Lecture 18: Scaling

Administrative: Last Deadlines

- A5 due Friday (3/13)
- Multiple W Credit assignments due next Monday (3/16)

- **Absolutely no submissions accepted for anything after 11:59 PM on Monday, March 16th**

Administrative: Project

- Poster session during final exam slot (3/16, 10:30-12:20)
  - Allen Center Atrium
  - We will provide easels
  - Guests/friends/collaborators OK to attend

- Two options:
  - Print poster yourself & bring it on Monday
  - Upload poster to Google Form by 9 AM **this Friday** & TAs will bring print-out to poster session

Administrative: Project

- Must **upload** PDF of poster to Gradescope
  - Please make 1 submission per group

- Failure to do this will result in 0% grade for poster

Summary of last few lectures:

More is More!

In language

BERT (Large)
340 million params

GPT - 3
175 billion params

In language

BERT
3.3 Billion tokens[1]
- **All of english wikipedia**
- **11,000 Books**

GPT - 3
~300 billion tokens
- **Common Crawl (Much of the internet)**

[1]https://aclanthology.org/W19-4828.pdf

In language

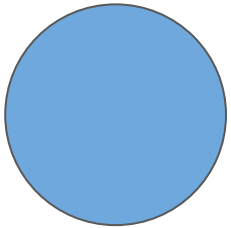
	SuperGLUE Average	BoolQ Accuracy	CB Accuracy	CB F1	COPA Accuracy	RTE Accuracy
Fine-tuned SOTA	**89.0**	**91.0**	**96.9**	**93.9**	**94.8**	**92.5**
Fine-tuned BERT-Large	69.0	77.4	83.6	75.7	70.6	71.7
GPT-3 Few-Shot	71.8	76.4	75.6	52.0	92.0	69.0

	WiC Accuracy	WSC Accuracy	MultiRC Accuracy	MultiRC F1a	ReCoRD Accuracy	ReCoRD F1
Fine-tuned SOTA	**76.1**	**93.8**	**62.3**	**88.2**	**92.5**	**93.3**
Fine-tuned BERT-Large	69.6	64.6	24.1	70.0	71.3	72.0
GPT-3 Few-Shot	49.4	80.1	30.5	75.4	90.2	91.1

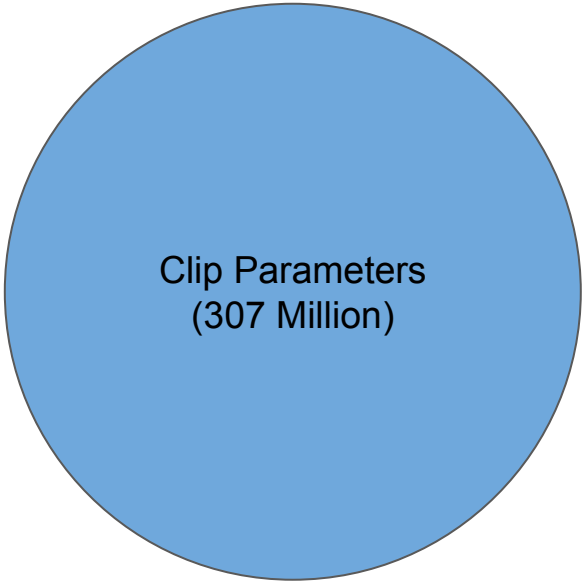
Table 3.8: Performance of GPT-3 on SuperGLUE compared to fine-tuned baselines and SOTA. All results are reported on the test set. GPT-3 few-shot is given a total of 32 examples within the context of each task and performs no gradient updates.

Image Source: Language Models are Few-Shot Learners, Brown et al

In vision!

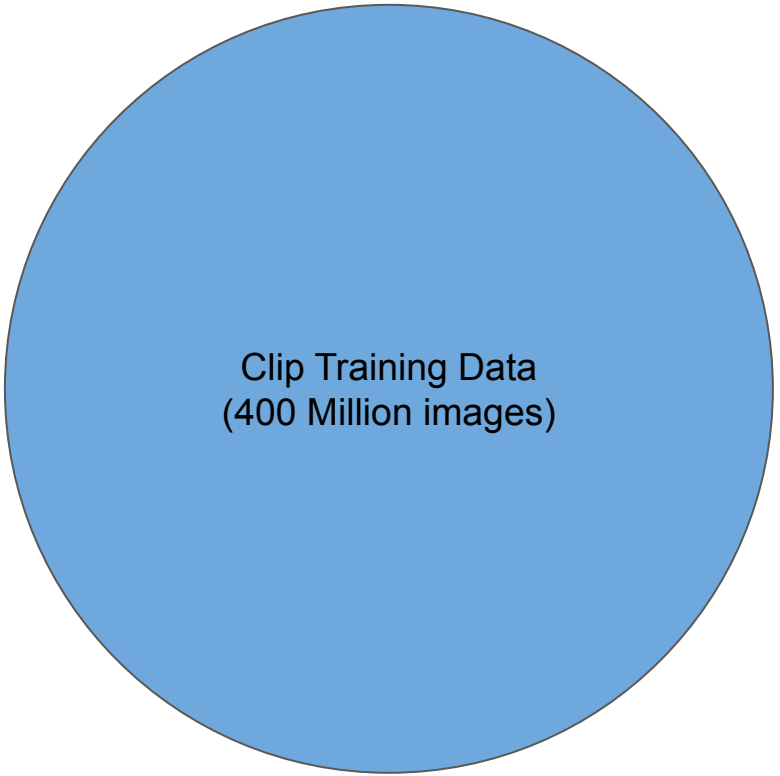


ImageNet ResNet Parameters
(44.5 Million)



In vision!

ImageNet ResNet Training Data
(1.28 Million)



In vision!

DATASET	IMAGENET RESNET101	CLIP VIT-L
 ImageNet	76.2%	76.2%

The Bitter Lesson

Rich Sutton

March 13, 2019

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. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

http://www.incompleteideas.net/IncIdeas/BitterLesson.html

Scaling Laws

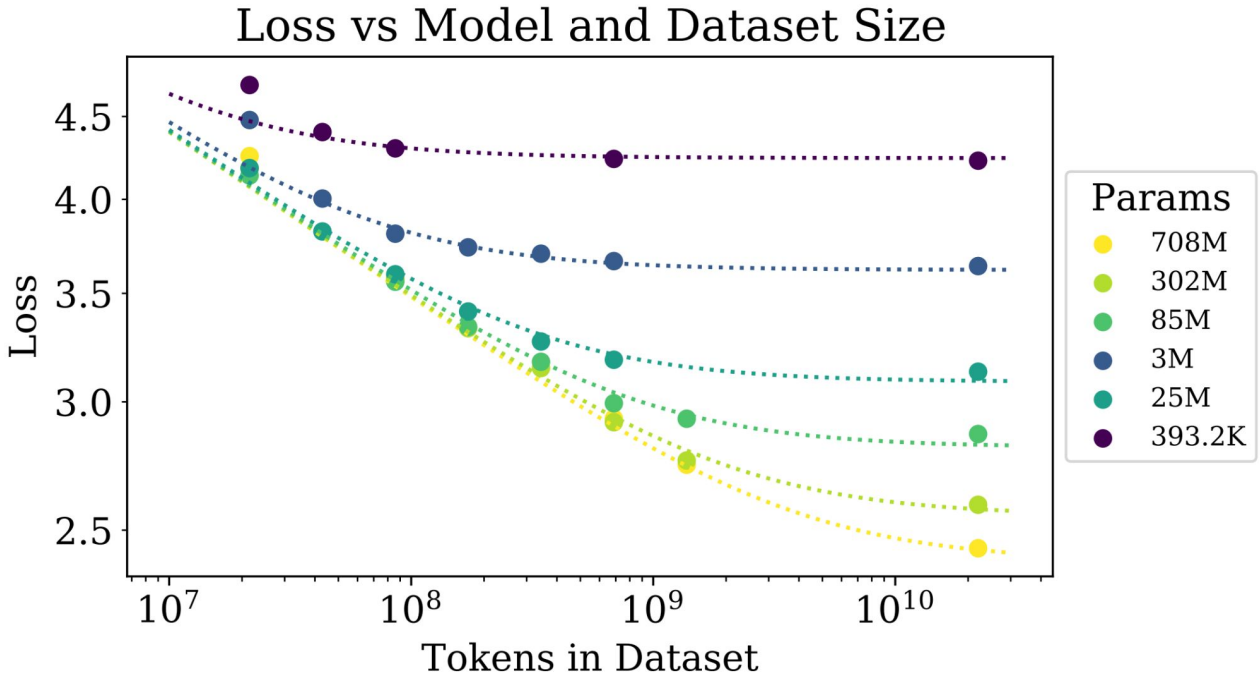


Image source: [Kaplan et al. Scaling Laws for Neural Language Models. 2020.]

Question: If scale reliably improves models, why don’t we have agi yet??

Part 1: The Difficulty of Scaling

Part 2: Parallelism

Part 3: Scaling Smarter

Part 4: Research on a Budget

Part 1: The Difficulty of Scaling

- Deepdive into the numbers and hardware of training large models

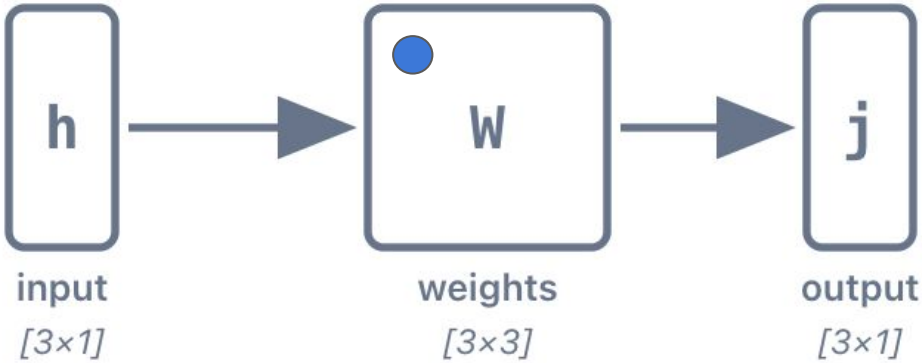
What does it take to train a model? Compute!

How much compute? (how to measure?)

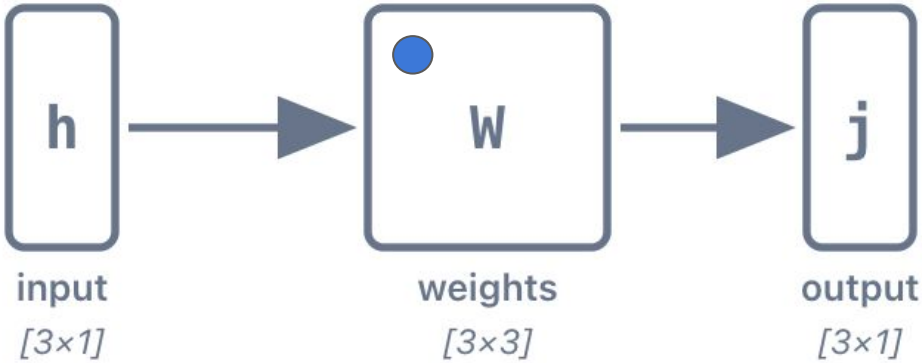
How many FLOPs?

~6PD **Why this?**

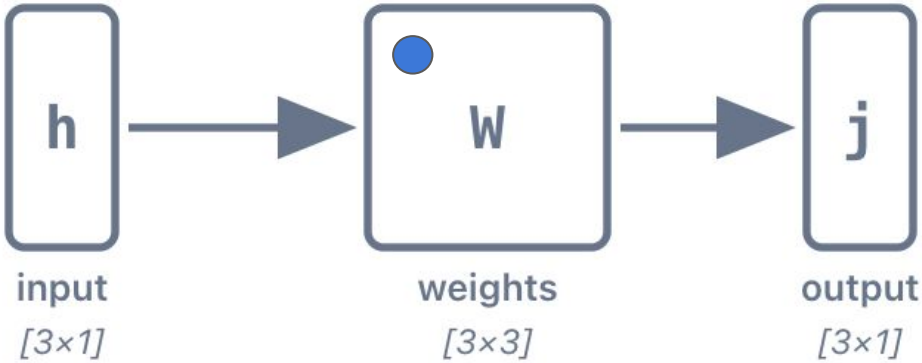
P = number of params in the model
D = number of training tokens



Let's count the number of flops for 1 parameter, for 1 element in batch.

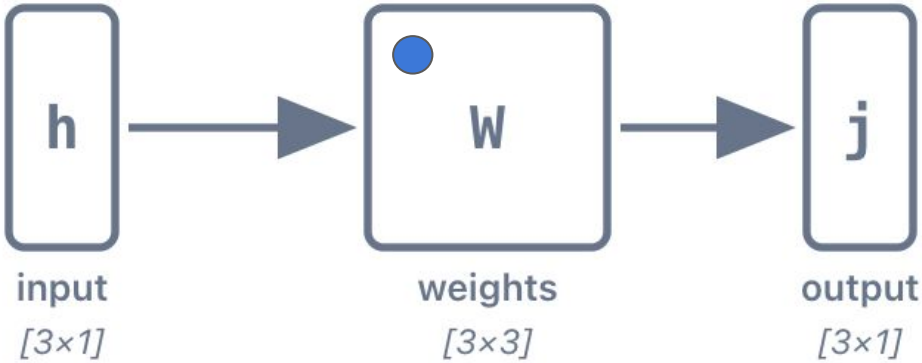


FLOP count: 0



FLOP count: 0

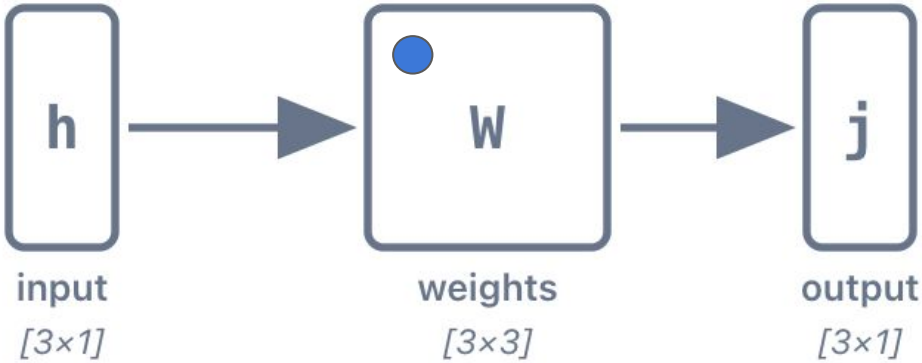
Forward Pass



FLOP count: 0

Forward Pass

Backward Pass

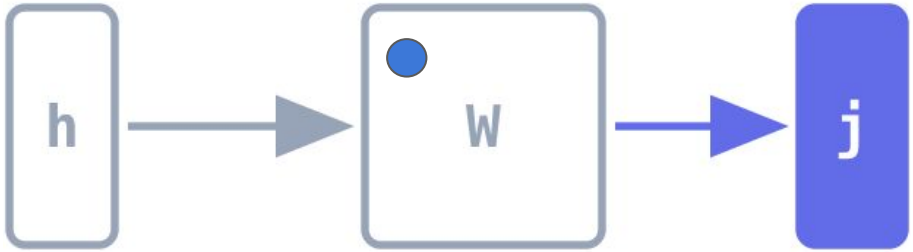


FLOP count: 0

Forward Pass

Backward Pass

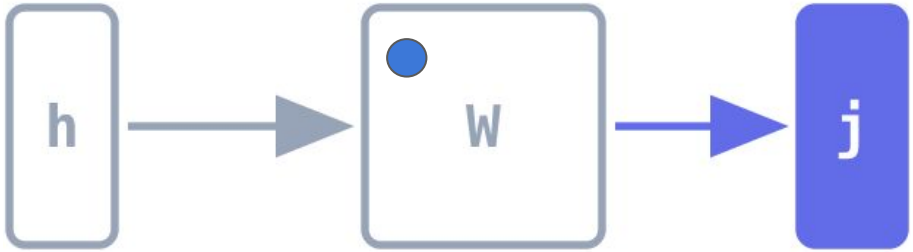
- **dL/dw**
- **dL/dh**



FLOP count: 0

$$\begin{bmatrix} W_{00} & W_{01} & W_{02} \\ W_{10} & W_{11} & W_{12} \\ W_{20} & W_{21} & W_{22} \end{bmatrix} \times \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} h_0W_{00} + h_1W_{01} + h_2W_{02} \\ h_0W_{10} + h_1W_{11} + h_2W_{12} \\ h_0W_{20} + h_1W_{21} + h_2W_{22} \end{bmatrix}$$

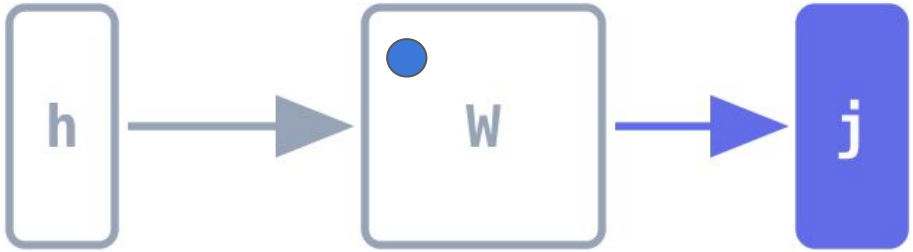
W h j



FLOP count: 1

$$\begin{bmatrix} W_{00} & W_{01} & W_{02} \\ W_{10} & W_{11} & W_{12} \\ W_{20} & W_{21} & W_{22} \end{bmatrix} \times \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} h_0W_{00} + h_1W_{01} + h_2W_{02} \\ h_0W_{10} + h_1W_{11} + h_2W_{12} \\ h_0W_{20} + h_1W_{21} + h_2W_{22} \end{bmatrix}$$

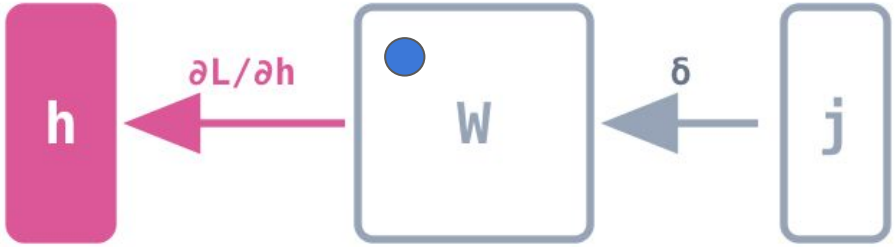
W h j



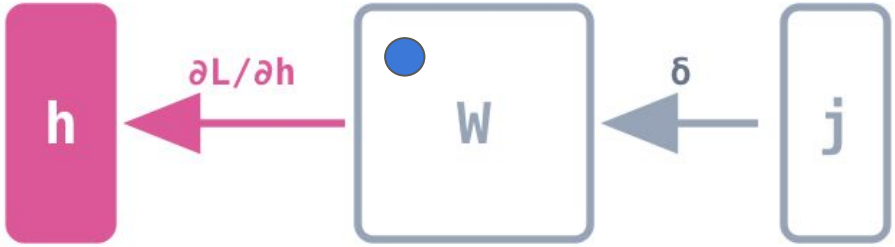
FLOP count: 2

$$\begin{bmatrix} W_{00} & W_{01} & W_{02} \\ W_{10} & W_{11} & W_{12} \\ W_{20} & W_{21} & W_{22} \end{bmatrix} \times \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} h_0W_0 + h_1W_{01} + h_2W_{02} \\ h_0W_{10} + h_1W_{11} + h_2W_{12} \\ h_0W_{20} + h_1W_{21} + h_2W_{22} \end{bmatrix}$$

W h j



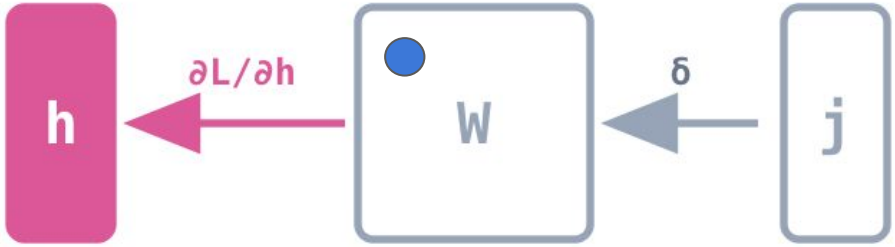
FLOP count: 2



FLOP count: 2

$$\begin{bmatrix} W_{00} & W_{10} & W_{20} \\ W_{01} & W_{11} & W_{21} \\ W_{02} & W_{12} & W_{22} \end{bmatrix} \times \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \delta_0 W_{00} + \delta_1 W_{10} + \delta_2 W_{20} \\ \delta_0 W_{01} + \delta_1 W_{11} + \delta_2 W_{21} \\ \delta_0 W_{02} + \delta_1 W_{12} + \delta_2 W_{22} \end{bmatrix}$$

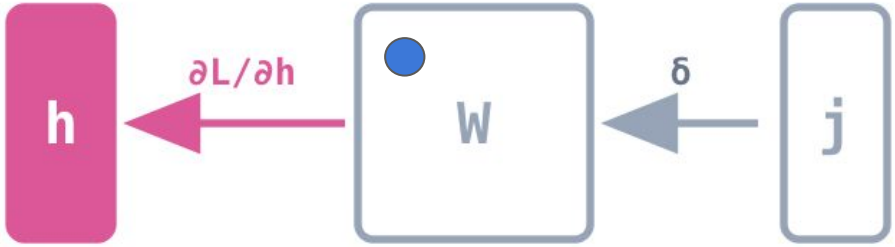
W^T $\quad$ $\delta = \partial L/\partial j$ $\quad$ $\partial L/\partial h$



FLOP count: 3

$$\begin{bmatrix} W_{00} & W_{10} & W_{20} \\ W_{01} & W_{11} & W_{21} \\ W_{02} & W_{12} & W_{22} \end{bmatrix} \times \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \delta_0 W_{00} + \delta_1 W_{10} + \delta_2 W_{20} \\ \delta_0 W_{01} + \delta_1 W_{11} + \delta_2 W_{21} \\ \delta_0 W_{02} + \delta_1 W_{12} + \delta_2 W_{22} \end{bmatrix}$$

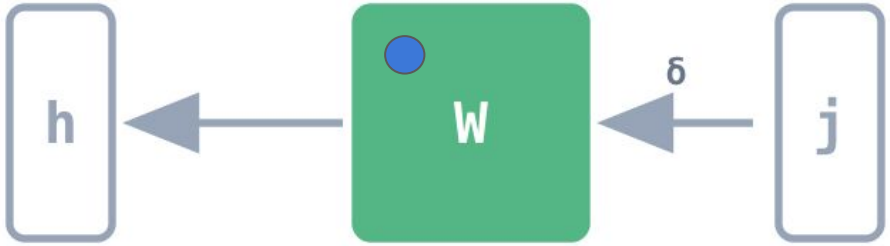
W^T $\quad \delta = \partial L/\partial j \quad \partial L/\partial h$



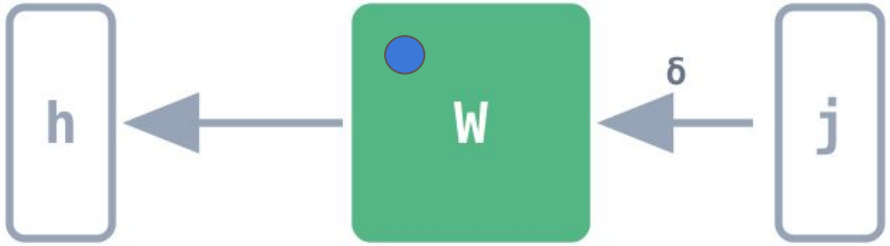
FLOP count: 4

$$\begin{bmatrix} W_{00} & W_{10} & W_{20} \\ W_{01} & W_{11} & W_{21} \\ W_{02} & W_{12} & W_{22} \end{bmatrix} \times \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \delta_0 W_{00} + \delta_1 W_{10} + \delta_2 W_{20} \\ \delta_0 W_{01} + \delta_1 W_{11} + \delta_2 W_{21} \\ \delta_0 W_{02} + \delta_1 W_{12} + \delta_2 W_{22} \end{bmatrix}$$

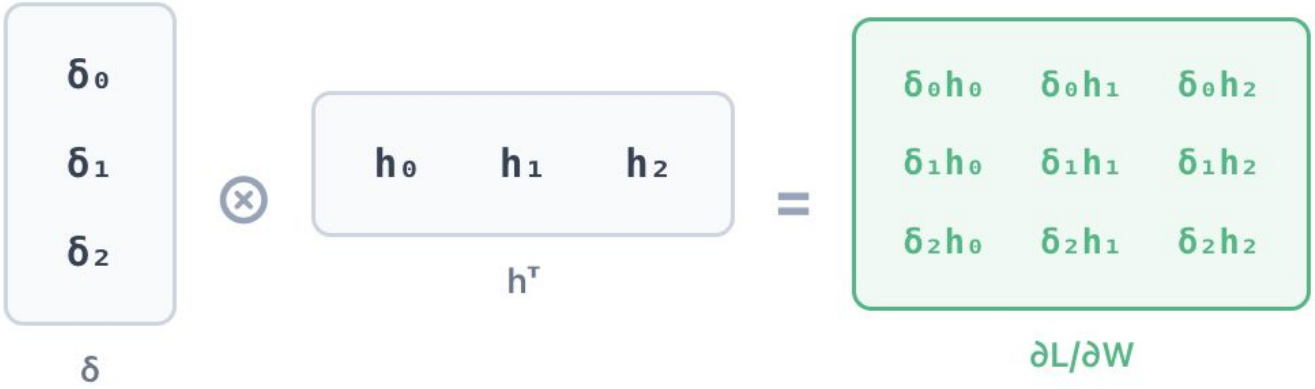
W^T $\quad \delta = \partial L/\partial j \quad \partial L/\partial h$

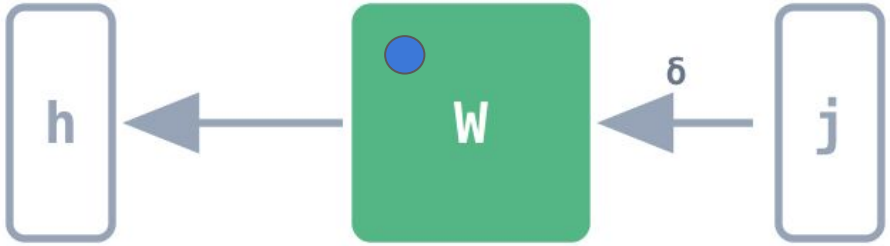


FLOP count: 4

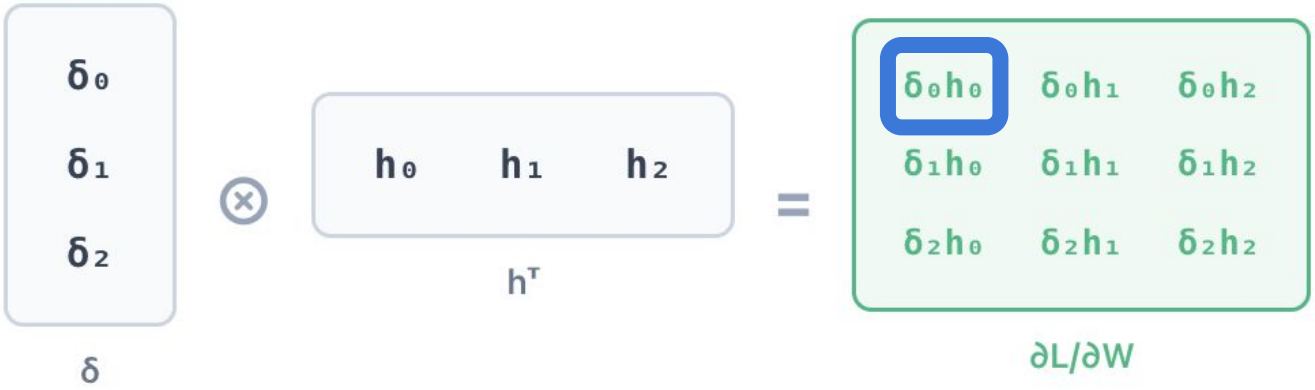

h
W
δ
j

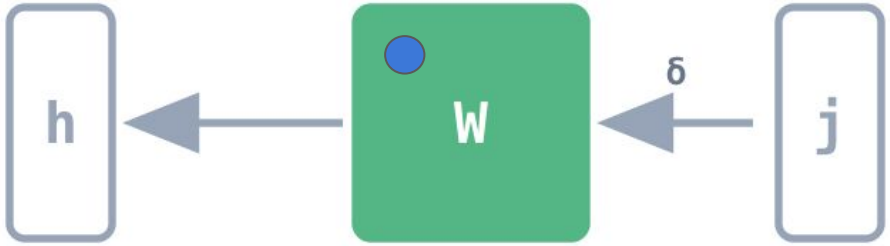
FLOP count: 4


δ₀
δ₁
δ₂
δ
⊗
h₀ h₁ h₂
hᵀ
=
δ₀h₀ δ₀h₁ δ₀h₂
δ₁h₀ δ₁h₁ δ₁h₂
δ₂h₀ δ₂h₁ δ₂h₂
∂L/∂W

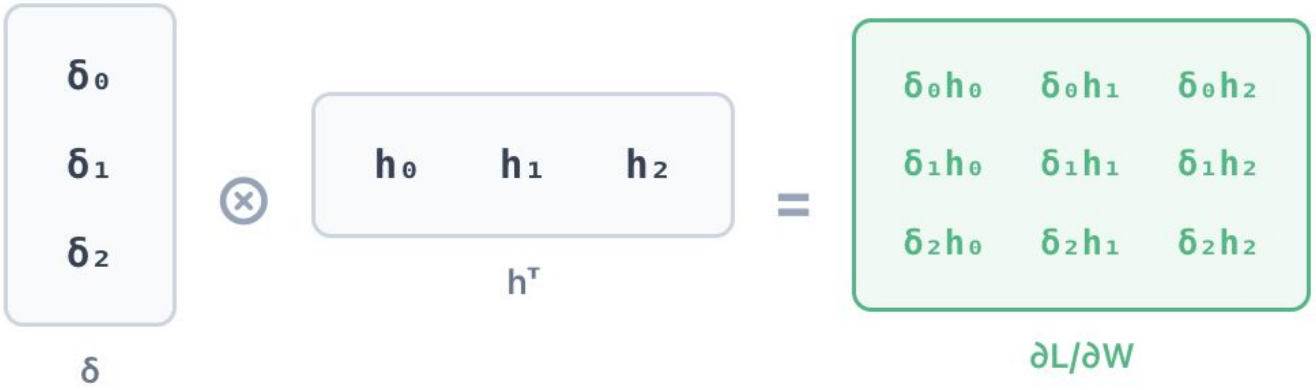

h
W
δ
j

FLOP count: 5


δ₀
δ₁
δ₂
δ
⊗
h₀ h₁ h₂
hᵀ
=
δ₀h₀ δ₀h₁ δ₀h₂
δ₁h₀ δ₁h₁ δ₁h₂
δ₂h₀ δ₂h₁ δ₂h₂
∂L/∂W



FLOP count: 6



Combine this with gradients from other datapoints in batch

Blog post on this topic

Training FLOPs = ~6PD

Training FLOPs = ~6PD

Concrete Example: Llama

P = 65.2 Billion params
D = 1.4 Trillion tokens

Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv:2302.13971, 2023.

Training FLOPs = ~6PD

Concrete Example: Llama

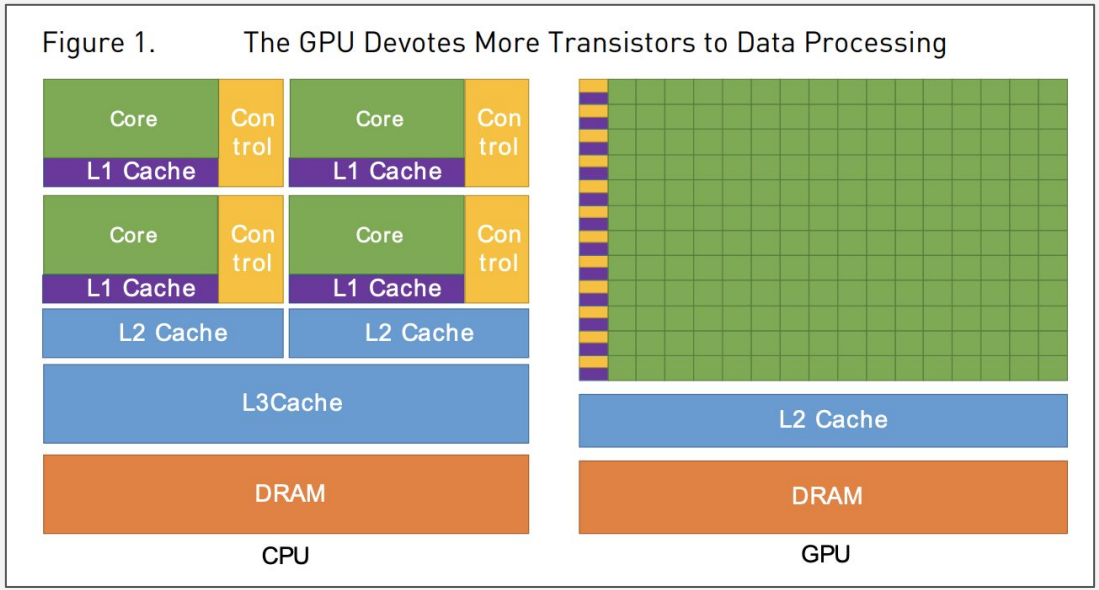
P = 65.2 Billion params
D = 1.4 Trillion tokens

547,680,000,000,000,000,000,000 FLOPs (5.48e23)

How are we going to do this?

Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv:2302.13971, 2023.

What is a GPU?



Central Processing Unit

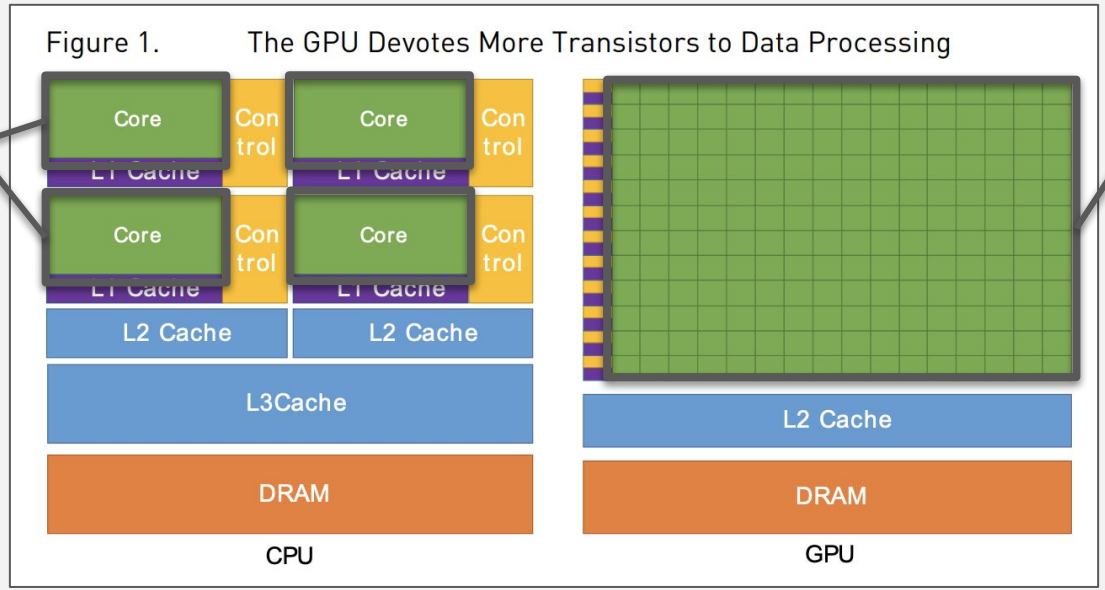
Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

What is a GPU?

Core = Does the actual processing (addition, multiplication)

Fewer cores, but larger and more capable



Many (thousands) of smaller simpler cores

Central Processing Unit

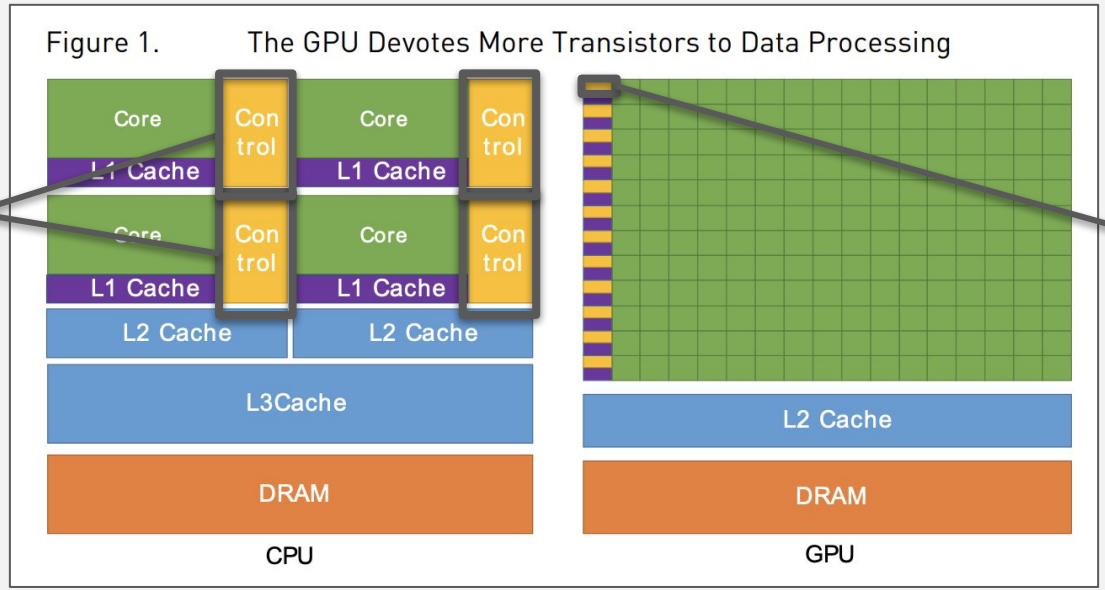
Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

What is a GPU?

Control = “Brain”. Decides what to do in what order. Controls cores.

One control per core allows each core to do an independent complex set of instructions



One control is shared by many cores. Each of these cores much then execute the same instruction

Central Processing Unit

Graphics Processing Unit

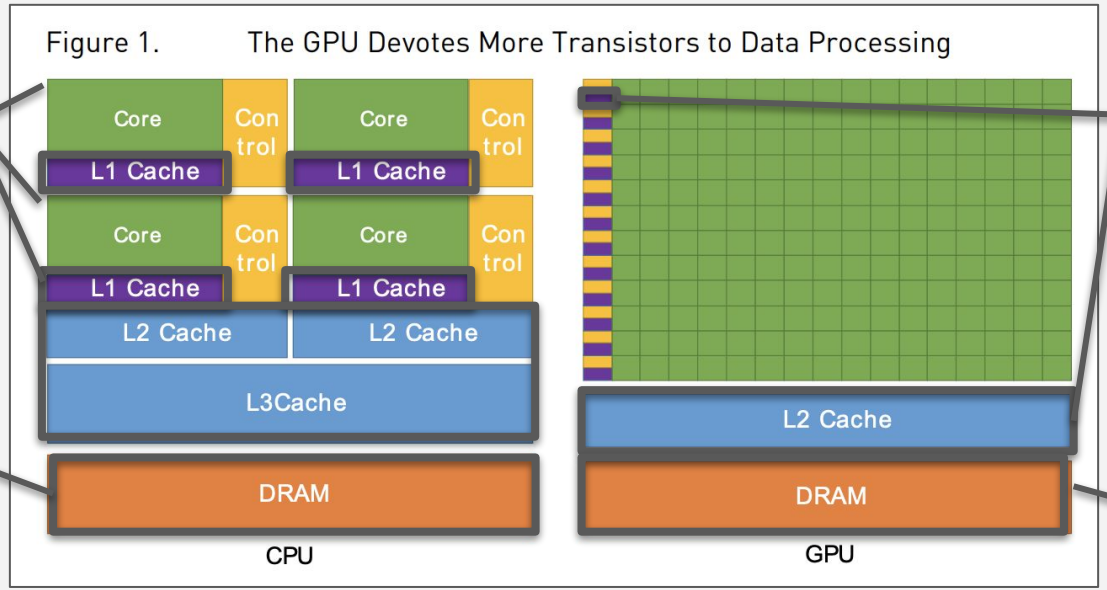
https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

What is a GPU?

Cached memory and DRAM Memory

Lots of quick access memory

Slower memory



Less quick access memory

Slower memory

Central Processing Unit

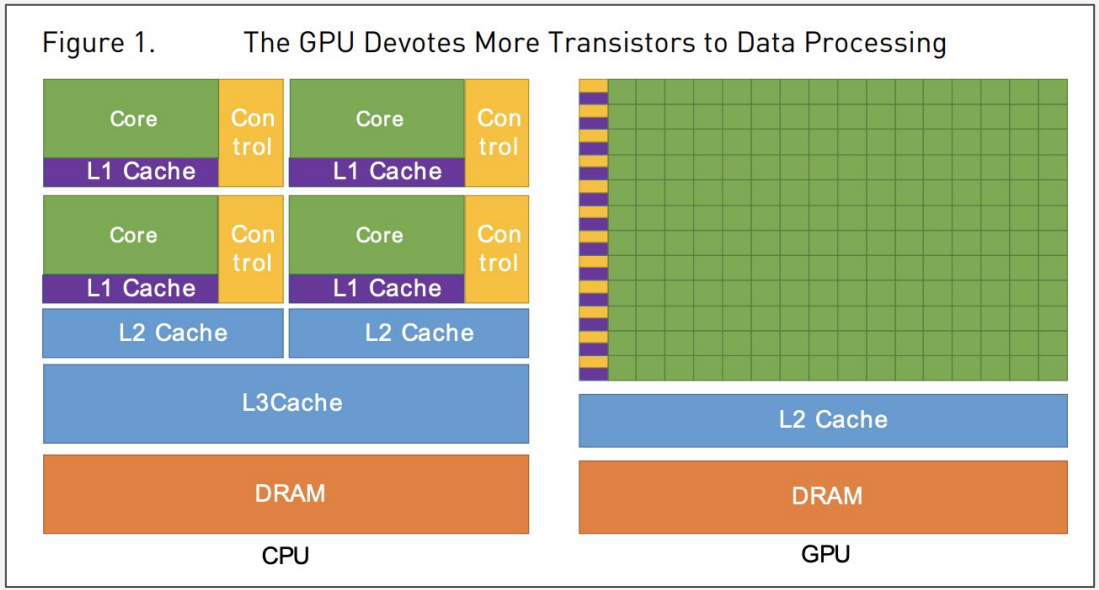
Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

What is a GPU?

Prioritizes Capability

Ideal for running a handful of complex tasks at once



Prioritizes Parallelism

Ideal for running thousand of simple computations at once

Central Processing Unit

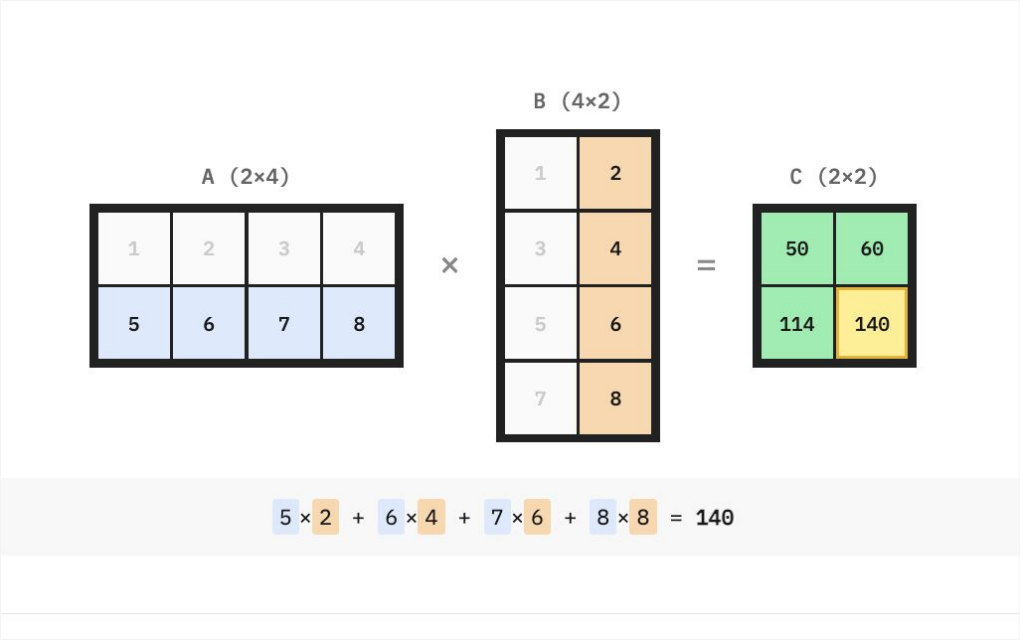
Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

Why do we use GPUs in deep learning?

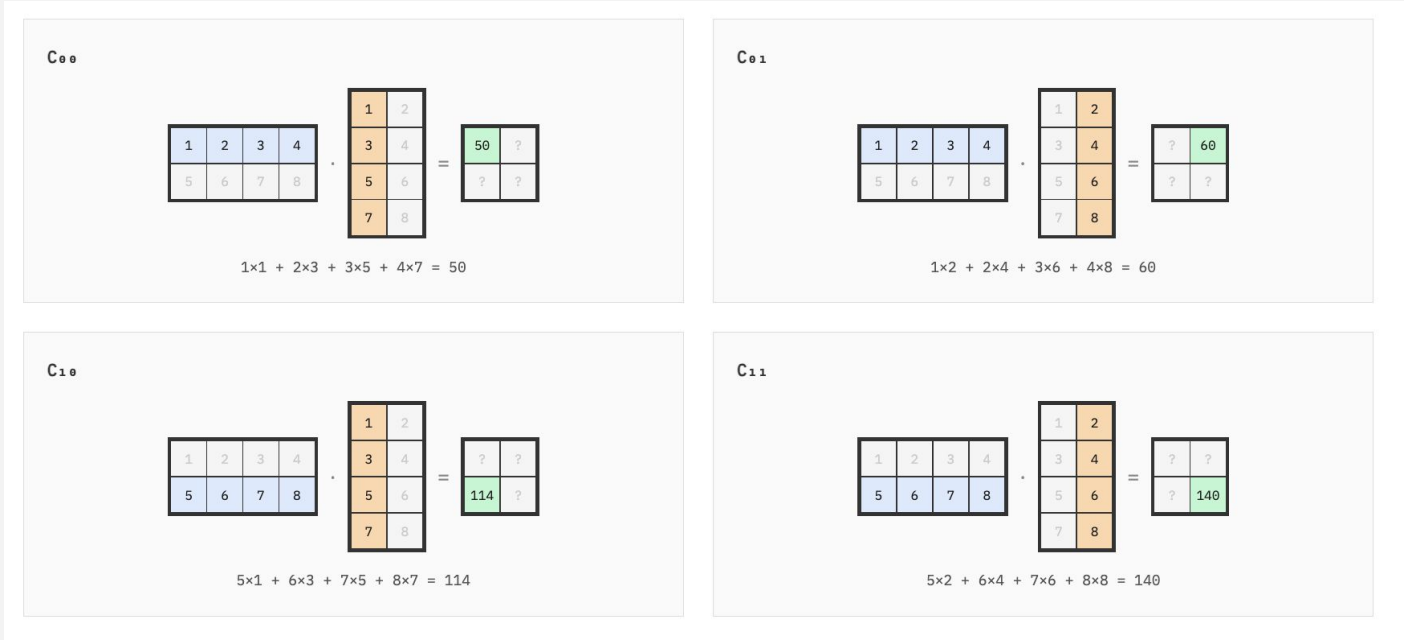
Why do we use GPUs in deep learning?

Deep Learning = Matrix Multiplication



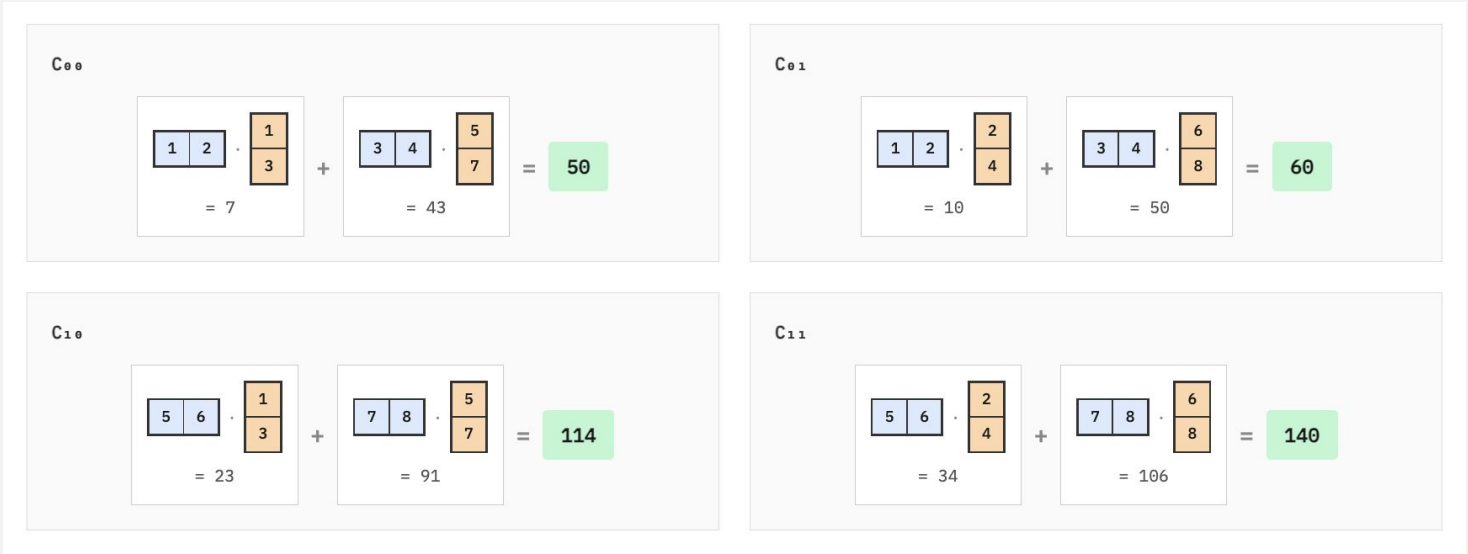
Why do we use GPUs in deep learning?

Deep Learning = Matrix Multiplication



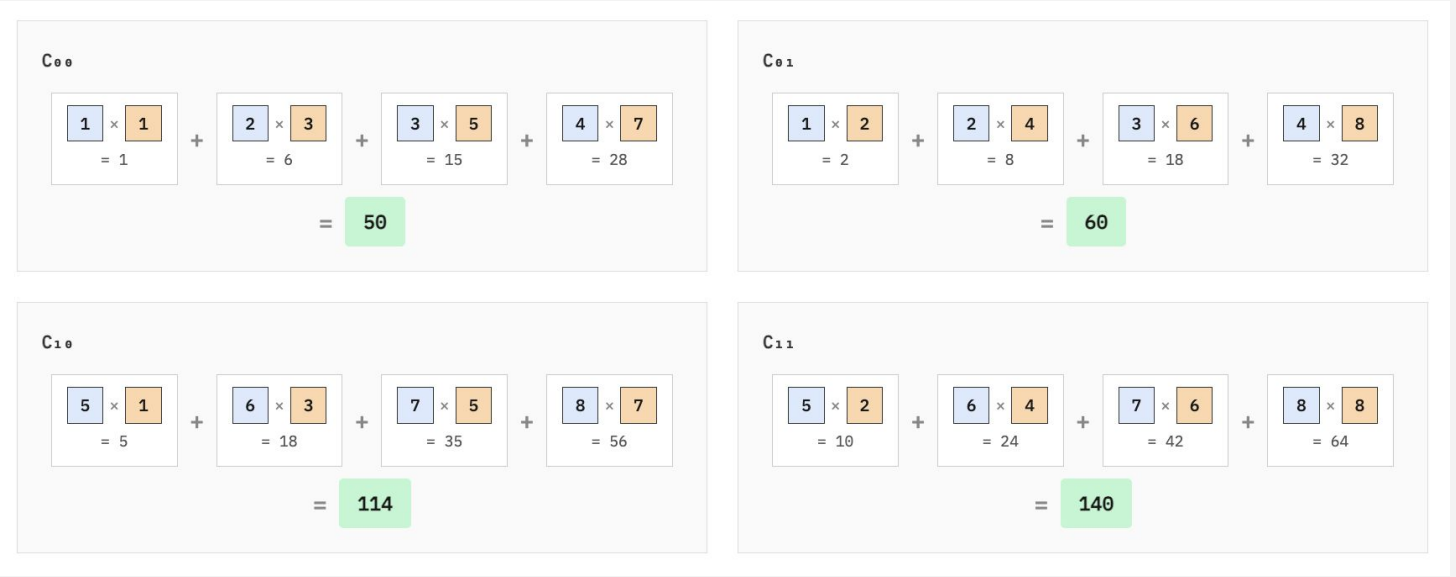
Why do we use GPUs in deep learning?

Deep Learning = Matrix Multiplication



Why do we use GPUs in deep learning?

Deep Learning = Matrix Multiplication



Note on other hardware

Many other things exist besides GPUs!

Main one is TPUs

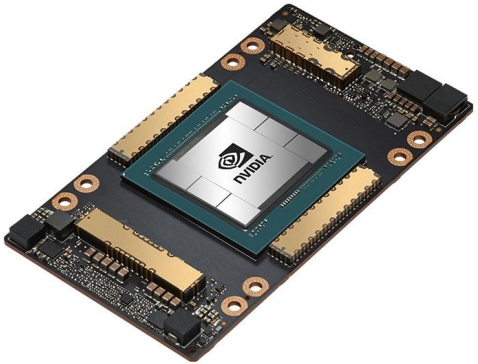
- Even more specialized than GPUs for MM
- Made exclusively by Google, not for sale

Returning to our example…

5.48e23 FLOPs to train Llama

Nvidia A100 stats

Returning to our example…

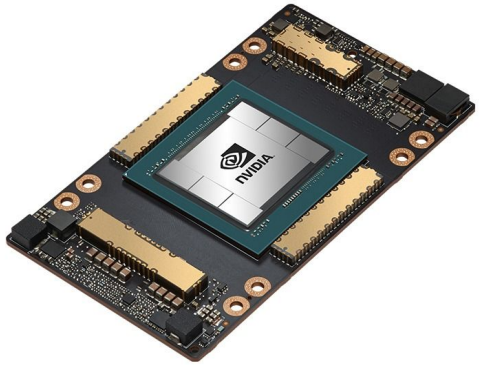


5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Nvidia A100 stats

Returning to our example…



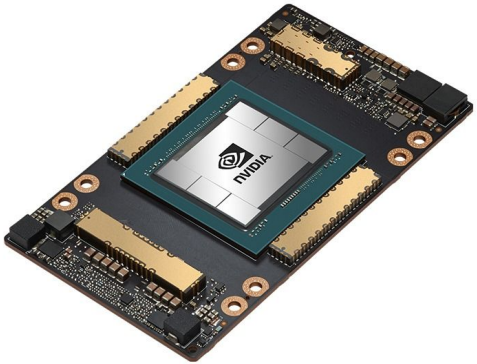
5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Total seconds = 1,370,000,000

Nvidia A100 stats

Returning to our example…



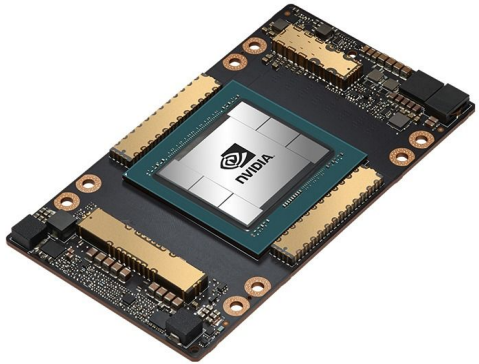
5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Total seconds = 1,370,000,000
Total hours = 380,555

Nvidia A100 stats

Returning to our example…



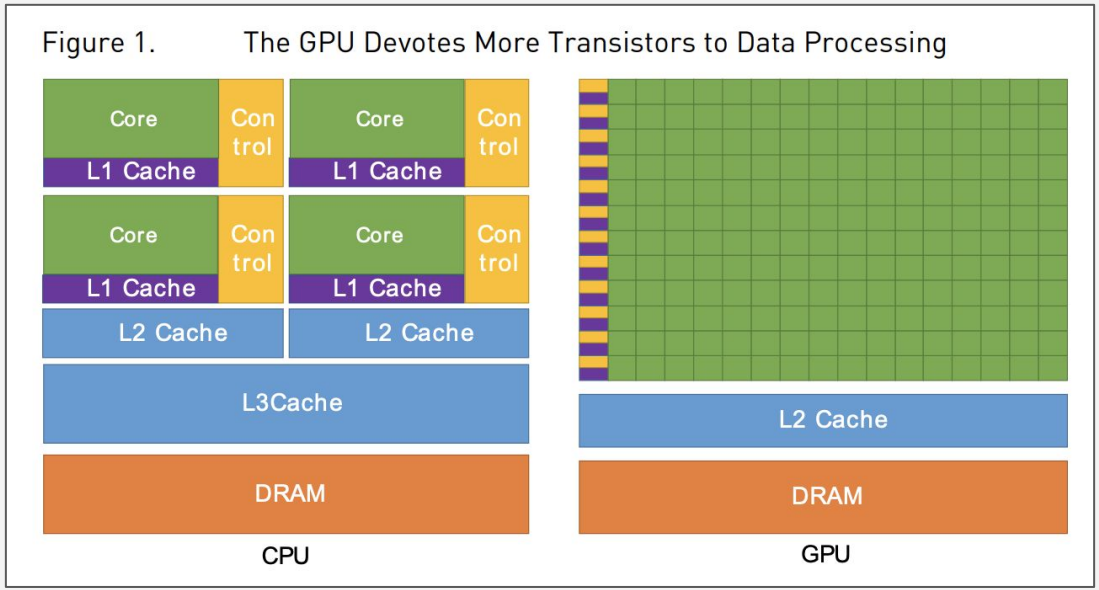
5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Total seconds = 1,370,000,000
Total hours = 380,555
Could finish training in 2 months with 256 A100s

Nvidia A100 stats

What is a GPU?



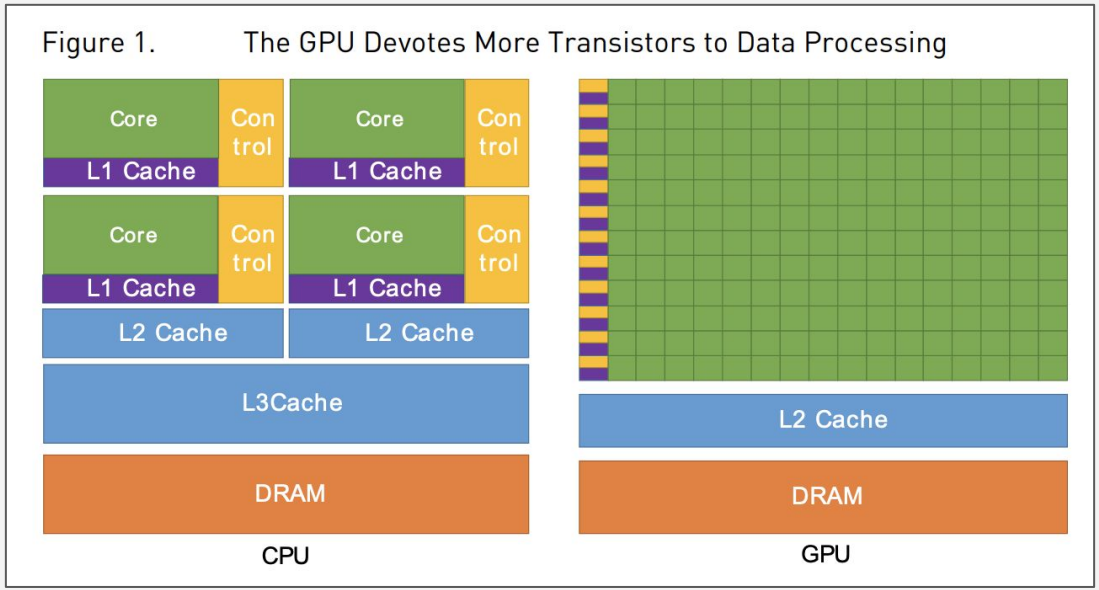
Central Processing Unit

Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

What is a GPU?

$500



$15,000

Central Processing Unit

Graphics Processing Unit

https://docs.nvidia.com/cuda/archive/11.2.0/pdf/CUDA_C_Programming_Guide.pdf

Returning to our example

5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Total seconds = 1,370,000,000
Total hours = 380,555
Could finish training in 2 months with 256 A100s

Nvidia A100 stats

Returning to our example

5.48e23 FLOPs to train Llama

GPU A100 does 300-600 TFLOPs / sec or ~4e14

Total seconds = 1,370,000,000
Total hours = 380,555
Cost $4/hour = **~$1.5 million**

Nvidia A100 stats

Wow scaling is difficult - we need $1.5 million just to train a 65 billion parameter model!

Wow scaling is difficult - we need $1.5 million just to train a 65 billion parameter model!

But.. the compute requirements are only one of our bounds!

Our second bound is **memory**

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Adam First Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Adam First Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
Adam Second Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Adam First Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
Adam Second Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
(sometimes) Adam Copy of params	FP32 = 4*P	4*65 Billion bytes = 260 GB

How much memory?

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Adam First Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
Adam Second Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
(sometimes) Adam Copy of params	FP32 = 4*P	4*65 Billion bytes = 260 GB
		~1 TB

How much memory?

Params + gradients + optimizer states = ~1TB

+ Activations
  - (need to store these to do backward pass)
  - Scale with batch size

What's the problem?

GPU A100 has 80GB, and this is pretty close to the best we have…

Summary: Why scaling is hard

Compute at scale is hard

- As dataset size and model size grows, so does number of FLOPs.
- GPUs can do a lot of FLOPs, but are $$$

Memory at scale is harder

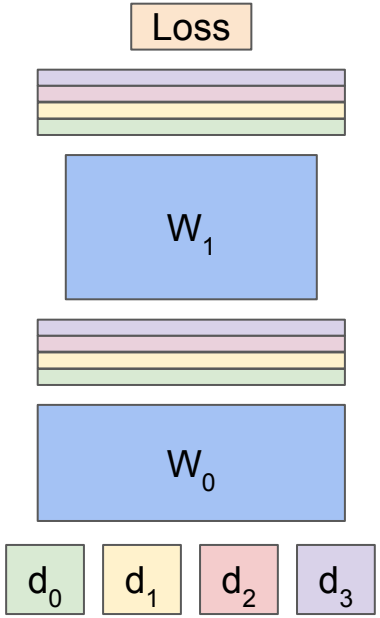
- How do we even use these GPUs if our models don’t fit???

Part 1: The Difficulty of Scaling

Part 2: Parallelism

- How to get a big model on a small GPU

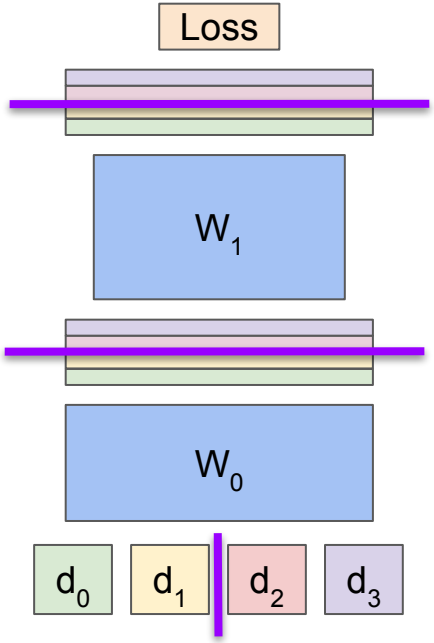
Problem: Model doesn't fit in GPU memory



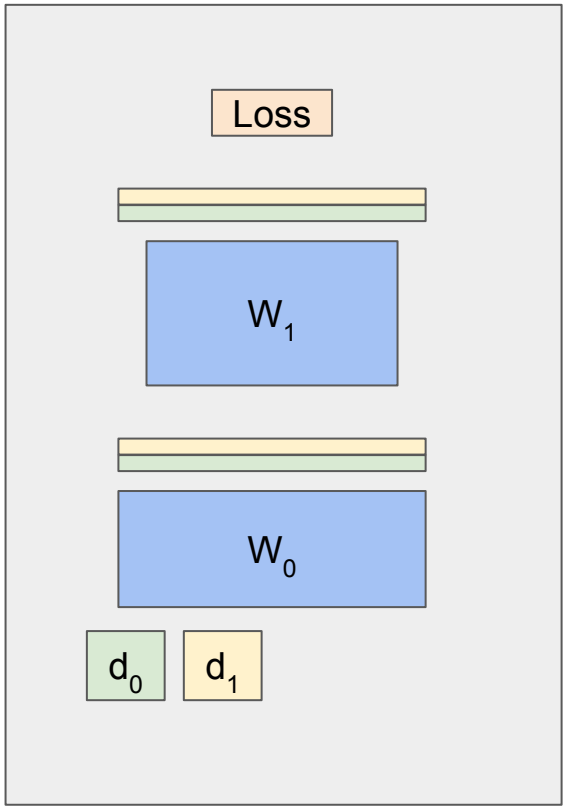
GPU 1

GPU 2

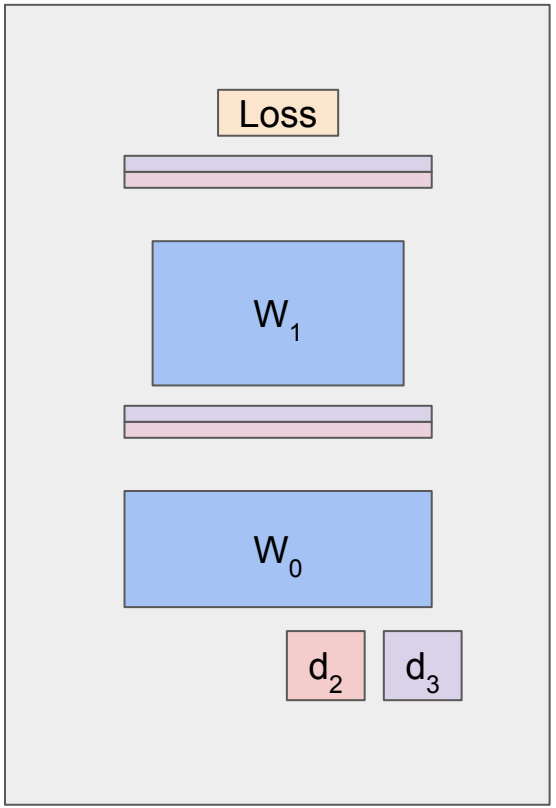
Problem: Model doesn't fit in GPU memory



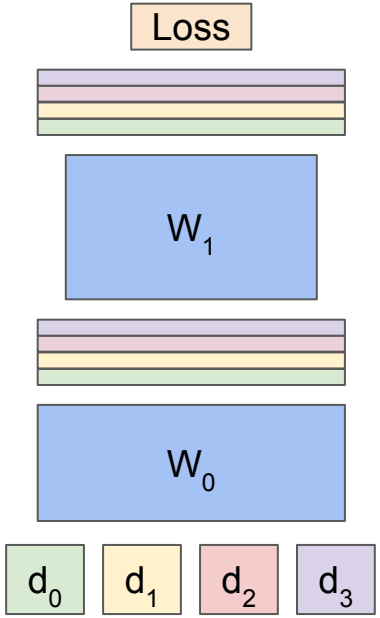
GPU 1



GPU 2



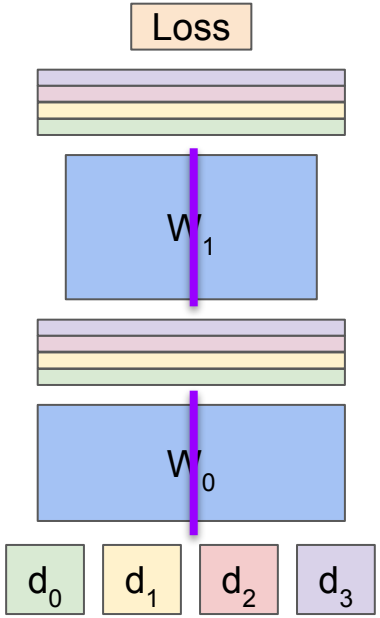
Problem: Model doesn't fit in GPU memory



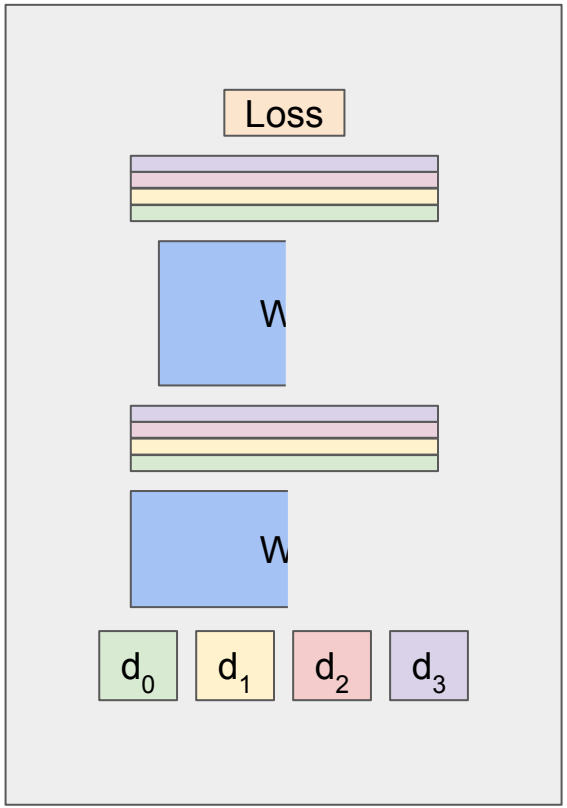
GPU 1

GPU 2

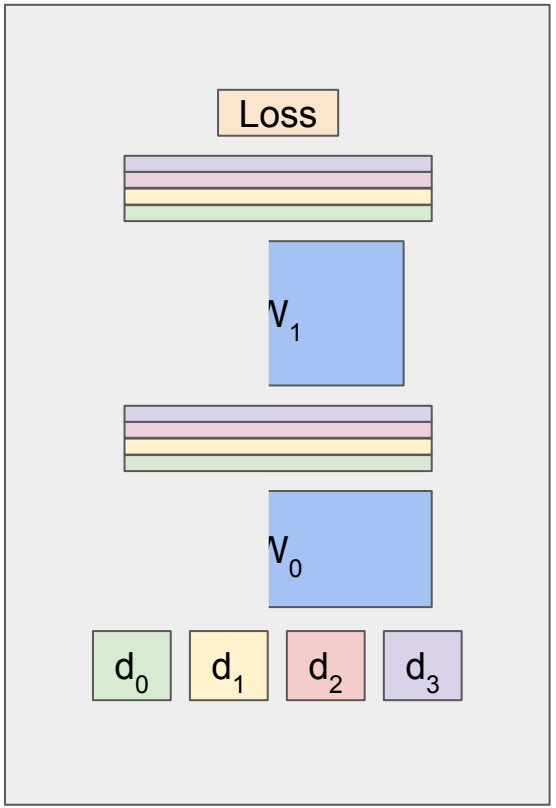
Problem: Model doesn't fit in GPU memory



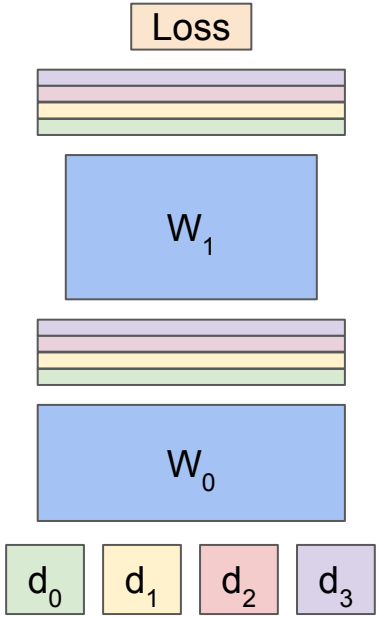
GPU 1



GPU 2



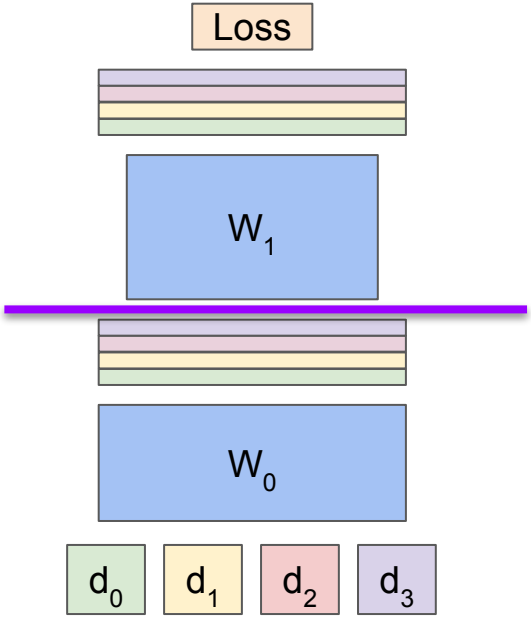
Problem: Model doesn't fit in GPU memory



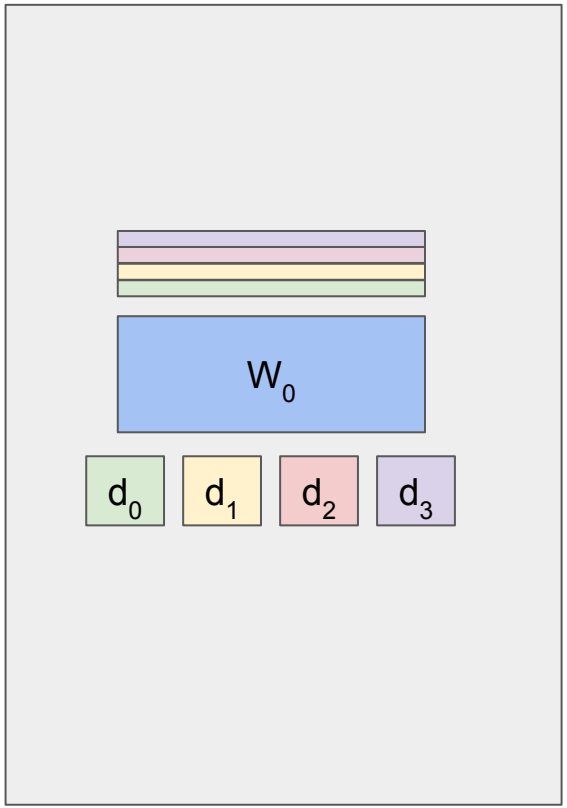
GPU 1

GPU 2

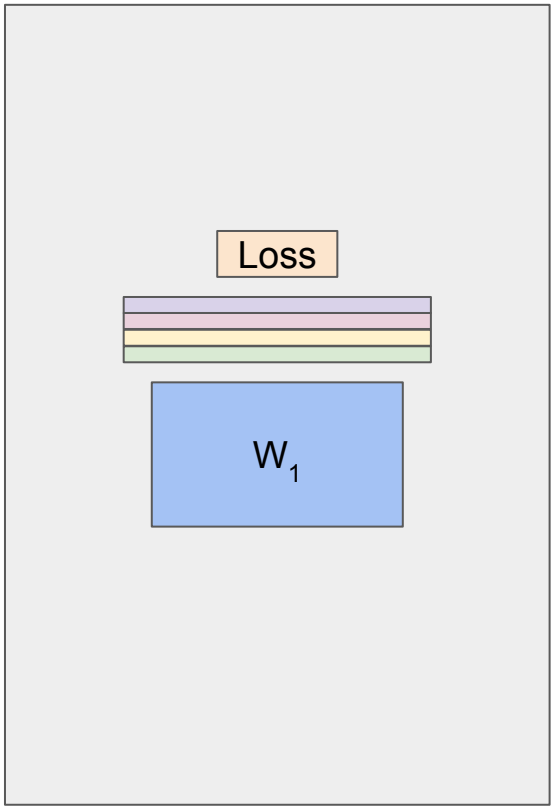
Problem: Model doesn't fit in GPU memory



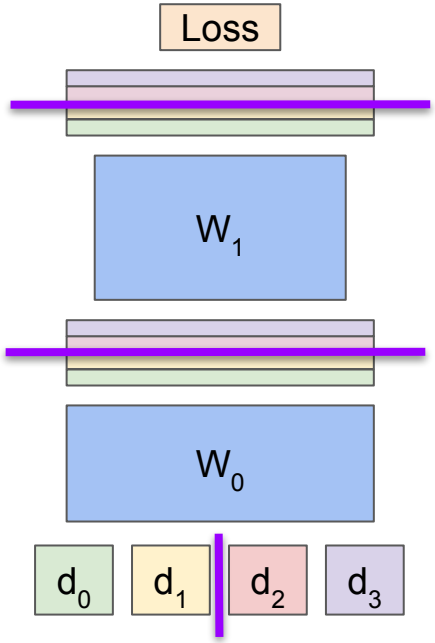
GPU 1



GPU 2



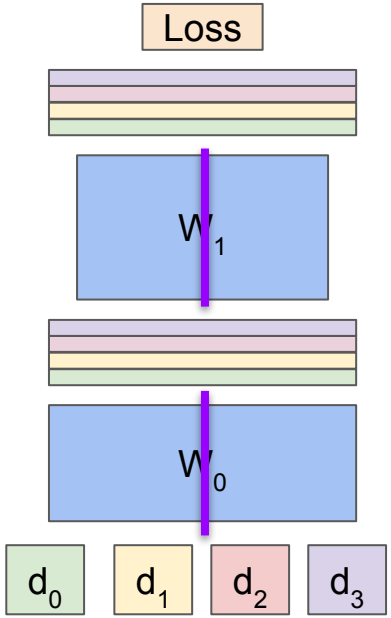
Data Parallelism



- Communicate (all-reduce) gradients after each backward pass
- All gpus still need a full model (memory issue)

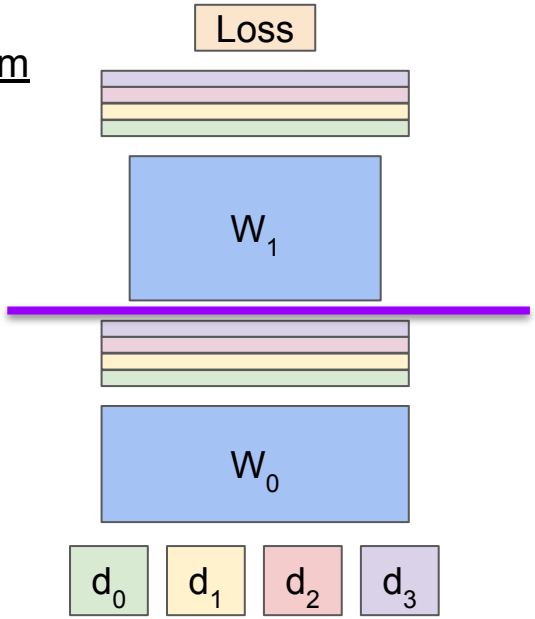
Tensor Parallelism

Model Parallelism



- Communicate activations after each layer in forward and backward pass (slow)

Pipeline Parallelism



- Communicate n times in forward/backward pass
- Pipeline “bubbles” (idle gpus)

Which do we do in practice?
All of these! (and more)

Which do we do in practice?
All of these! (and more)

Pros: We can now train big models on big batch sizes even with gpu memory limitations

Cons: Communication overhead and idle GPUs involved in parallelization drives up our time/cost

Which do we do in practice?
All of these! (and more)

Pros: We can now train big models on big batch sizes even with gpu memory limitations

Cons: Communication overhead and idle GPUs involved in parallelization drives up our time/cost

Recall our calculation using "ideal" GPU conditions

Llama = 5.48e23 flops
GPU A100 = 4e14 flops/s
Cost $4/hour

5.48e23/4e14
= 1,370,000,000 seconds
= 380,555 hours
= ~$1.5 million

Which do we do in practice?
All of these! (and more)

Pros: We can now train big models on big batch sizes even with gpu memory limitations

Cons: Communication overhead and idle GPUs involved in parallelization drives up our time/cost

Recall our calculation using "ideal" GPU conditions

Llama = 5.48e23 flops
GPU A100 = 4e14 flops/s
Cost $4/hour

5.48e23/4e14
= 1,370,000,000 seconds
= 380,555 hours
= ~$1.5 million

Reality

= 1,022,362 hours
= ~ $4 million

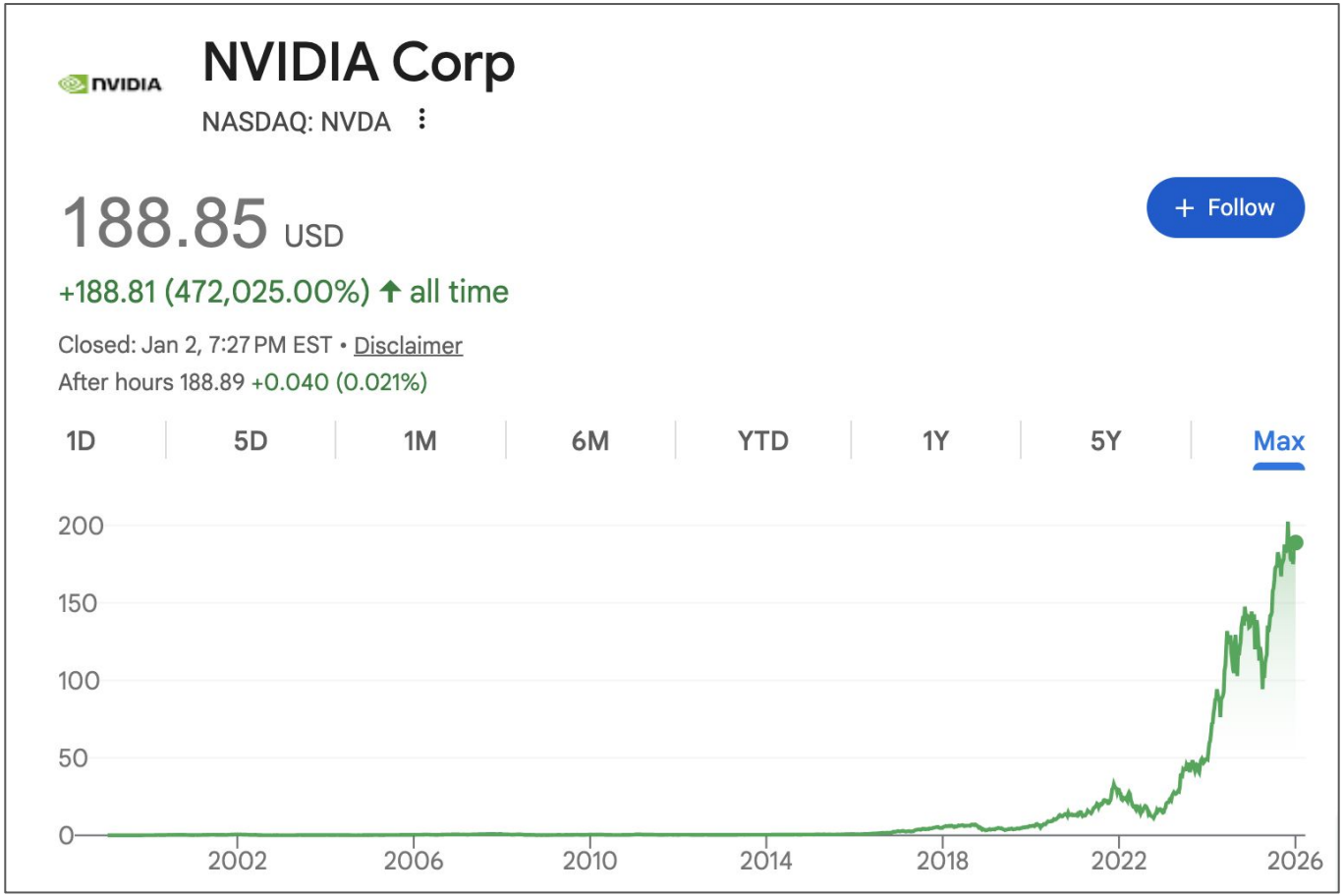
(Other reasons for this much slow down as well)

$4 million is so cheap by today's standards!

The $6 Million AI Bombshell: How DeepSeek Shook Wall Street And AI Leadership

Is DeepSeek's claim that it built its AI models with less than $6 million real — or misleading and calculated? Is the Chinese company telling the world tiny, unverified

https://www.forbes.com/sites/markminevich/2025/02/06/the-6-million-ai-bombshell-how-deepseek-shook-wall-street-and-ai-leadership/


NVIDIA Corp
NASDAQ: NVDA
188.85 USD
+ Follow
+188.81 (472,025.00%) all time
Closed: Jan 2, 7:27 PM EST • Disclaimer
After hours 188.89 +0.040 (0.021%)
1D
5D
1M
6M
YTD
1Y
5Y
Max
200
150
100
50
0
2002
2006
2010
2014
2018
2022
2026

Part 1: The Difficulty of Scaling

Part 2: Parallelism

Part 3: Scaling Smarter

- Making the best out of your FLOPs (Chinchilla)
- More parameters without more FLOPs (MoEs)
- Smaller parameters, bigger models (Quantization)

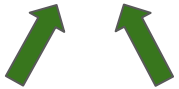
Making the best out of your FLOPs (Chinchilla)

$$\text{FLOPs} = \sim 6PD$$

How much of each?

Making the best out of your FLOPs (Chinchilla)

$$\text{FLOPs} = \sim 6PD$$



How much of each?

Observation: 30 billion param with 1 trillion training token is the same number of FLOPs as 60 billion params with 0.5 trillion tokens

Which to do?

Making the best out of your FLOPs (Chinchilla)

Idea:

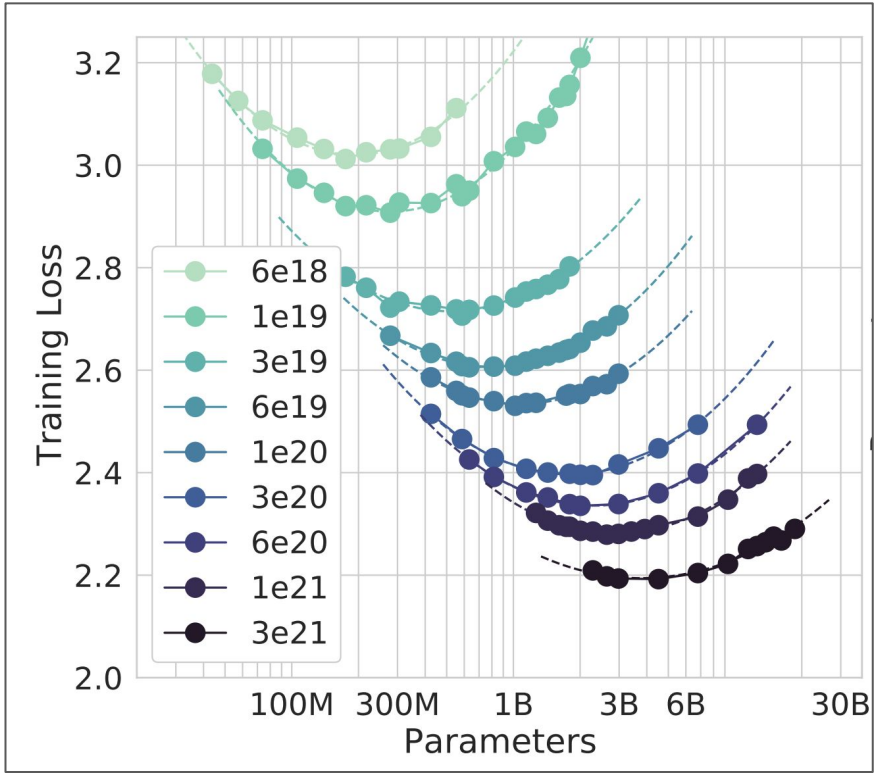
- pick a set a number of FLOPs (6e18)
- sweep different parameter and dataset size combos that have this number of FLOPs
- plot loss

Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

Making the best out of your FLOPs (Chinchilla)

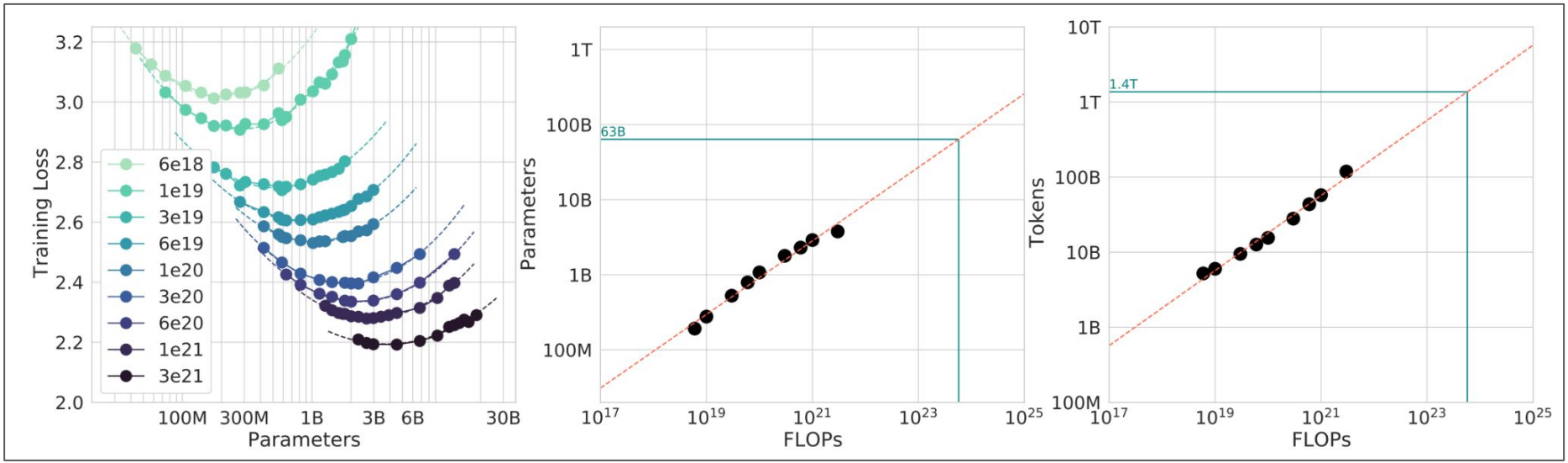
Idea:

- pick a set a number of FLOPs (6e18)
- sweep different parameter and dataset size combos that have this number of FLOPs
- plot loss



Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

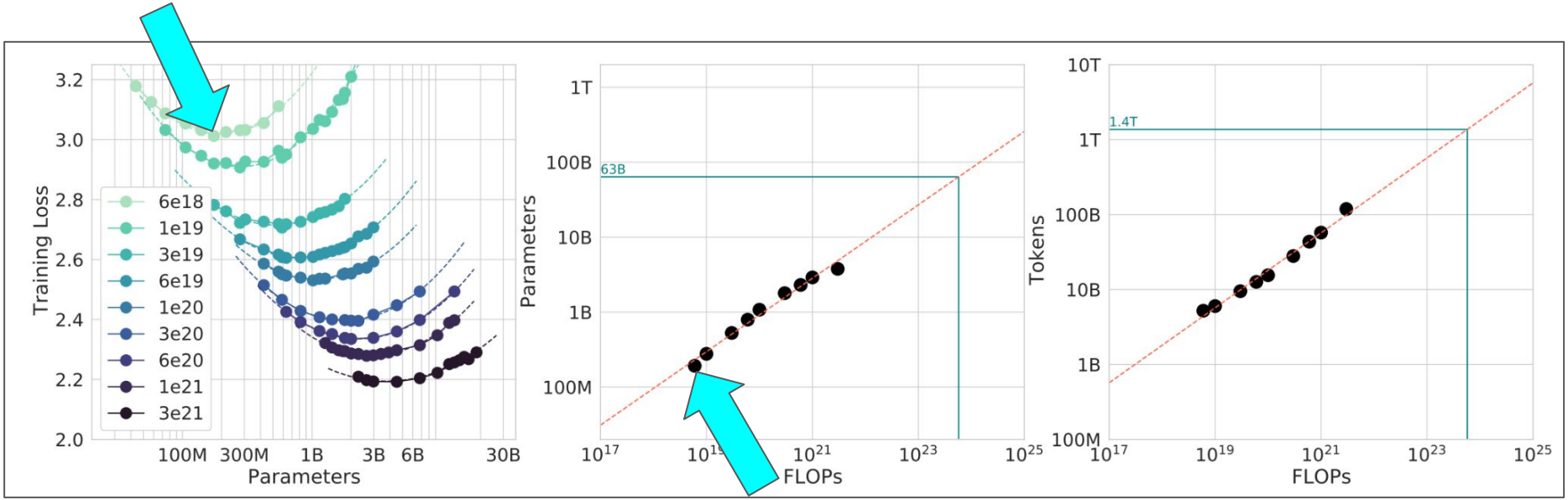
Making the best out of your FLOPs (Chinchilla)



Very much an active area of research!

Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

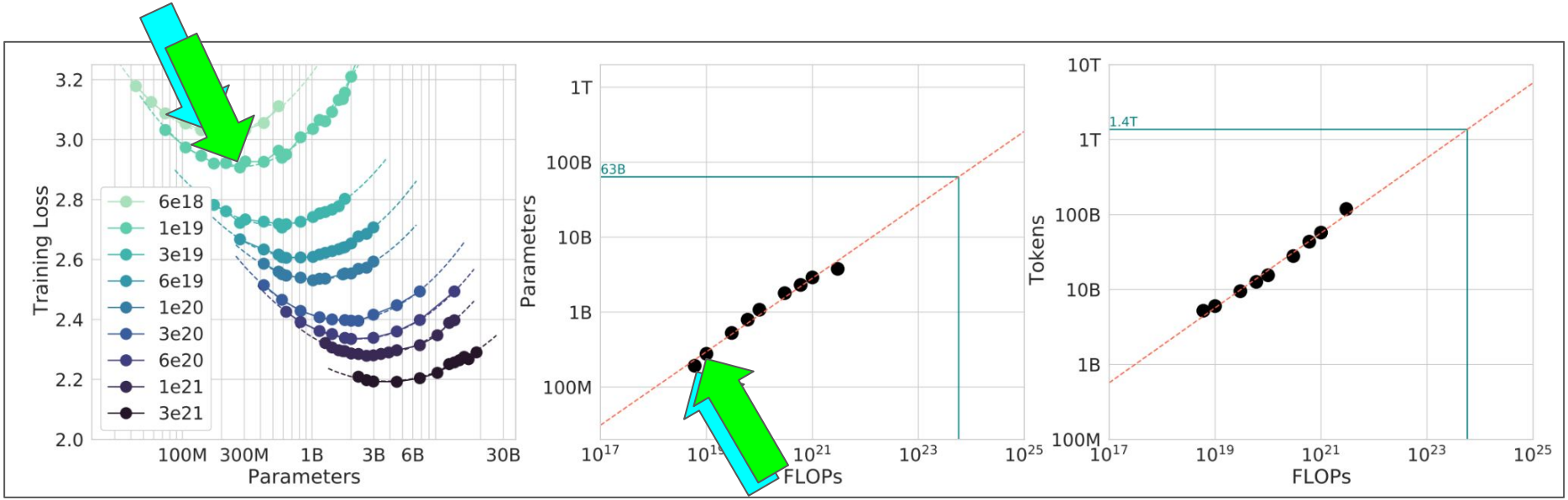
Making the best out of your FLOPs (Chinchilla)



Very much an active area of research!

Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

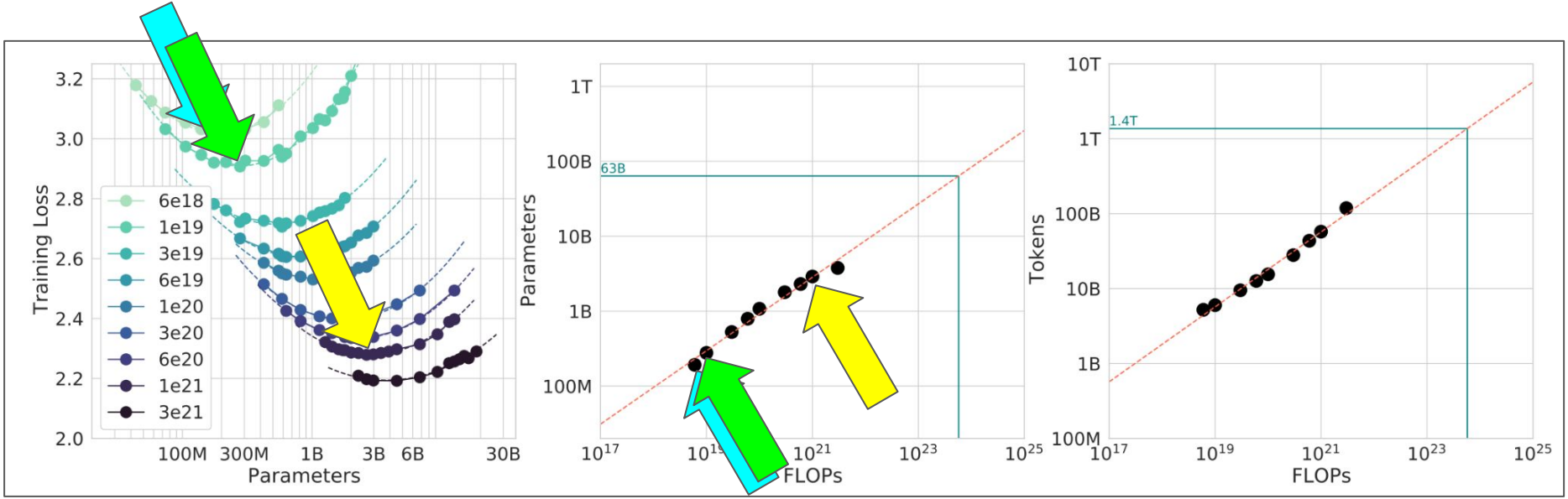
Making the best out of your FLOPs (Chinchilla)



Very much an active area of research!

Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

Making the best out of your FLOPs (Chinchilla)



Very much an active area of research!

Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

Smaller parameters, bigger models (Quantization)

Recall from earlier

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

Model Parameters	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Gradients	FP32 and FP16/BF16 = ~2*P	2*65 Billion bytes = 130 GB
Adam First Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
Adam Second Moment	FP32 = 4*P	4*65 Billion bytes = 260 GB
(sometimes) Adam Copy of params	FP32 = 4*P	4*65 Billion bytes = 260 GB
		~1 TB

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

-145320909488958745892090210056.090990832723736266 62

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

-145320909488958745892090210056.090990832723736266 62

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

- Sign (only ever takes 1 bit)

-145320909488958745892090210056.090990832723736266 62

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

- Sign (only ever takes 1 bit)
- Precision

-145320909488958745892090210056.090990832723736266 62

1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

- Sign (only ever takes 1 bit)
- Precision
- Range

-145320909488958745892090210056.090990832723736266 62

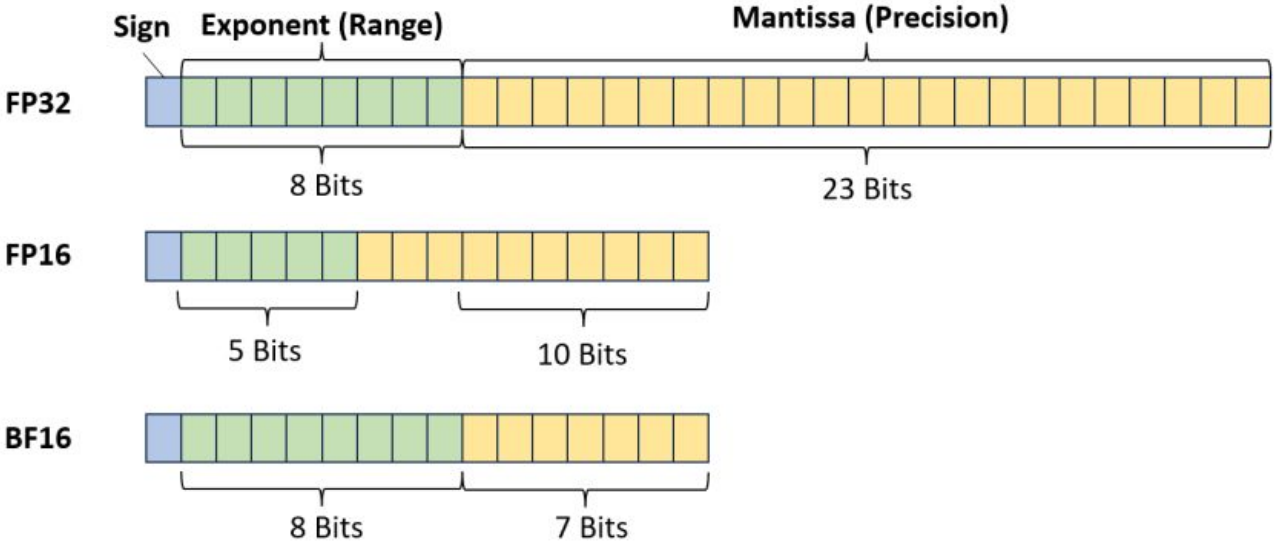
1 parameter = FP32, FP16, BF16 = 4 bytes or 2 bytes

What does this even mean?
Each bit allows us to communicate 1 piece of information.

- Sign (only ever takes 1 bit)
- Precision
- Range

Think of the two numbers in scientific notation: -1 * 5.468e18

Smaller parameters, bigger models (Quantization)



https://blog.eleuther.ai/transformer-math/

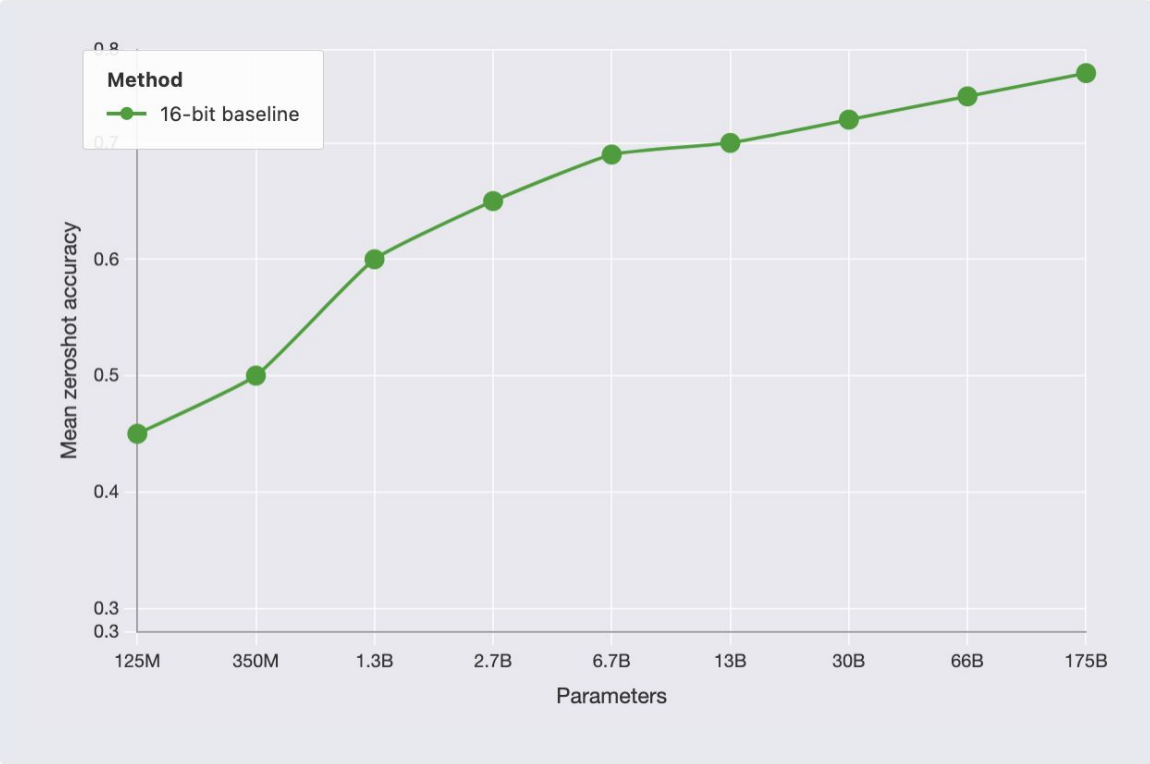
Smaller parameters, bigger models (Quantization)

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

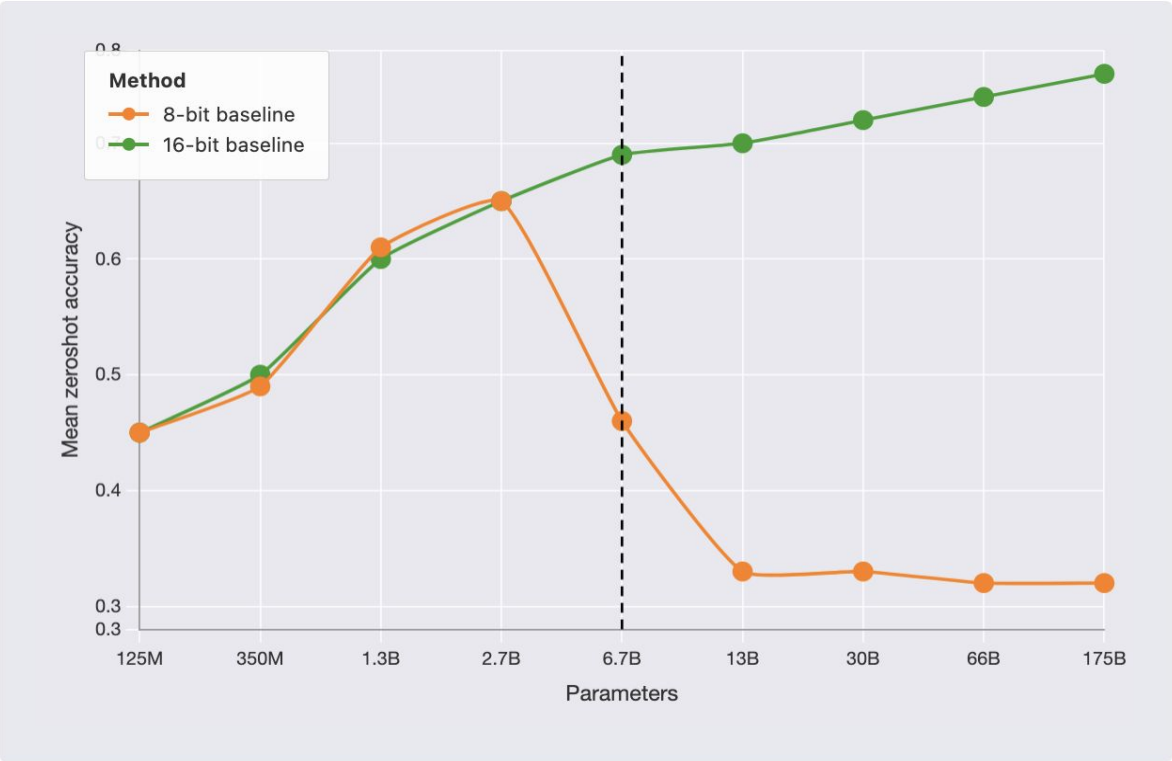
Tim Dettmers[λ*] Mike Lewis[†] Younes Belkada[§∓] Luke Zettlemoyer[†λ]

Note, this work specifically focuses on Inference but explains the concept nicely

Great blog post on this topic by the LLM.int8() author!



[Great blog post on this topic by the LLM.int8() author!]



[Great blog post on this topic by the LLM.int8() author!]

Observation: In large models, some features are very large, while most are very small. Quantizing to maintain the large features, causes the small features to collapse to a small number of values.

[-0.10, -0.23, 0.08, -0.38, -0.28, -0.29, -2.11, 0.34, -0.53, -67.0]

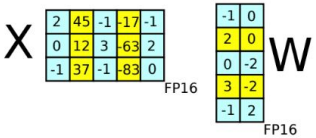
[-0.00, -0.00, 0.00, -0.53, -0.53, -0.53, -2.11, 0.53, -0.53, -67.00]

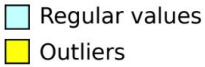
Great blog post on this topic by the LLM.int8() author!

Idea:

- Quantize large features with 16 bits
- Quantize small value with 8 bits

LLM.int8()



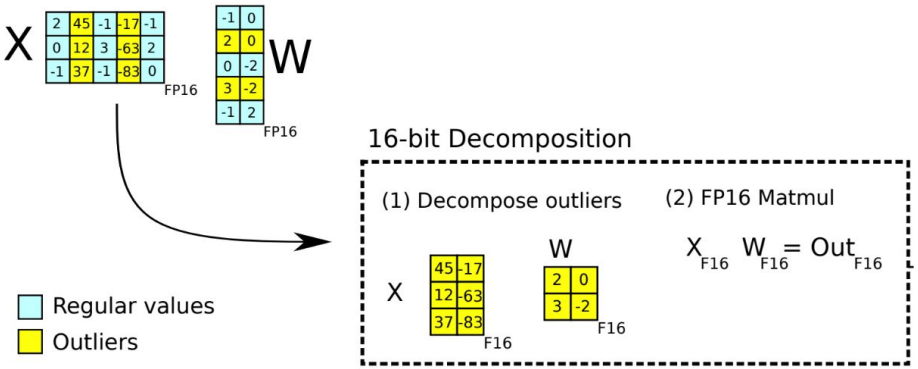


Great blog post on this topic by the LLM.int8() author!

Idea:

- Quantize large features with 16 bits
- Quantize small value with 8 bits

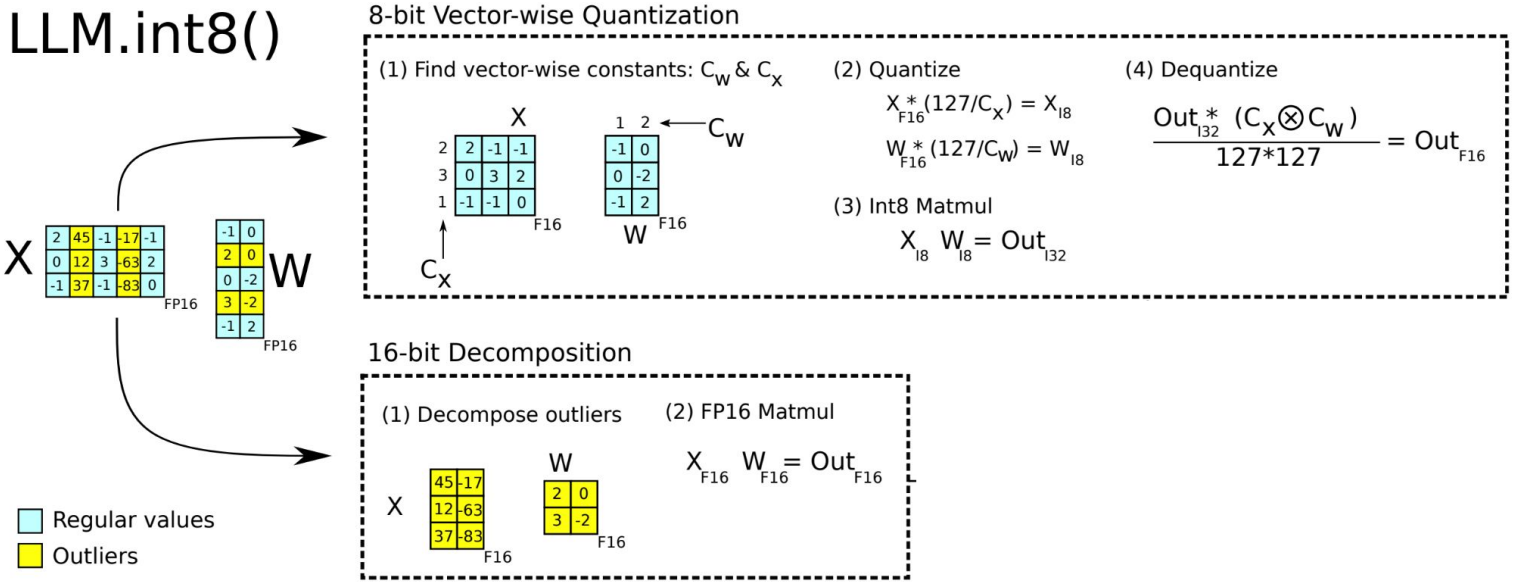
LLM.int8()



Great blog post on this topic by the LLM.int8() author!

Idea:

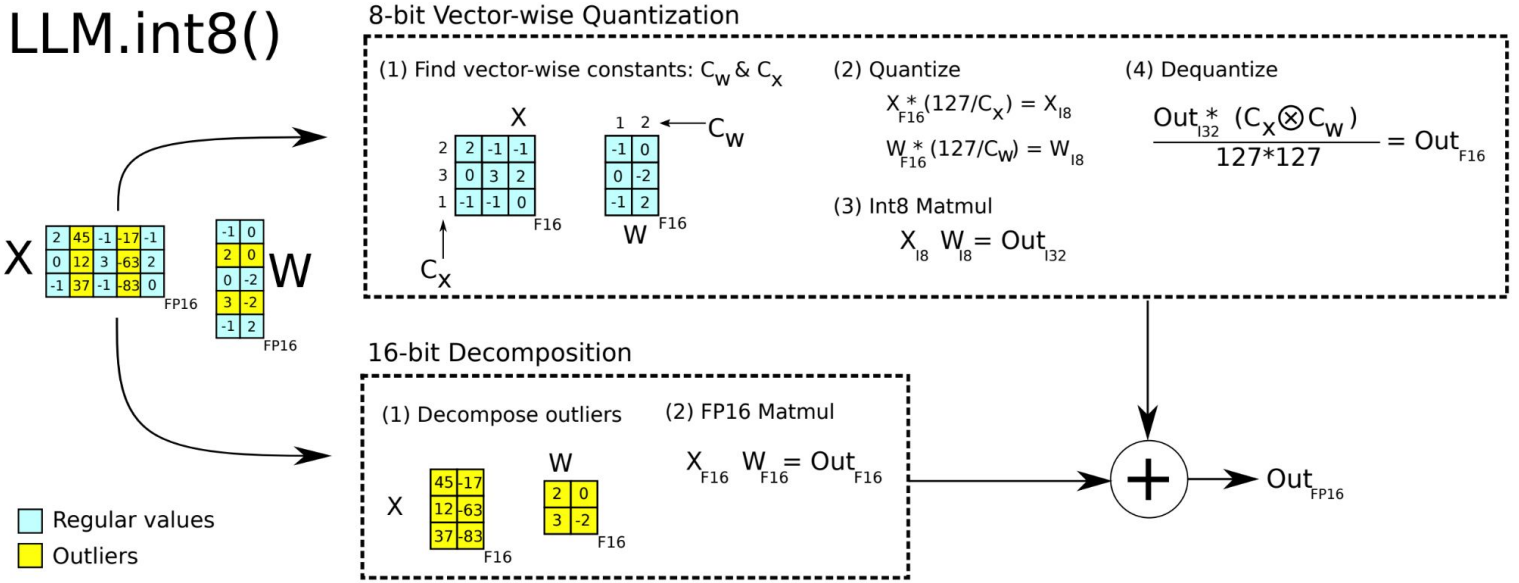
- Quantize large features with separate high precision
- Quantize small value with int8



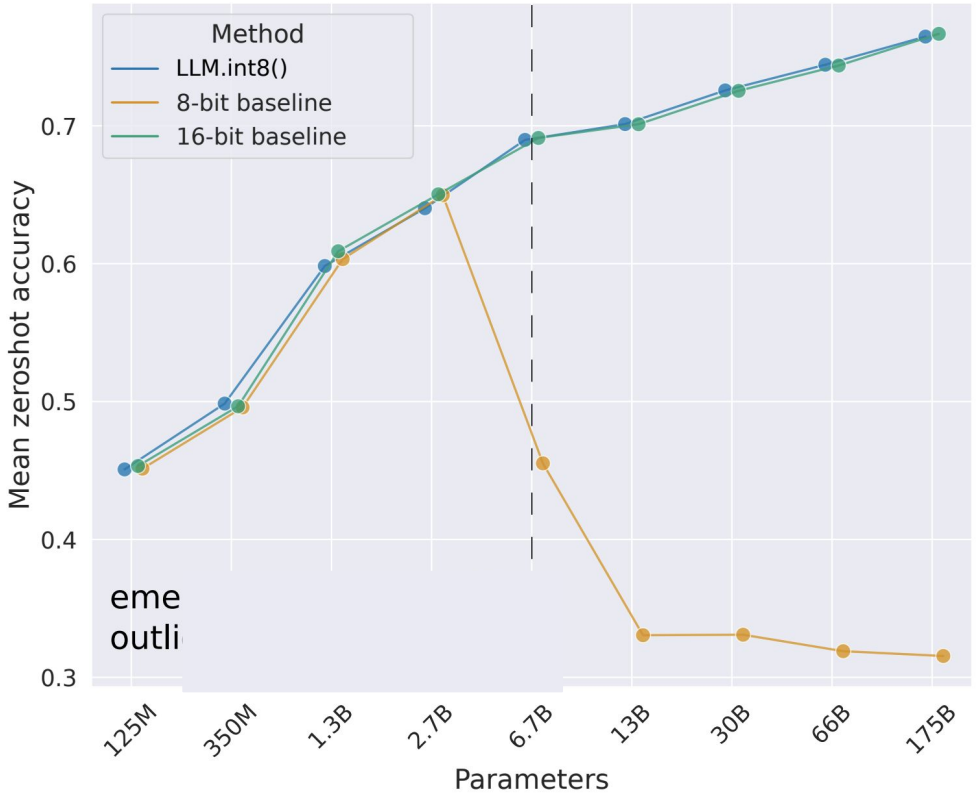
Great blog post on this topic by the LLM.int8() author!

Idea:

- Quantize large features with separate high precision
- Quantize small value with int8



Great blog post on this topic by the LLM.int8() author!



Matches full 16-bit model!

This is called “mixed precision” and it very commonly used

Great blog post on this topic by the LLM.int8() author!

Quantization / mixed precision in training!

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

Guangxuan Xiao [*1] **Ji Lin** [*1] **Mickael Seznec** [2] **Hao Wu** [2] **Julien Demouth** [2] **Song Han** [1]

https://github.com/mit-han-lab/smoothquant

FP8 FORMATS FOR DEEP LEARNING

Paulius Micikevicius, Dusan Stosic, Patrick Judd, John Kamalu, Stuart Oberman, Mohammad Shoeybi, Michael Siu, Hao Wu
NVIDIA
{pauliusm, dstosic, pjudd, jkamalu, soberman, mshoeybi, msiu, skyw}@nvidia.com

Neil Burgess, Sangwon Ha, Richard Grisenthwaite
Arm
{neil.burgess, sangwon.ha, richard.grisenthwaite}@arm.com

Naveen Mellempudi, Marius Cornea, Alexander Heinecke, Pradeep Dubey
Intel
{naveen.k.mellempudi, marius.cornea, alexander.heinecke, pradeep.dubey}@intel.com

Stable and low-precision training for large-scale vision-language models

Mitchell Wortsman[*1] **Tim Dettmers**[*1] **Luke Zettlemoyer**[12] **Ari Morcos**[†2]
Ali Farhadi[†1] **Ludwig Schmidt**[†134]

More parameters without more FLOPs

More parameters without more FLOPs

Recall our compute cost of ~6PD

This means if we want a bigger model, we have linearly more FLOPs.

However more parameters can encode more information, decrease loss…

Idea: what if we don’t use every parameter for every forward pass?

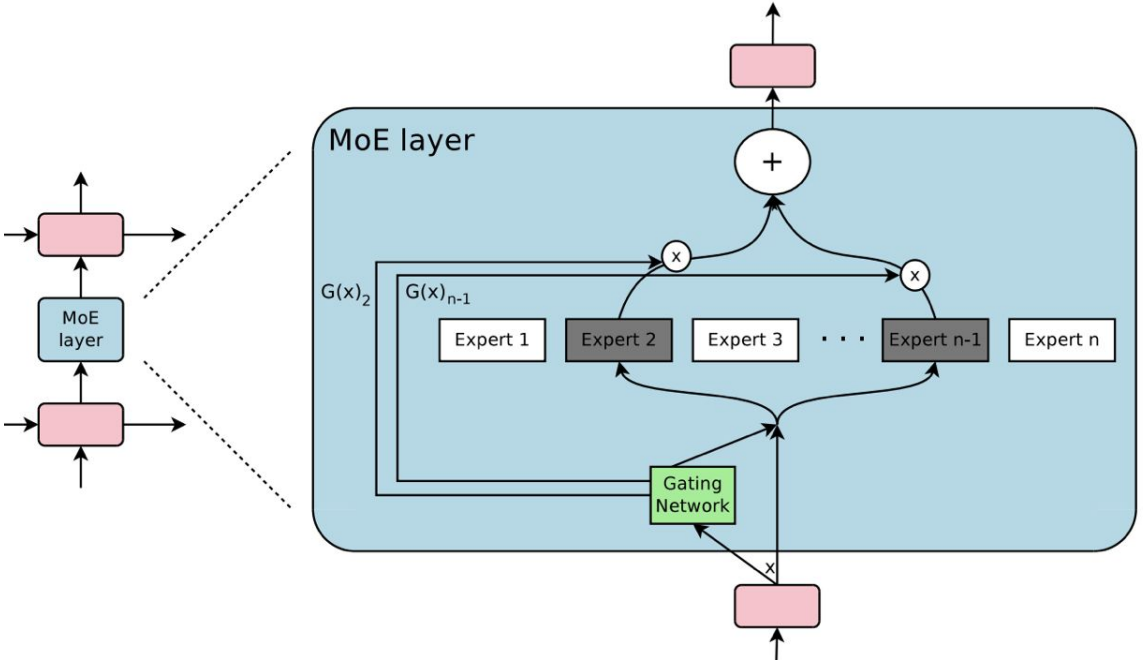
Idea: what if we don't use every parameter for every forward pass?

OUTRAGEOUSLY LARGE NEURAL NETWORKS:
THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini*†[1], Krzysztof Maziarz*[2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

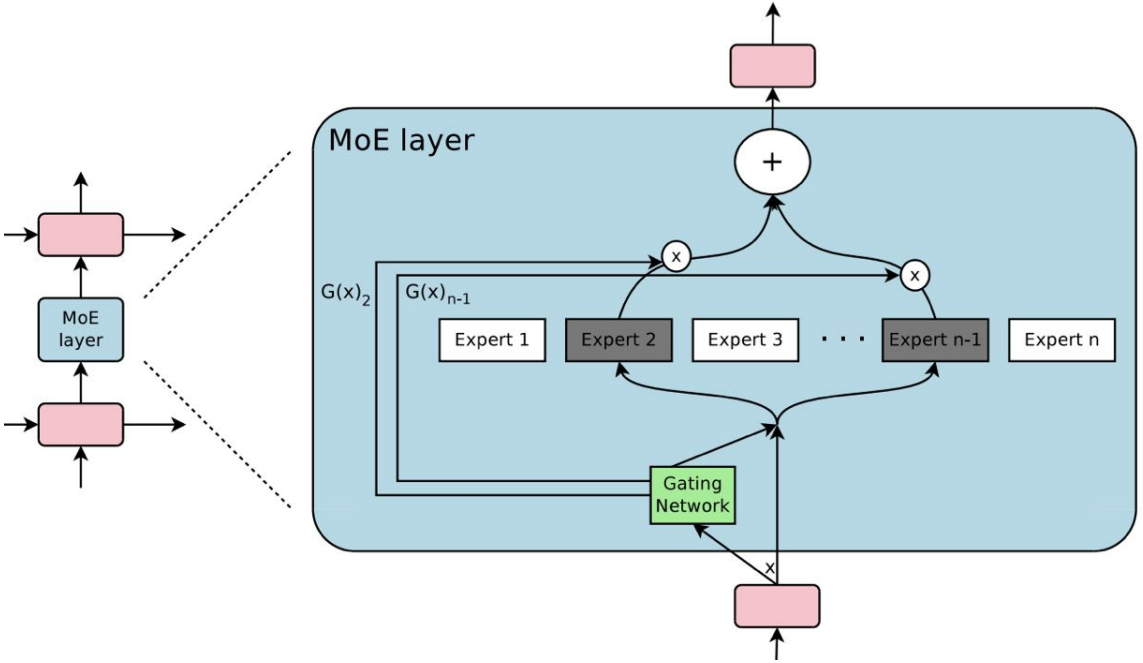
2017

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

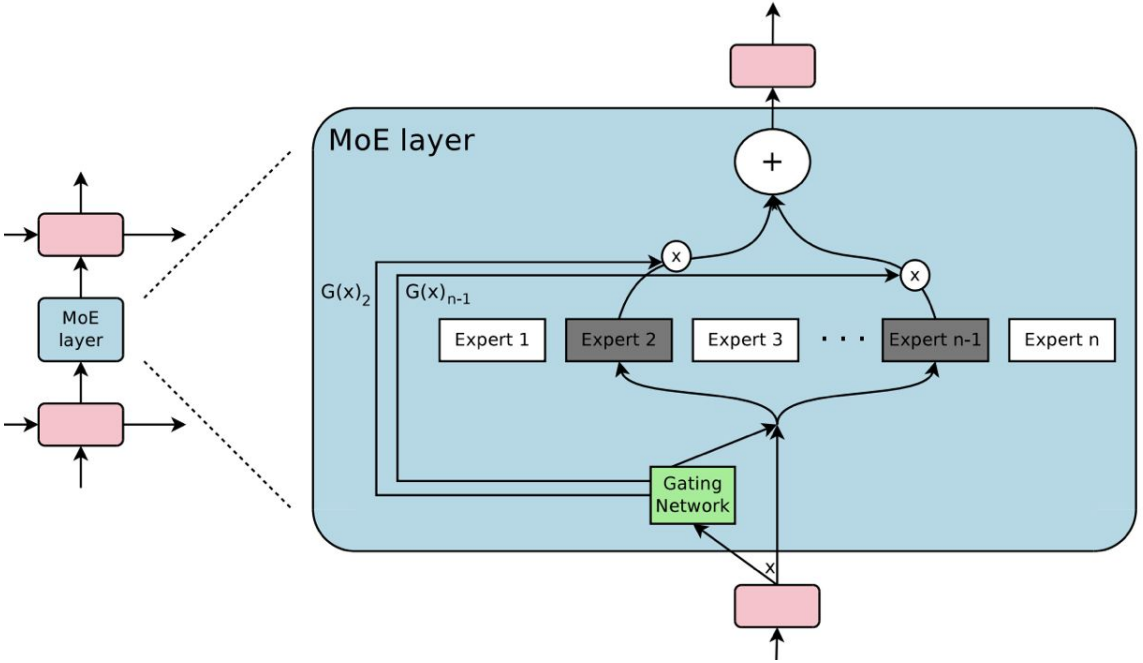
Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



$$y = \sum_{i=1}^{n} E_i(x)$$

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

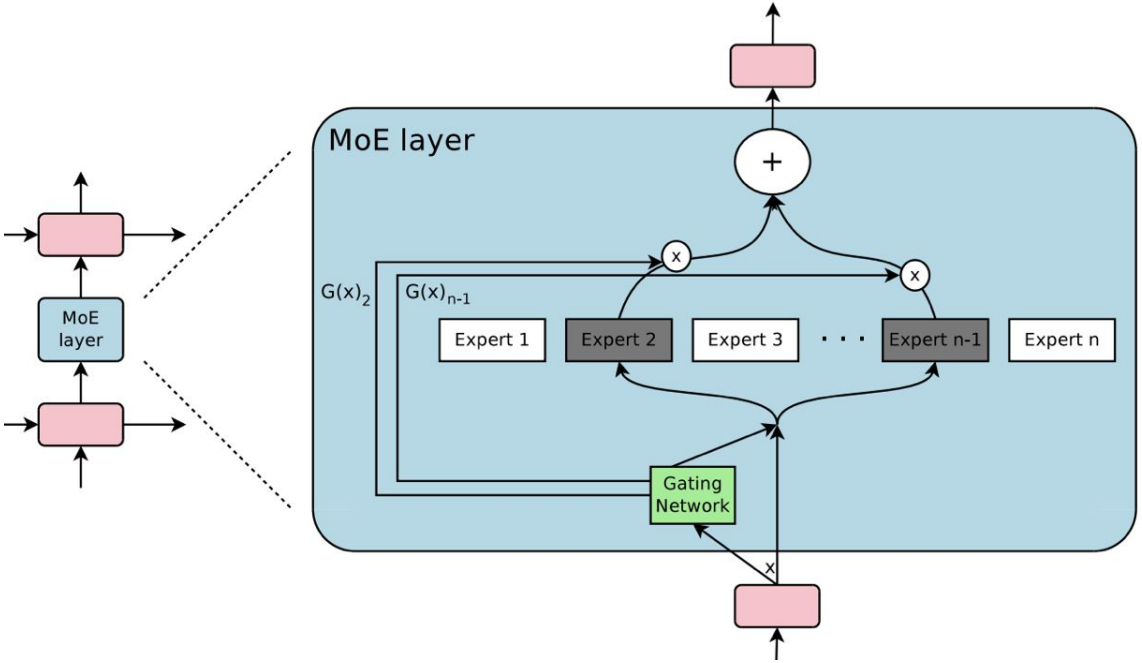
Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer

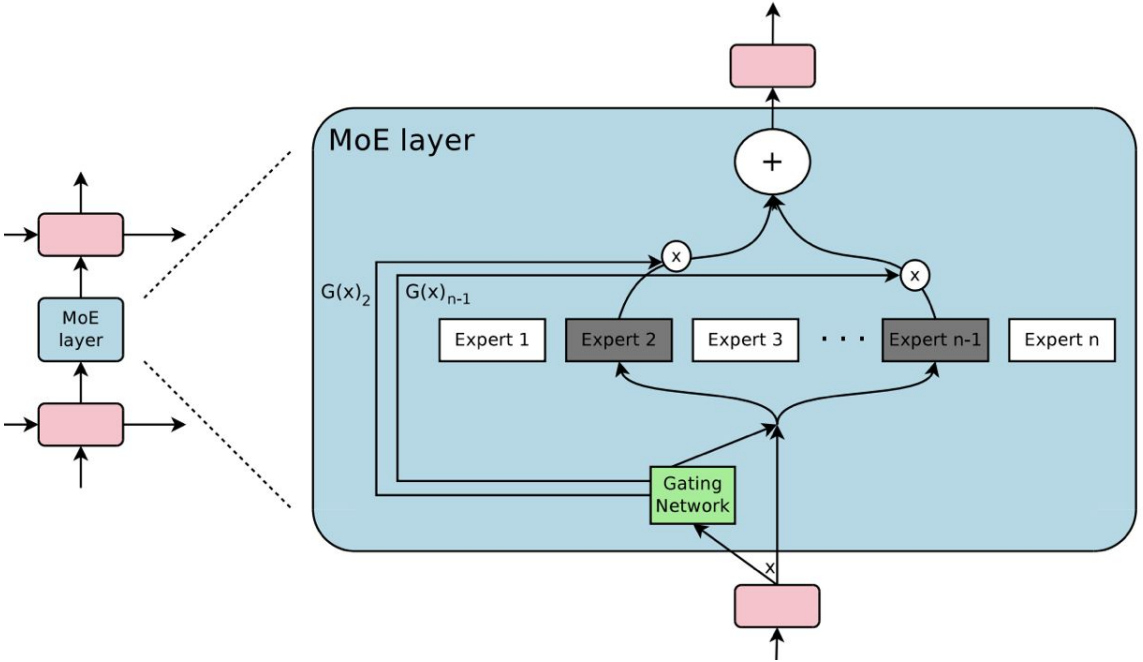


$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

$$G_\sigma(x) = Softmax(x \cdot W_g)$$

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

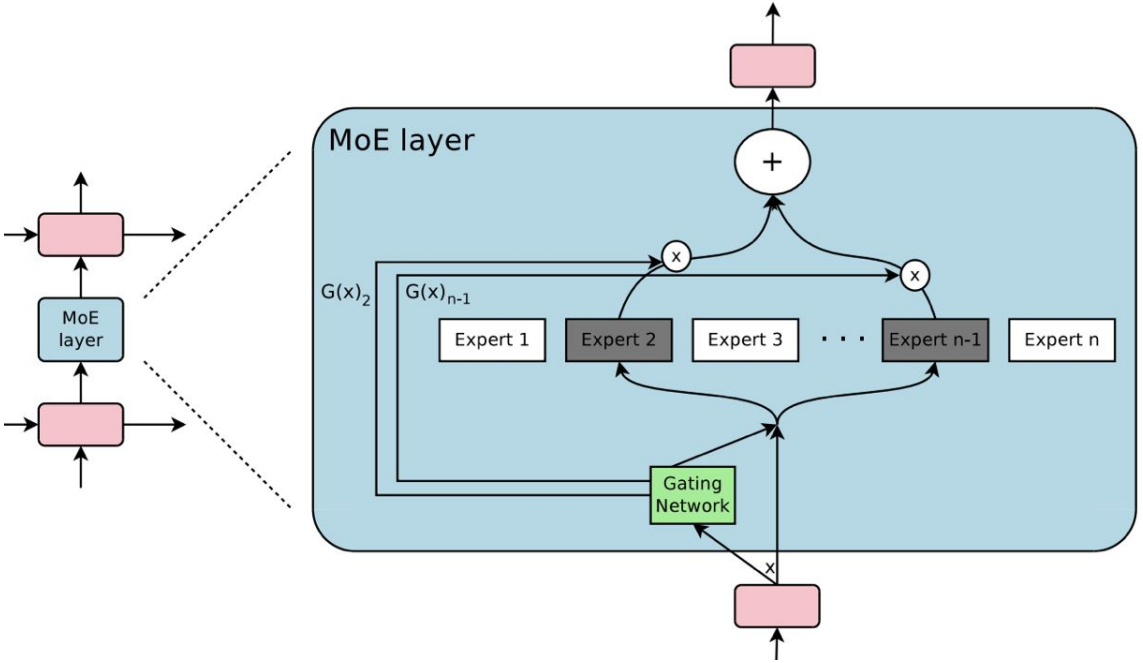
~~$G_\sigma(x) = Softmax(x \cdot W_g)$~~

$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + \text{noise}$$

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

~~$G_\sigma(x) = Softmax(x \cdot W_g)$~~

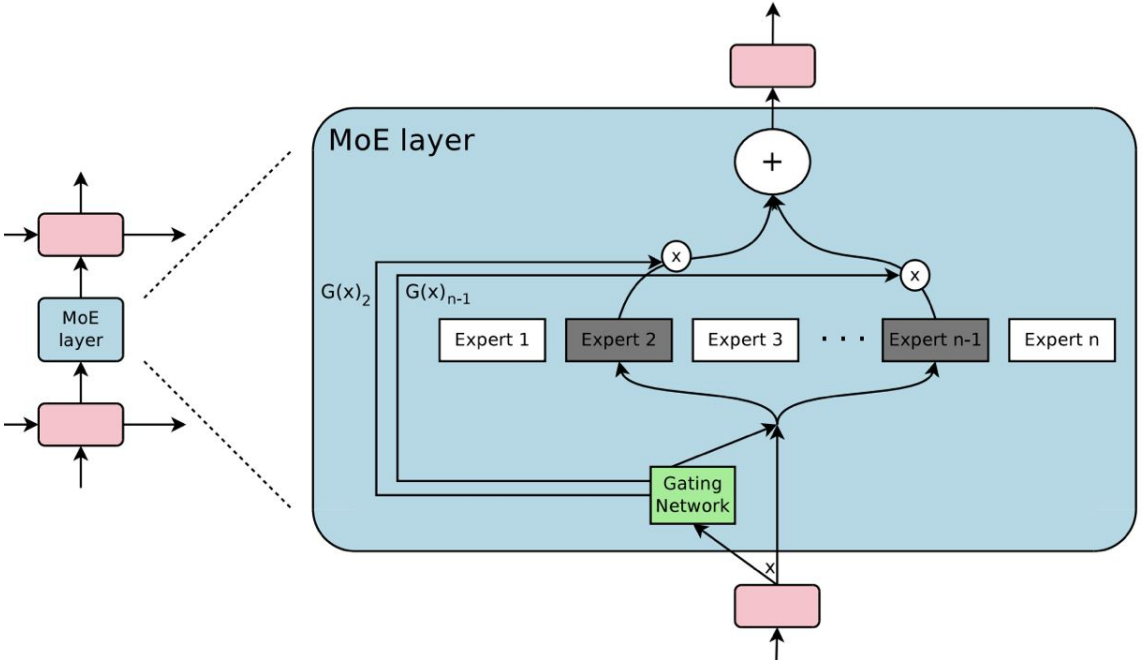
$$G(x) = Softmax(KeepTopK(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + \text{noise}$$

Important: Adds terms to loss function penalizing always choosing the same expert

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer



$$y = \sum_{i=1}^{n} G(x)_i E_i(x)$$

~~$G_\sigma(x) = Softmax(x \cdot W_g)$~~

$$G(x) = Softmax(KeepTopK(H(x), k))$$

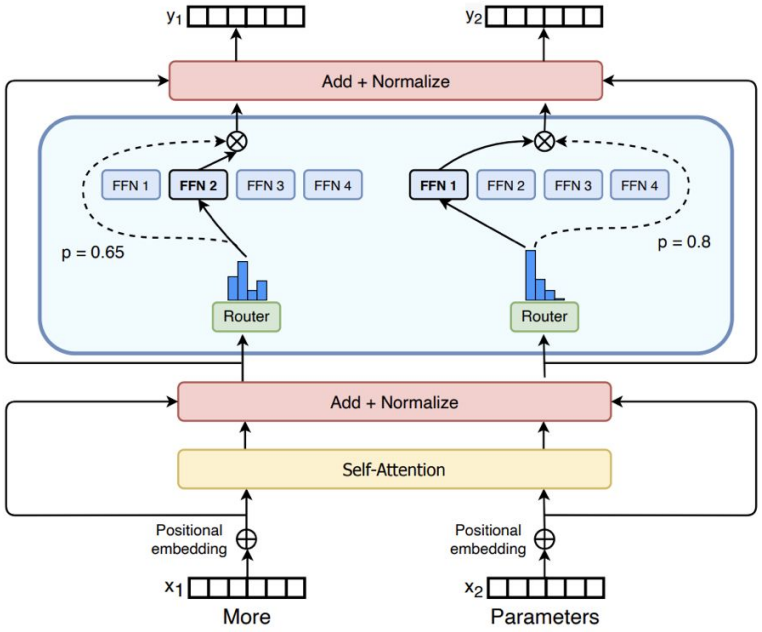
$$H(x)_i = (x \cdot W_g)_i + \text{noise}$$

Important: Adds terms to loss function penalizing always choosing the same expert

Scaled to 100 Billion params!

Shazeer et al., "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," ICLR, 2017.

Switch Transformers (2022)

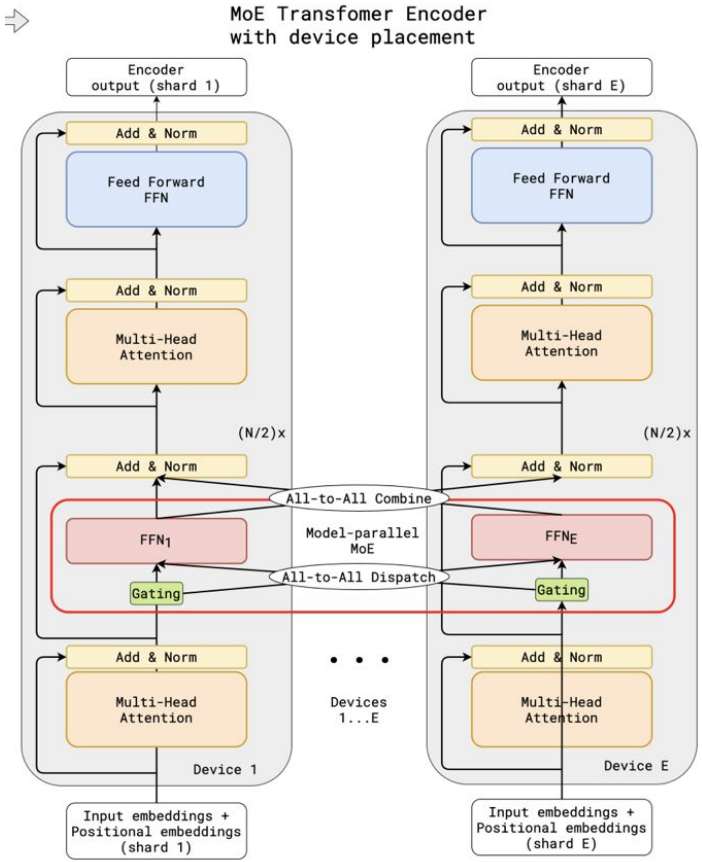


Super similar

- Only chooses 1 expert per embedding
- Simplified load balancing loss
- Scales to 1.4 T params

Fedus et al., "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," JMLR, 2022.

Gshard



New type of model parallelism

- Fewer communications than standard tensor parallelism
- No Pipeline bubble problem

Lepikhin et al., "GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding," ICLR, 2021.

OpenAI

August 5, 2025 Release Product

Introducing gpt-oss

gpt-oss-120b and gpt-oss-20b push the frontier of open-weight reasoning models

Model	Layers	Total Params	Active Params Per Token	Total Experts	Active Experts Per Token
gpt-oss-120b	36	117B	5.1B	128	4
gpt-oss-20b	24	21B	3.6B	32	4

https://openai.com/index/introducing-gpt-oss/

Lots more stuff we aren't covering here….

For Training…

Low Rank Fine-tuning
Efficient forms of attention
Gradient checkpointing (recompute activations as needed)

For Inference…

KV Caching
Speculative decoding (use a smaller model when you can)

Part 1: The Difficulty of Scaling

Part 2: Parallelism

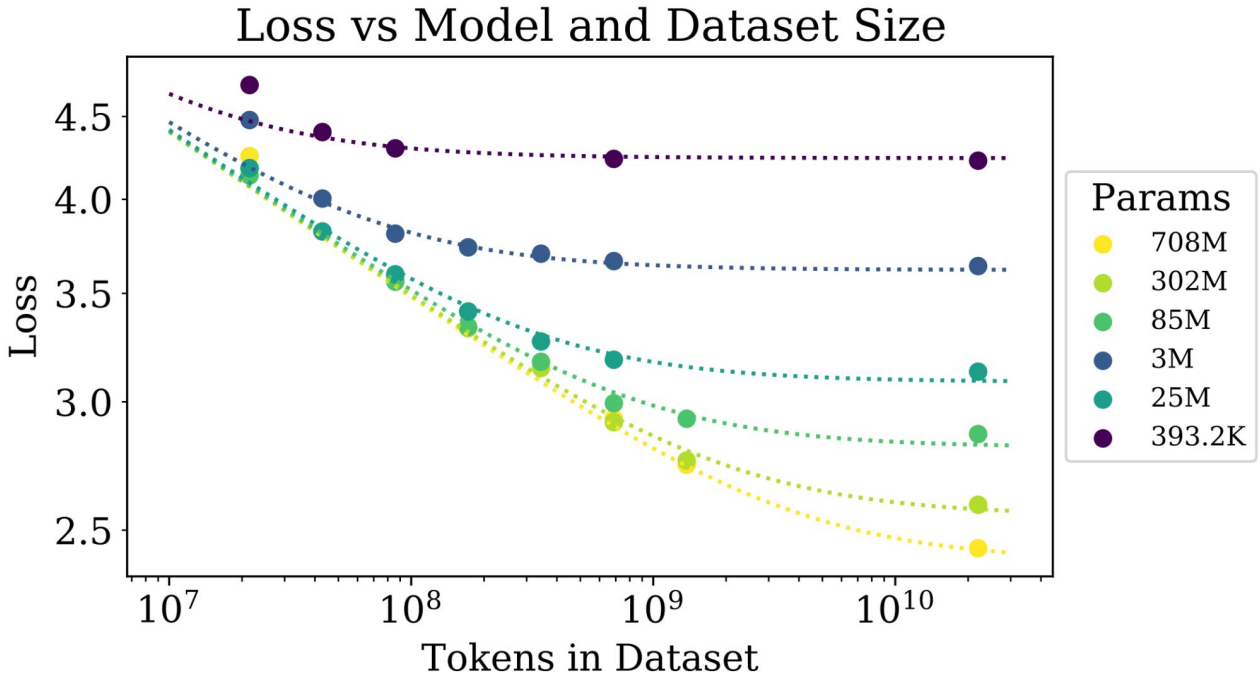
Part 3: Scaling Smarter

Part 4: Research on a Budget

- Cutting edge research without $10 million

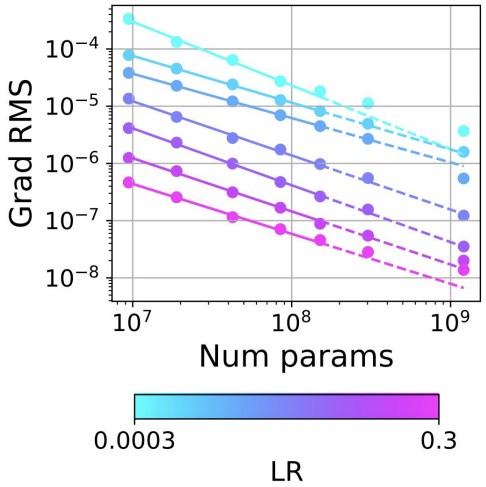
We can use small models to predict big models!

We can use small models to predict big models!



We can use small models to predict big models!

Small-scale proxies for large-scale Transformer training instabilities

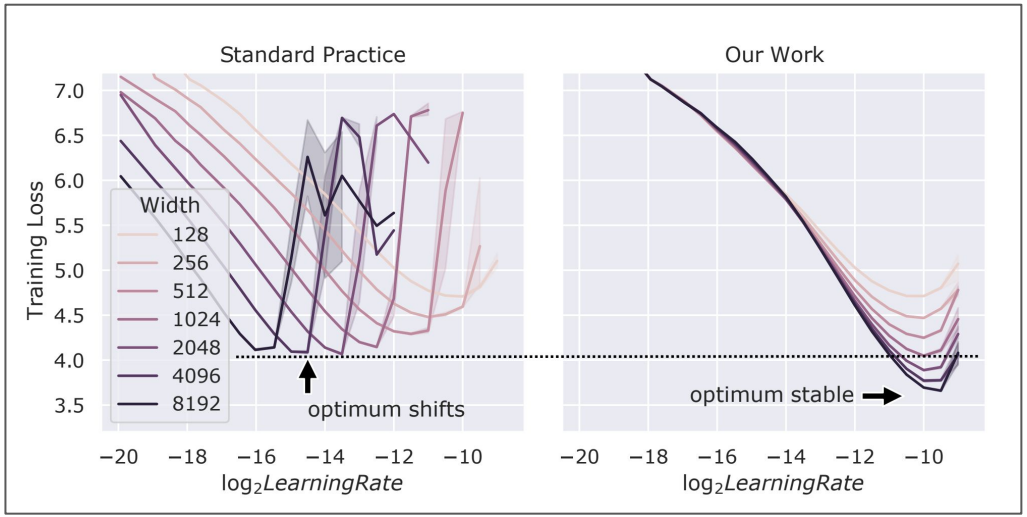


Looks at trends in how gradients change as models get bigger!

Predicted issue for large models before ever running one!

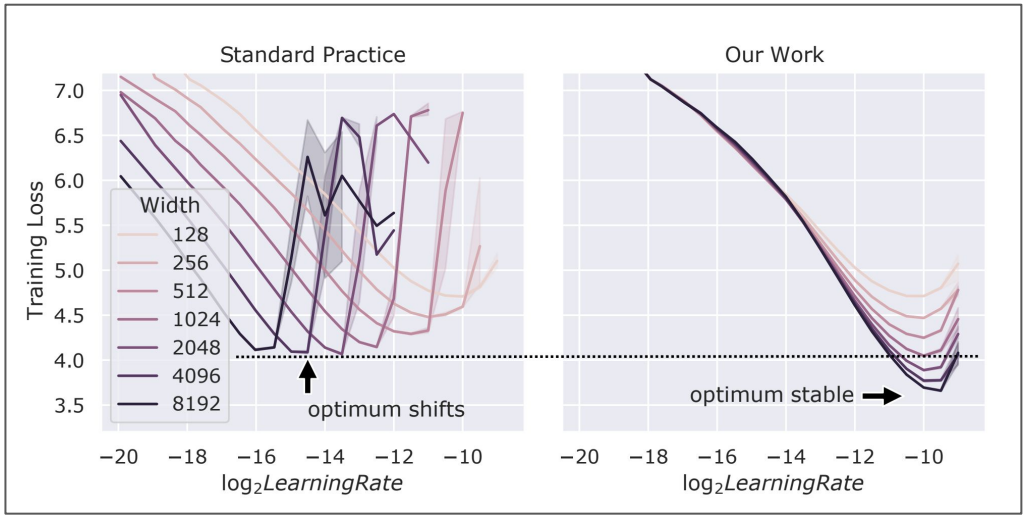
Wortsman et al., "Small-Scale Proxies for Large-Scale Transformer Training Instabilities," arXiv:2309.14322, 2023.

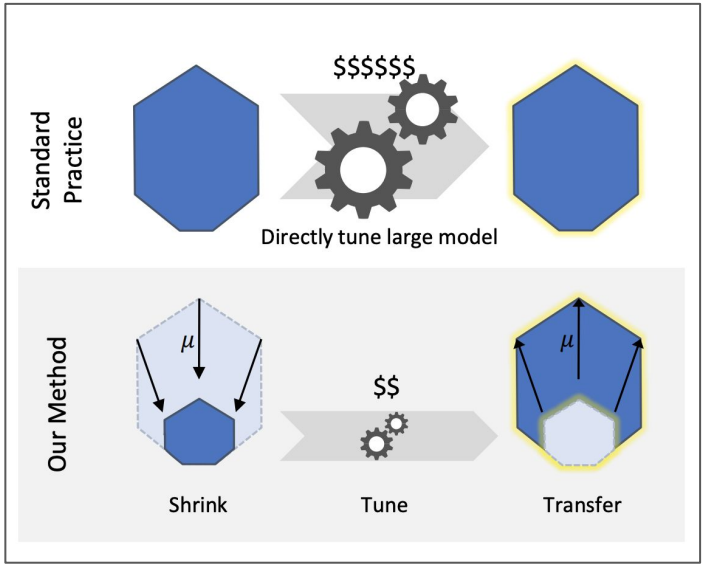
We can use small models to predict big models!



Yang et al., "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer," arXiv:2203.03466, 2022.

We can use small models to predict big models!





Yang et al., "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer," arXiv:2203.03466, 2022.

Using Small Models to predict Big Models

GPT-4

This report also discusses a key challenge of the project, developing deep learning infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to make predictions about the expected performance of GPT-4 (based on small runs trained in similar ways) that were tested against the final run to increase confidence in our training.

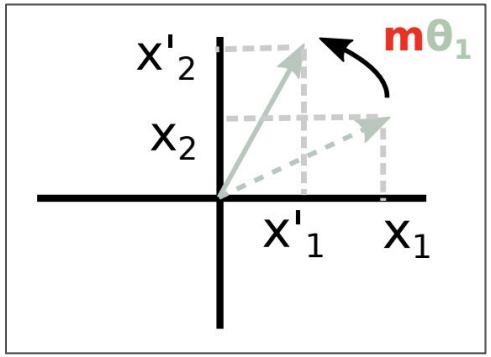
OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.

Many things used in Big Models were published on Small Models

Many things used in Big Models were published on Small Models

RoPE

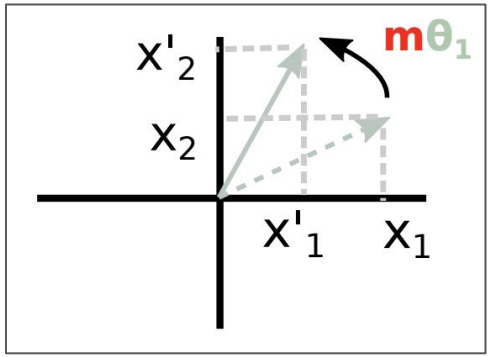
https://arxiv.org/pdf/2104.09864



Many things used in Big Models were published on Small Models

RoPE

https://arxiv.org/pdf/2104.09864



SwiGLU

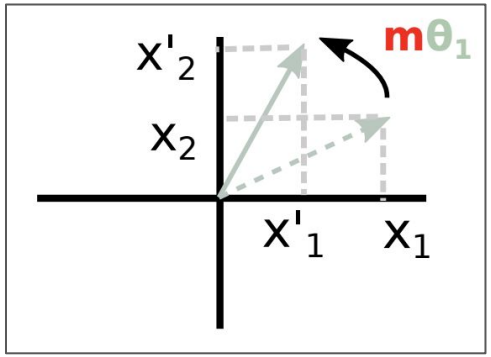
https://arxiv.org/pdf/2002.05202

Training Steps	65,536	524,288
$\text{FFN}_{\text{ReLU}}(baseline)$	1.997 (0.005)	1.677
FFN_{GELU}	1.983 (0.005)	1.679
$\text{FFN}_{\text{Swish}}$	1.994 (0.003)	1.683
FFN_{GLU}	1.982 (0.006)	1.663
$\text{FFN}_{\text{Bilinear}}$	1.960 (0.005)	1.648
$\text{FFN}_{\text{GEGLU}}$	**1.942** (0.004)	**1.633**
$\text{FFN}_{\text{SwiGLU}}$	**1.944** (0.010)	**1.636**
$\text{FFN}_{\text{ReGLU}}$	1.953 (0.003)	1.645

Many things used in Big Models were published on Small Models

RoPE

https://arxiv.org/pdf/2104.09864



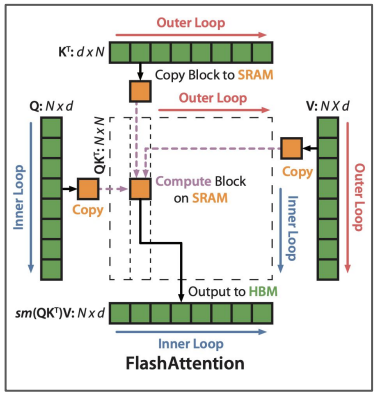
SwiGLU

https://arxiv.org/pdf/2002.05202

Training Steps	65,536	524,288
FFN_{ReLU} (*baseline*)	1.997 (0.005)	1.677
FFN_{GELU}	1.983 (0.005)	1.679
$\text{FFN}_{\text{Swish}}$	1.994 (0.003)	1.683
FFN_{GLU}	1.982 (0.006)	1.663
$\text{FFN}_{\text{Bilinear}}$	1.960 (0.005)	1.648
$\text{FFN}_{\text{GEGLU}}$	**1.942** (0.004)	**1.633**
$\text{FFN}_{\text{SwiGLU}}$	**1.944** (0.010)	**1.636**
$\text{FFN}_{\text{ReGLU}}$	1.953 (0.003)	1.645

FlashAttention

https://arxiv.org/abs/2205.14135

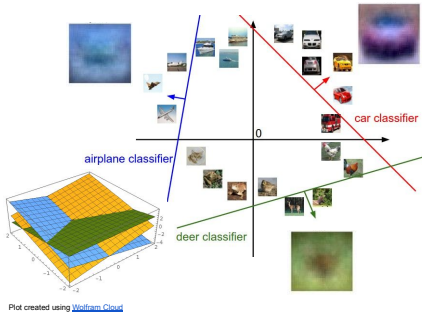


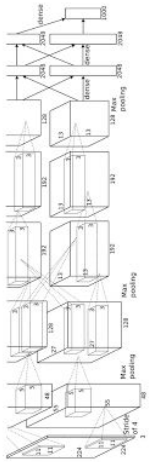
Plus way way more!

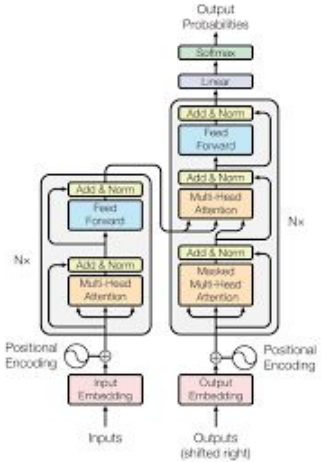
And we don't even know what cool research closed-source models are using

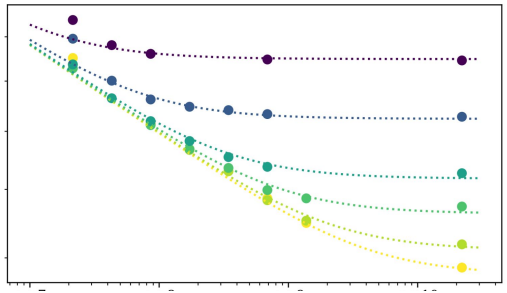
Summary:

- It's a great time to be an AI researcher!
- Lot's of research opportunities at scale
- Research at small scale transfers to work at big scale
- You have all the tools to do it!









Memory Calculation

https://huggingface.co/spaces/hf-accelerate/model-memory-usage

https://blog.eleuther.ai/transformer-math/

https://github.com/zhengzangw/awesome-huge-models

https://medium.com/@dzmitrybahdanau/the-flops-calculus-of-language-model-training-3b19c1f025e4

GPU stuff:
https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture09.pdf