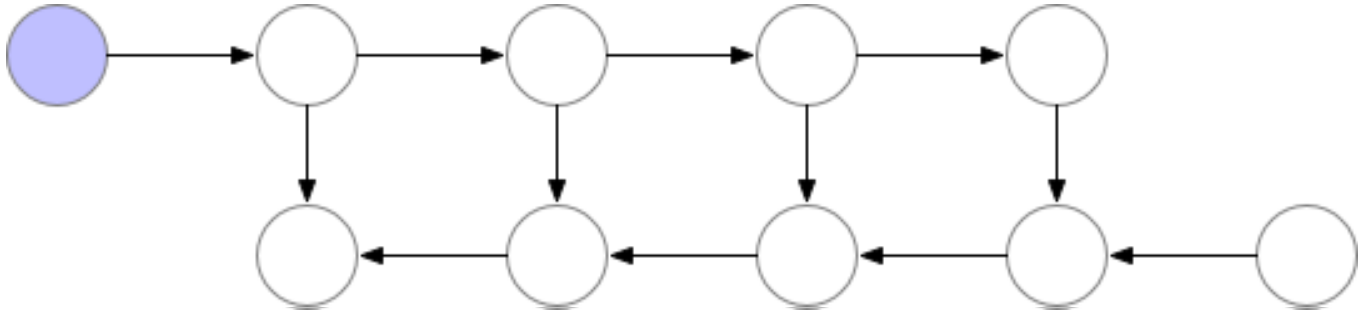


How to get the most out of your resources Part 1: Gradient Accumulation

- Limitations of a very small batch size?
  - Significant noise in parameter updates
  - Many parameter updates
  - Cannot replicate lab’s work
- Iteratively compute forward + backward passes and sum up gradients.
- Only run the optimiser step once (your minibatch * number gradient accumulation steps = target batch)

How to get the most out of your resources Part 1: Gradient Checkpointing

- Recall: The activations for a batch size 1 for Llama 65B is 172GB



Memory: O(1)
Compute: $O(L^2)$
#forwardper layer: 1-L

- We could forget each activation during the forward pass and then recompute them during the backward pass
  - This adds significant more compute
- Compromise! Strategically select activations throughout computational graph, so only a fraction of the graphs need to be recomputed.

Chen et al. Training Deep Nets with Sublinear Memory Cost 2016
https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

Next week

- Finetuning at Scale
- Prompt Engineering
- Foundation model Evals!

- Note on examinable content