Efficient LLMs

Yatin Nandwani
Research Scientist, IBM Research

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Semester 1,
2025-2026

IBM Research, India
Conversational-AI

Yatin Nandwani

Sonam Mishra

Dinesh Khandelwal

Gaurav Pandey

Vineet Kumar

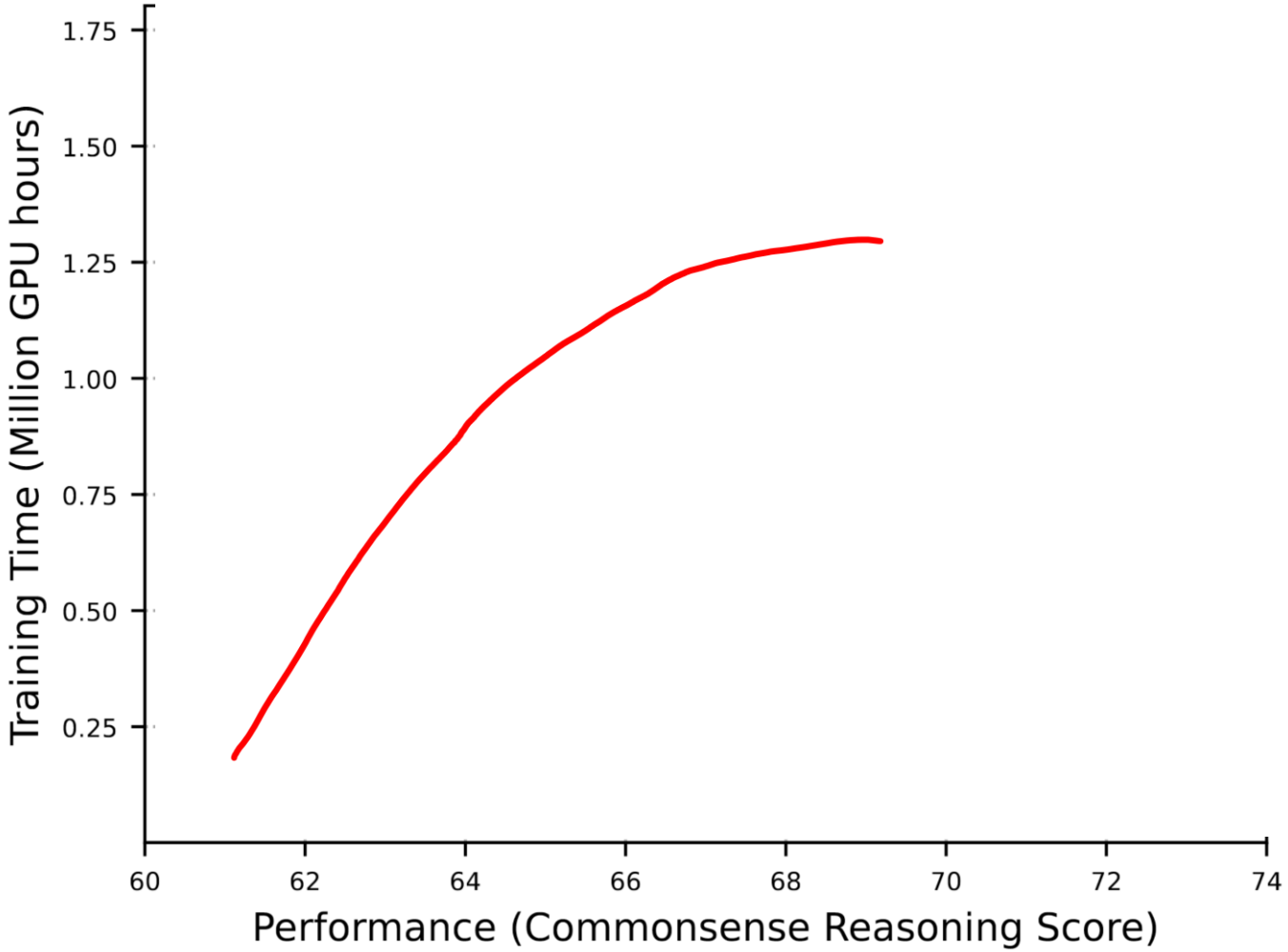
Meghanadh Pulivarthi

Kushagra Bhushan

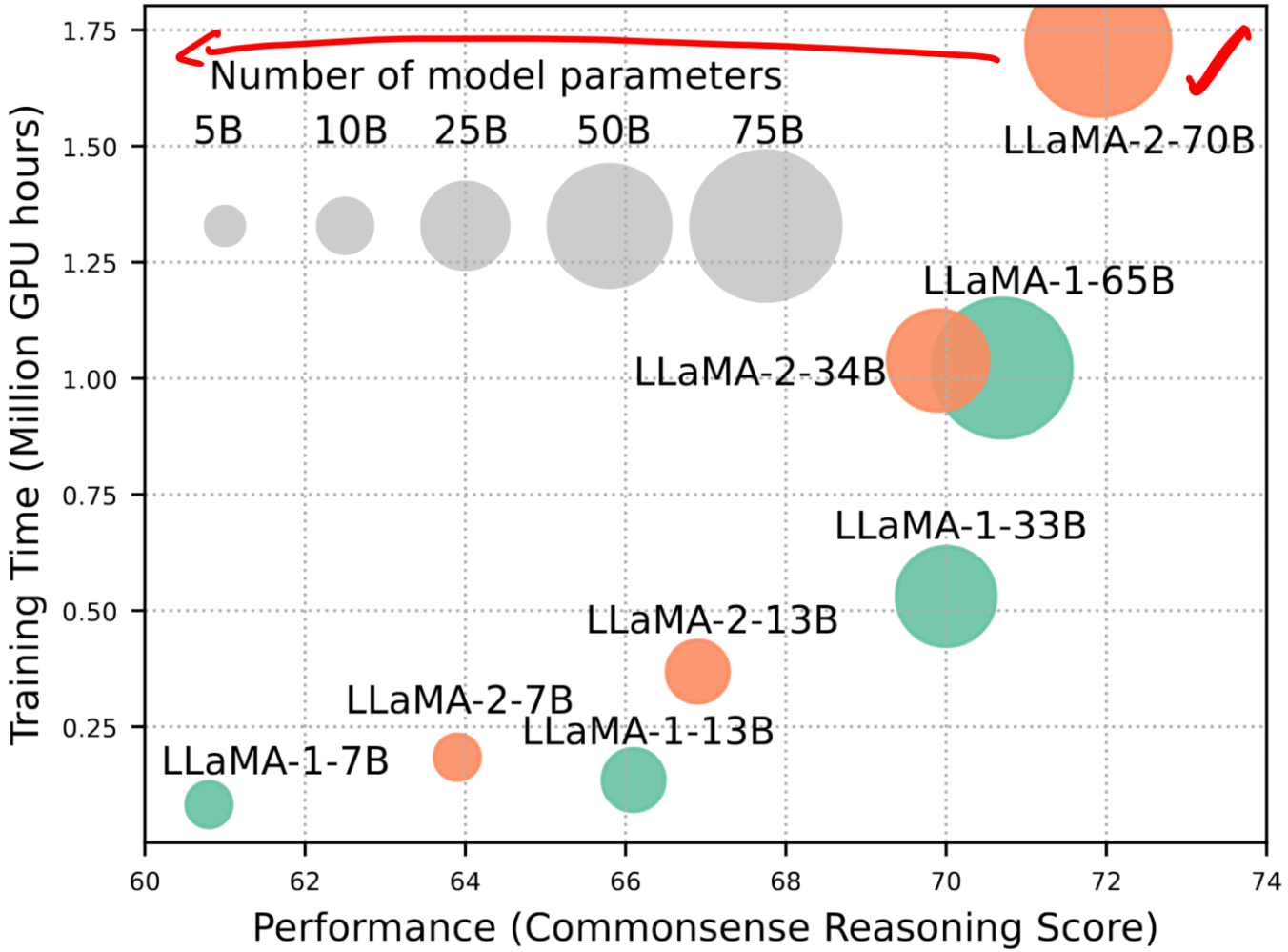
Dinesh Raghu

Sachindra Joshi

Training Resources vs Performance



Training Resources vs Performance

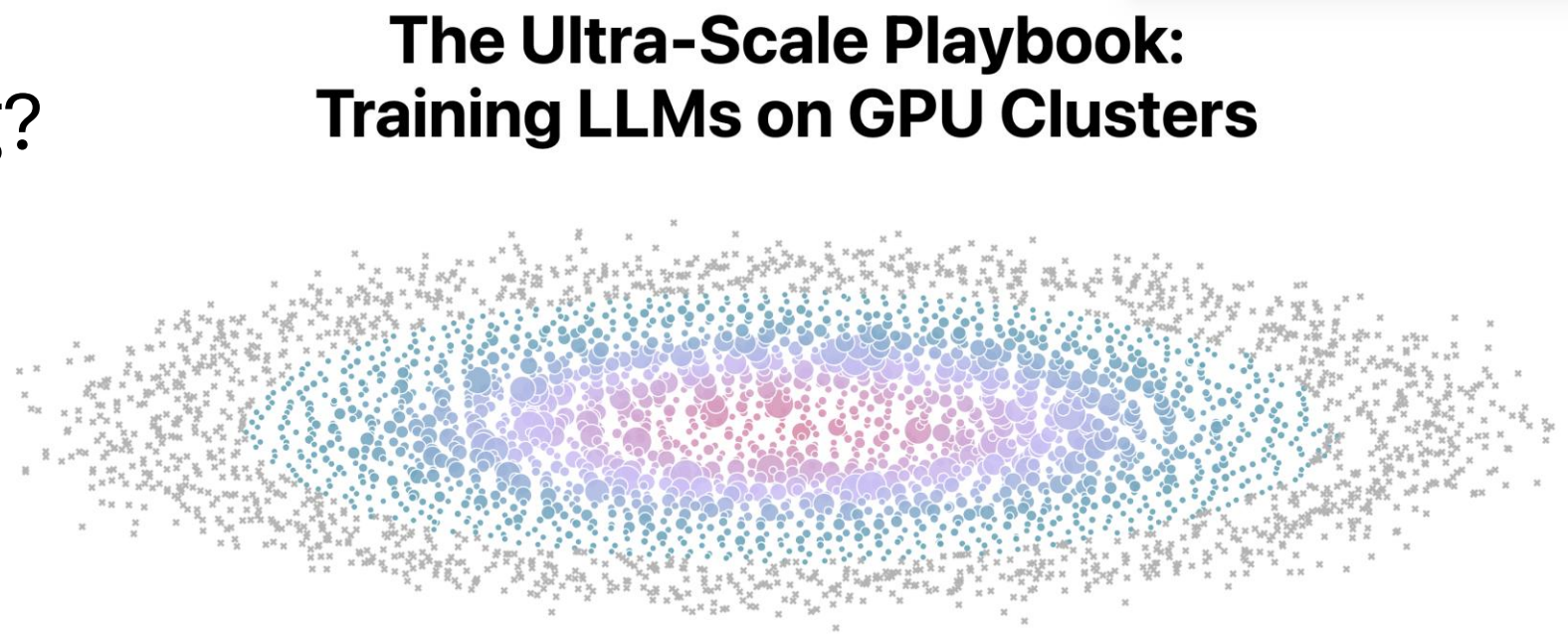


- Based on Nvidia A100 80GB GPU

https://huggingface.co/spaces/optimum/llm-perf-leaderboard

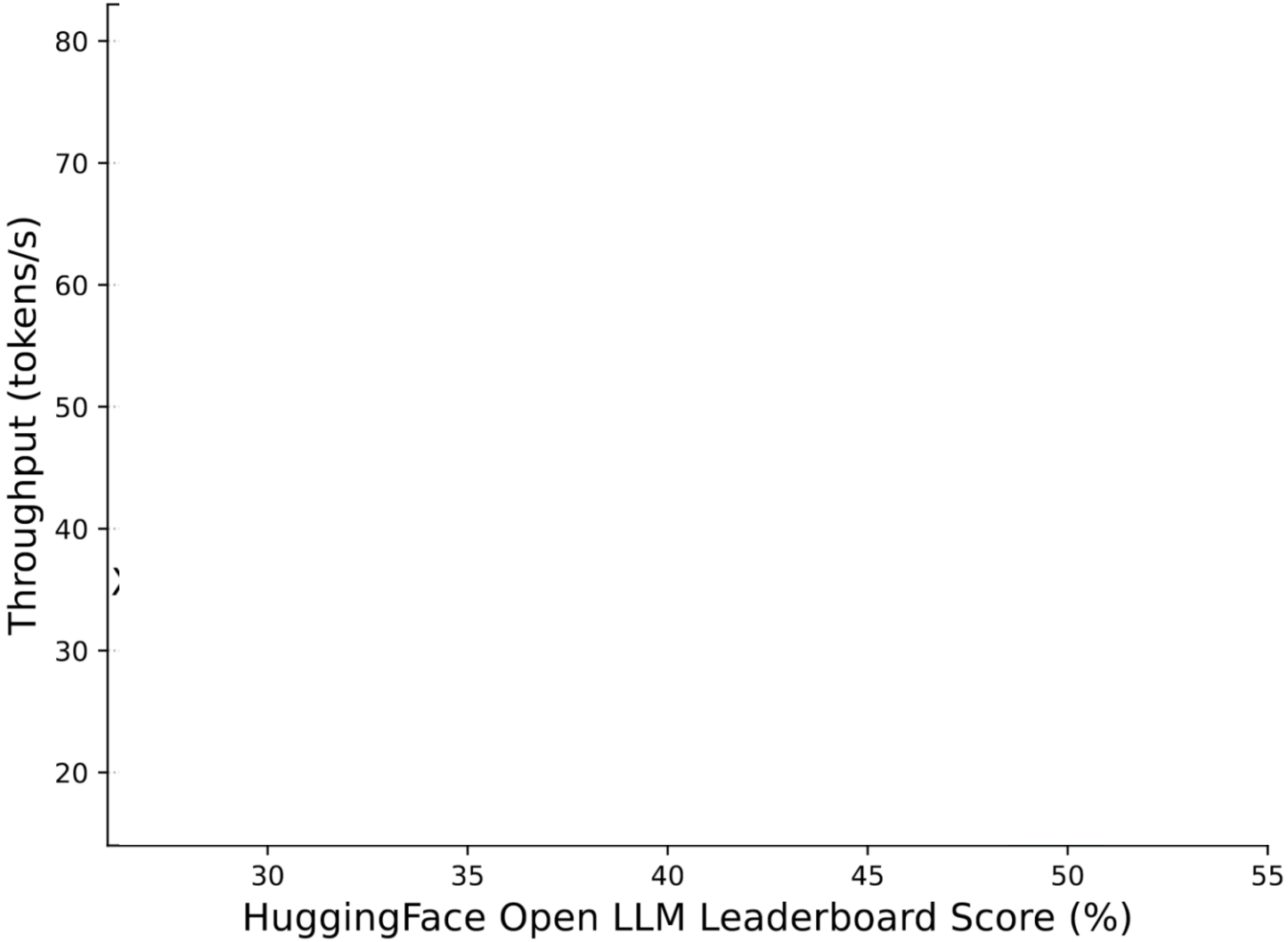
Efficient LLMs

- How to scale training?
  - *Data Parallelism*
  - *Tensor Parallelism*
  - *Context Parallelism*
  - *Pipeline Parallelism*

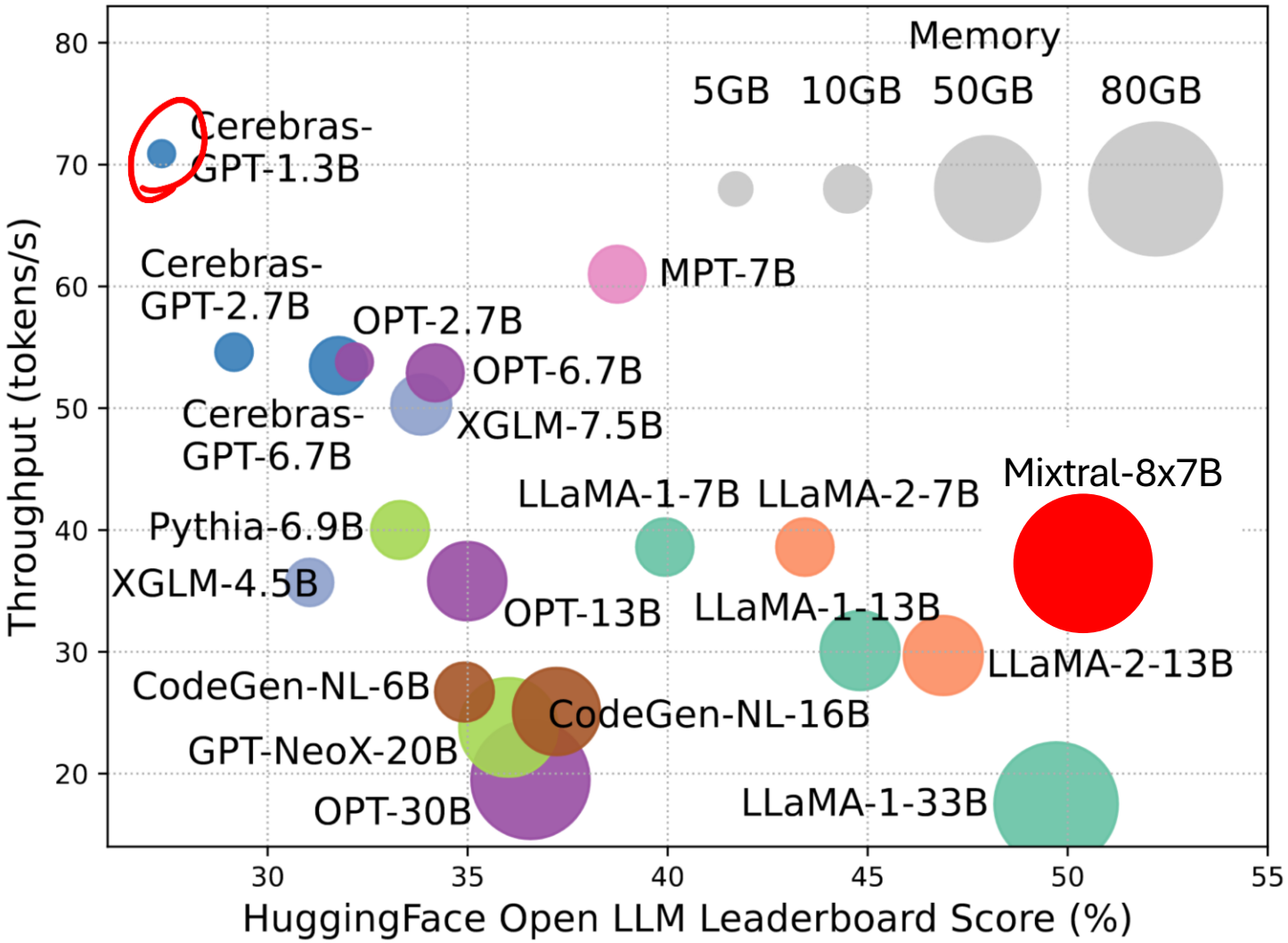


We ran over 4,000 scaling experiments on up to 512 GPUs and measured throughput (size of markers) and GPU utilization (color of markers). Note that both are normalized per model size in this visualization.

Inference Throughput vs Performance

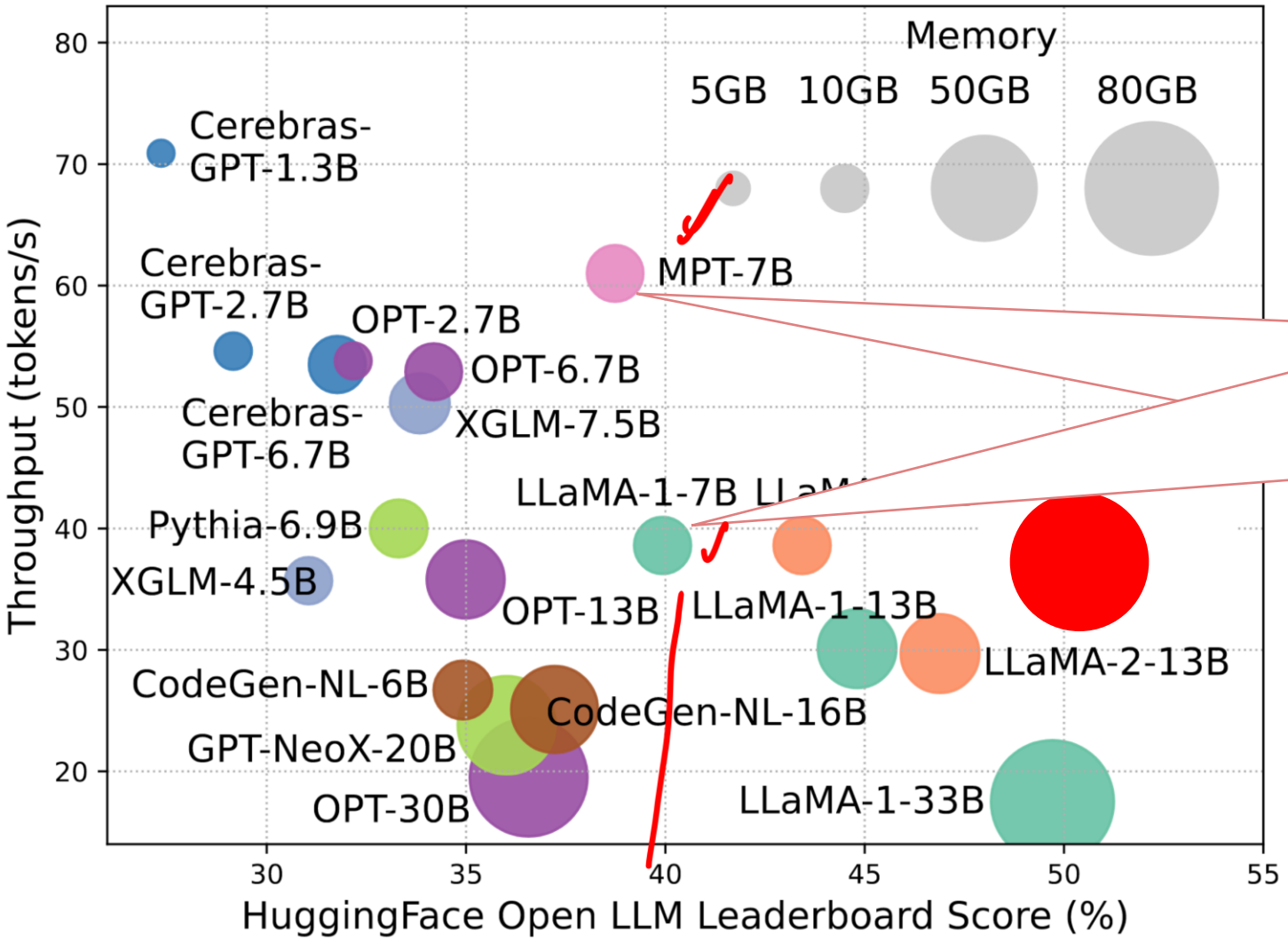


Inference Throughput vs Performance



- On Nvidia A100 80GB GPU;
- 16-bit quantized
- Batch Size - 1
- Prompt size of 256
- Generating 1000 tokens

Inference Throughput vs Performance



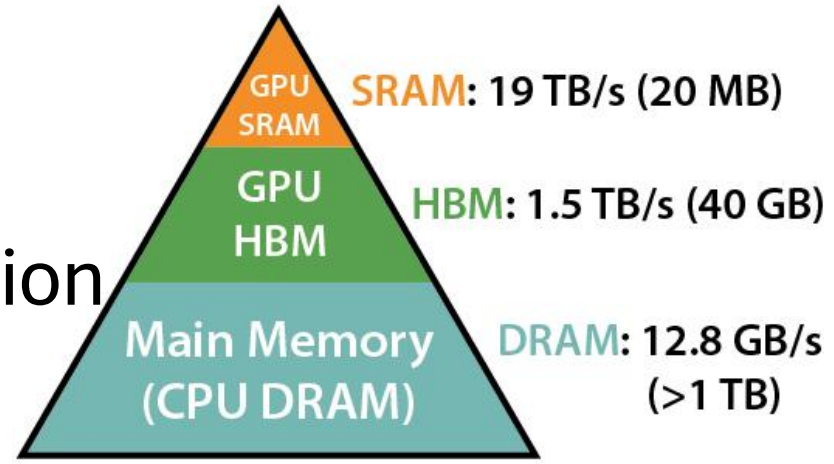
- Similar performance, different throughput! How?
- Efficient implementation –
  - Fused kernel for attention

Efficient LLMs

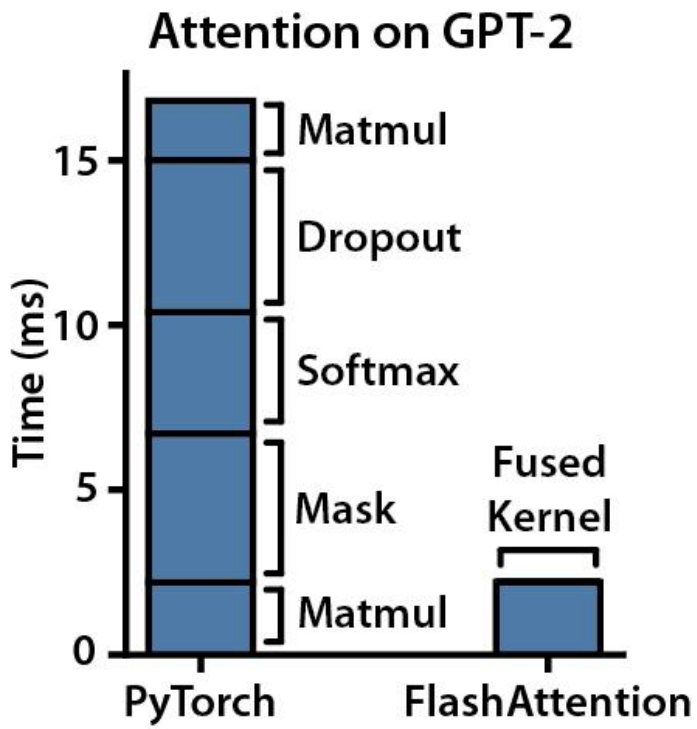
- How to scale training?

  Parallelism …

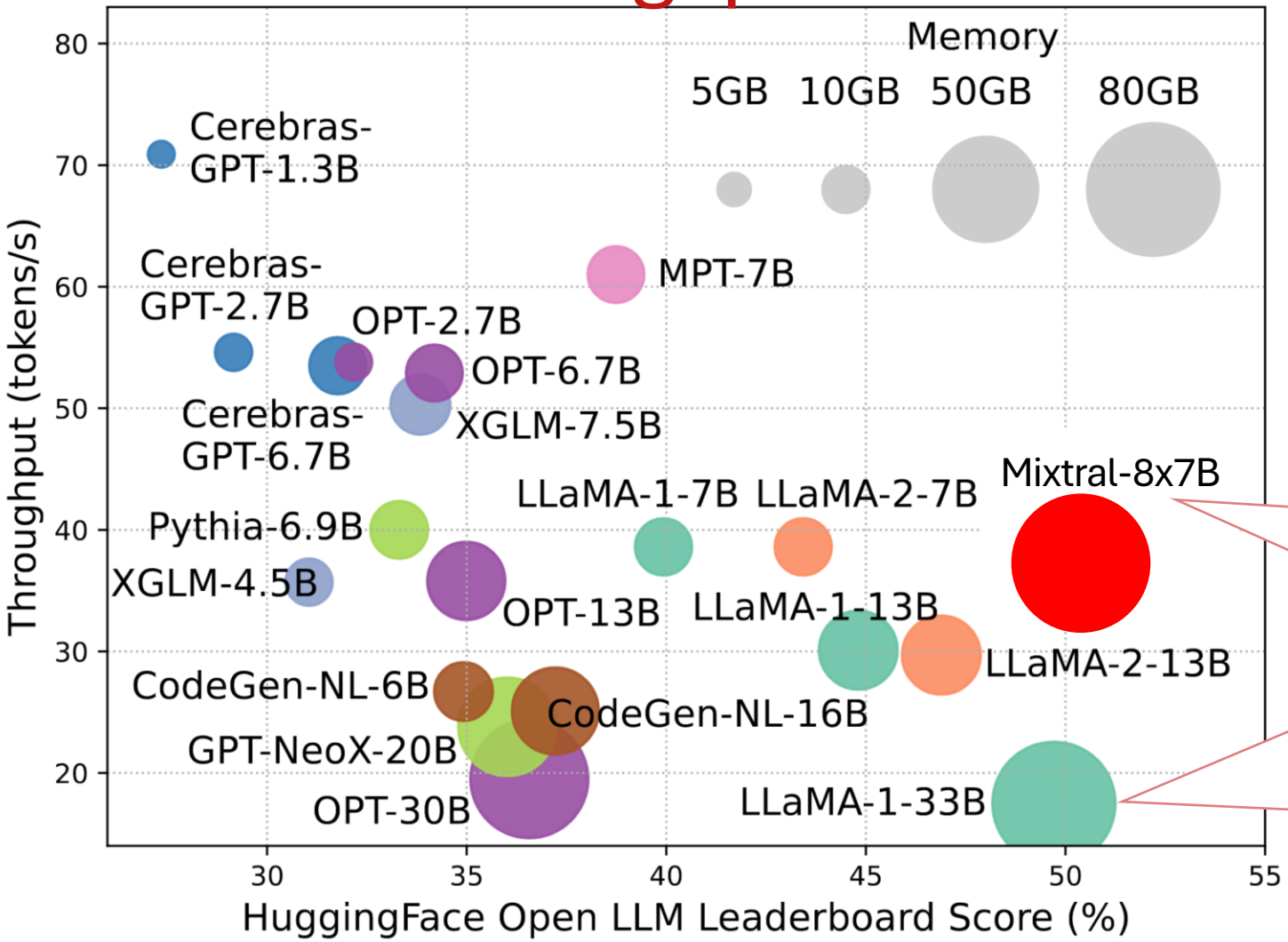
- Efficient implementation
  - Flash Attention
  - Paged Attention



Memory Hierarchy with Bandwidth & Memory Size



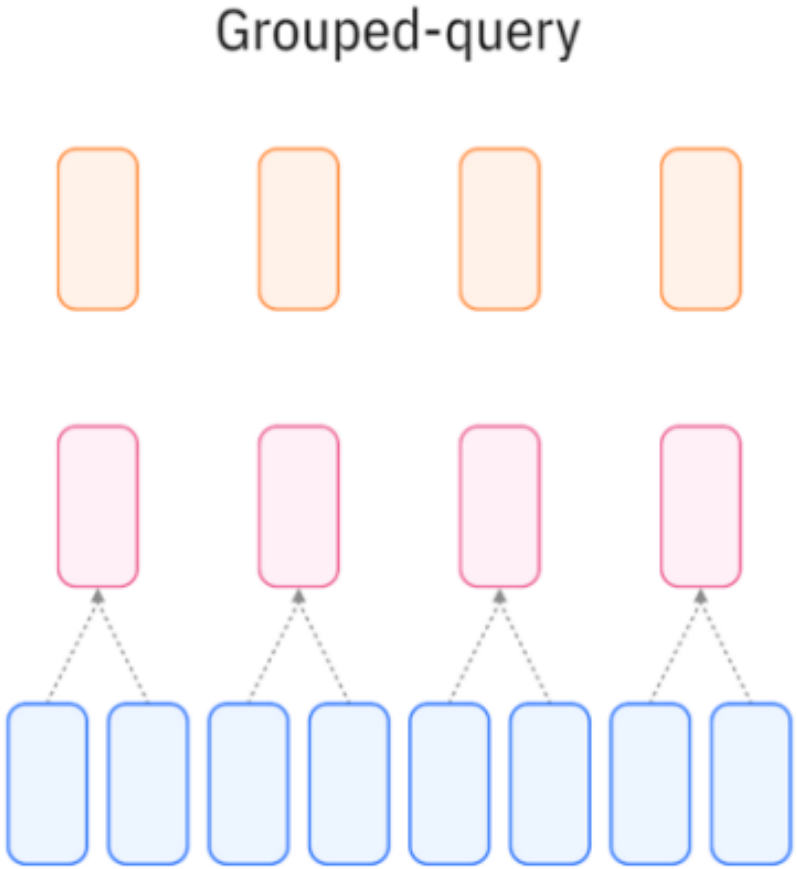
Inference Throughput vs Performance



- Similar performance, different throughput! How?
- Efficient design –
  - Grouped Query Attention
  - MoE

Efficient LLMs

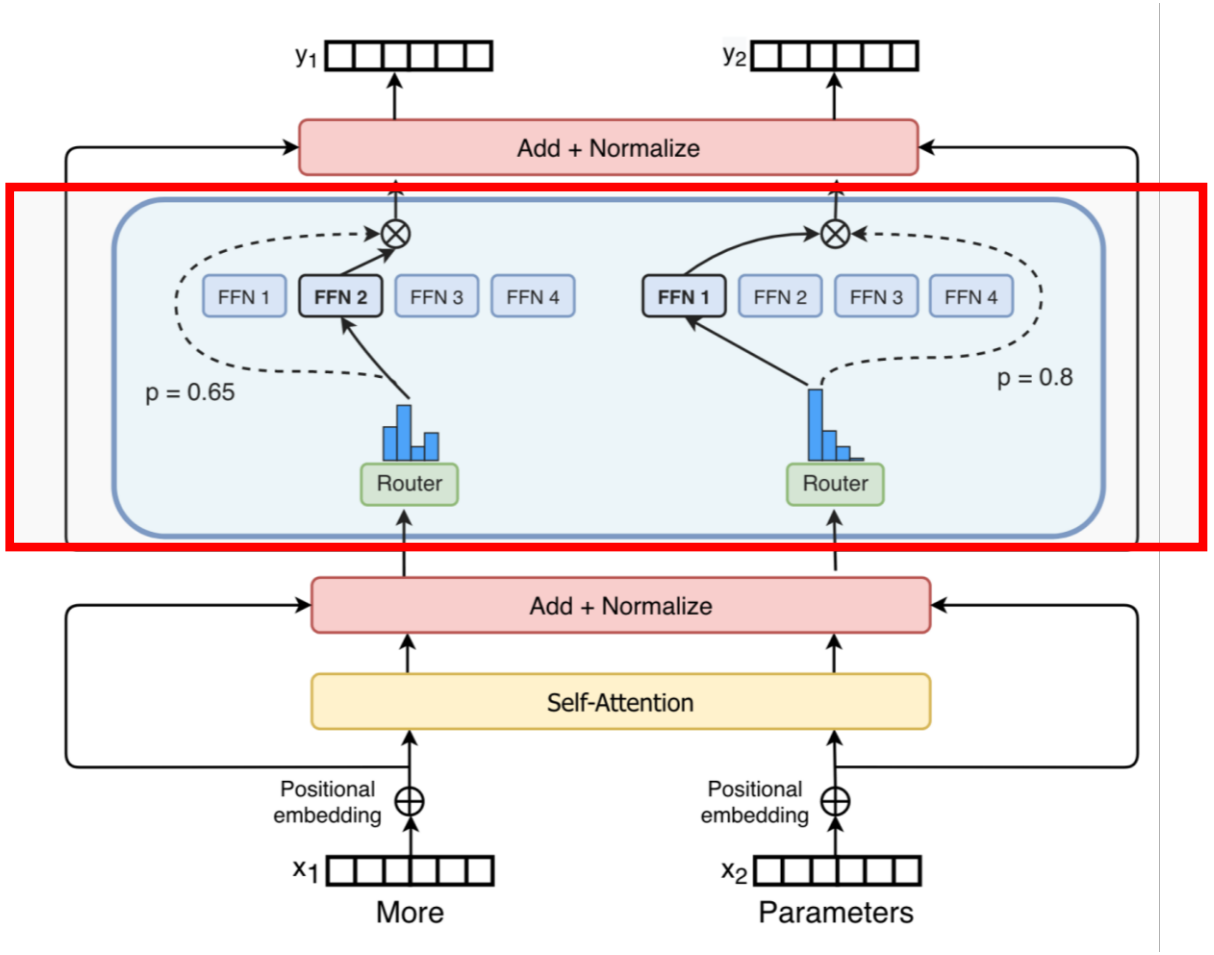
- How to scale training?
  Parallelism …
- Efficient implementation
  - *Fused kernels …*
- Efficient design
  - Grouped Query Attention
  - Mixture of Experts



Efficient LLMs

- How to scale training?
  Parallelism …
- Efficient implementation
  - *Fused kernels …*
- Efficient design
  - Grouped Query Attention
  - Mixture of Experts

Mixture of Experts



Efficient LLMs

- How to scale training?

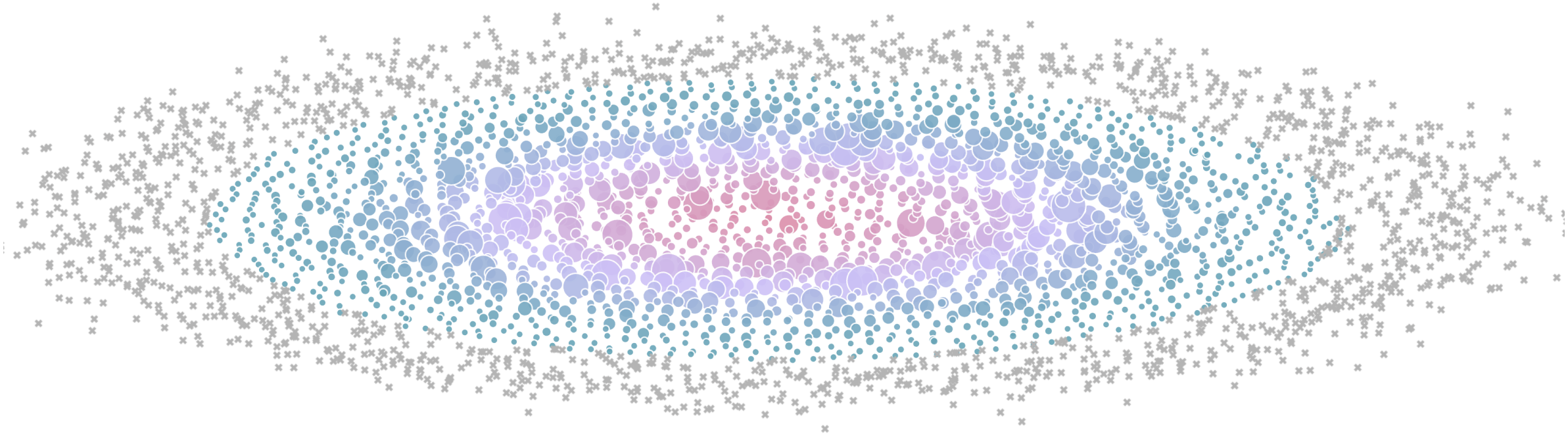
  Parallelism …

- Efficient implementation
  - *Fused kernels …*

- Efficient design
  - *Grouped Query Attention*
  - *Mixture of Experts*

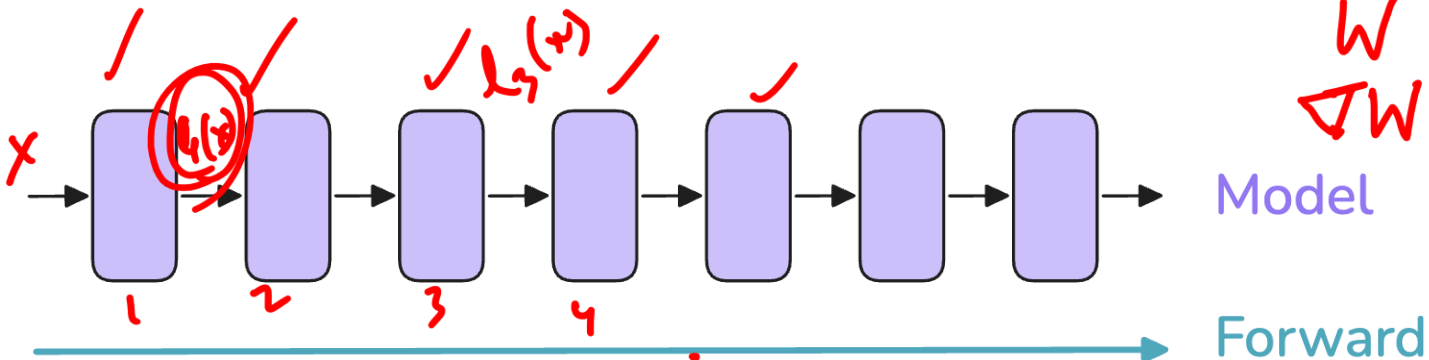
The Ultra-Scale Playbook: Training LLMs on GPU Clusters

https://huggingface.co/spaces/nanotron/ultrascale-playbook



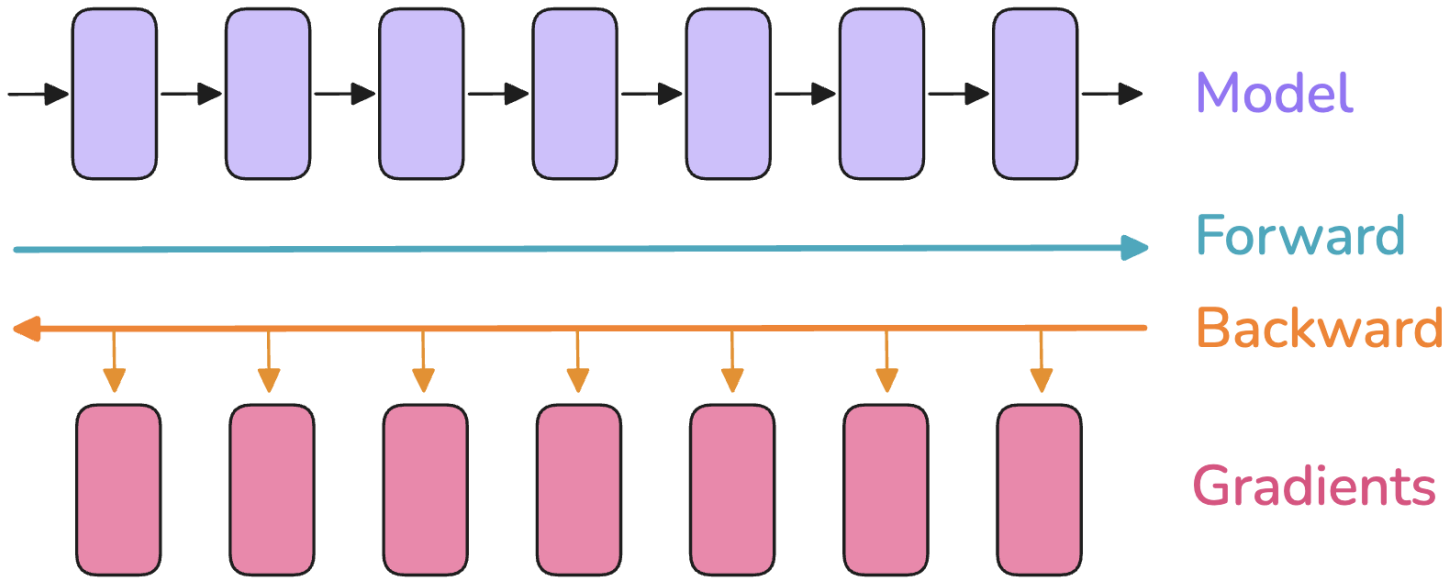
Over 4,000 scaling experiments on up to 512 GPUs and measured throughput (size of markers) and GPU utilization (color of markers) - Normalized per model size in this visualization.

First Steps: Training on One GPU



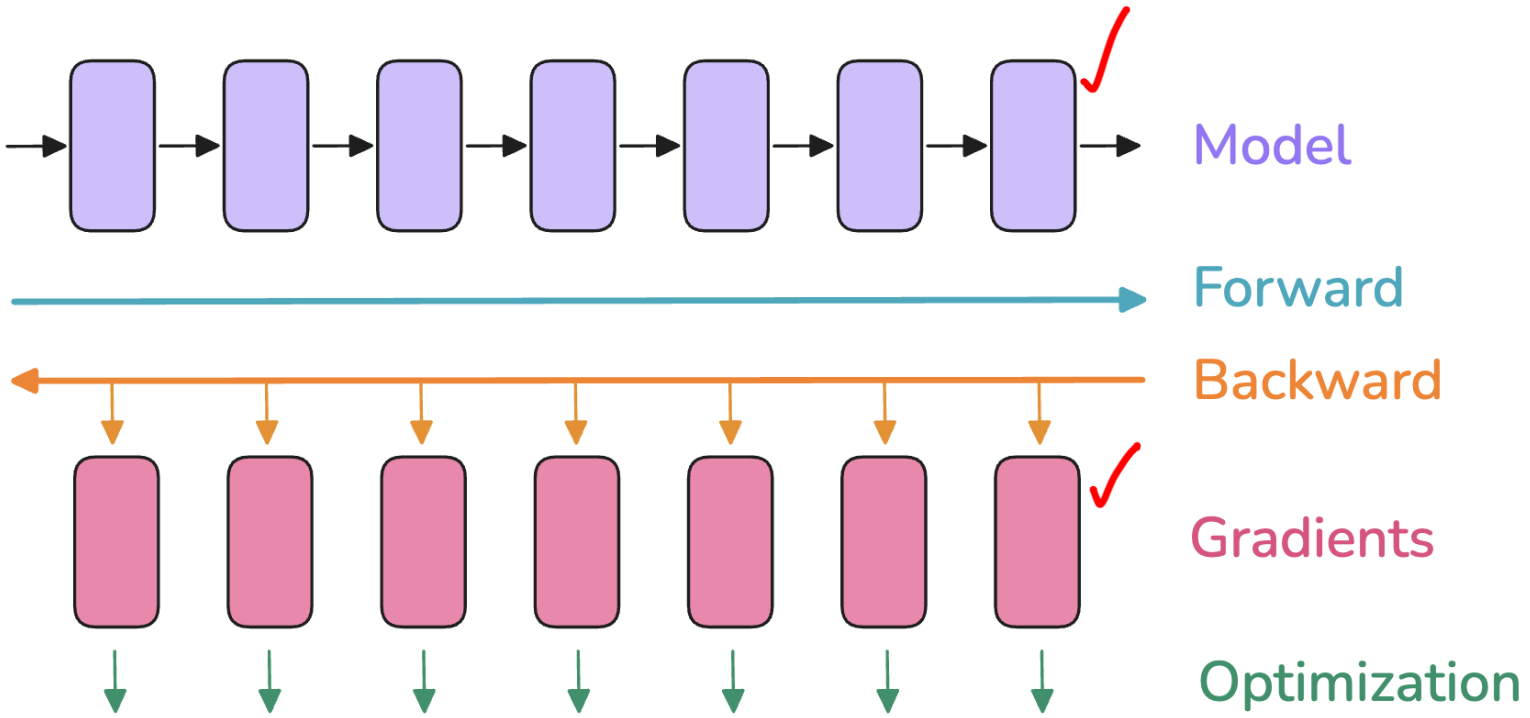
1. **Forward Pass:** pass inputs through the model to yield its outputs and compute Loss

First Steps: Training on One GPU



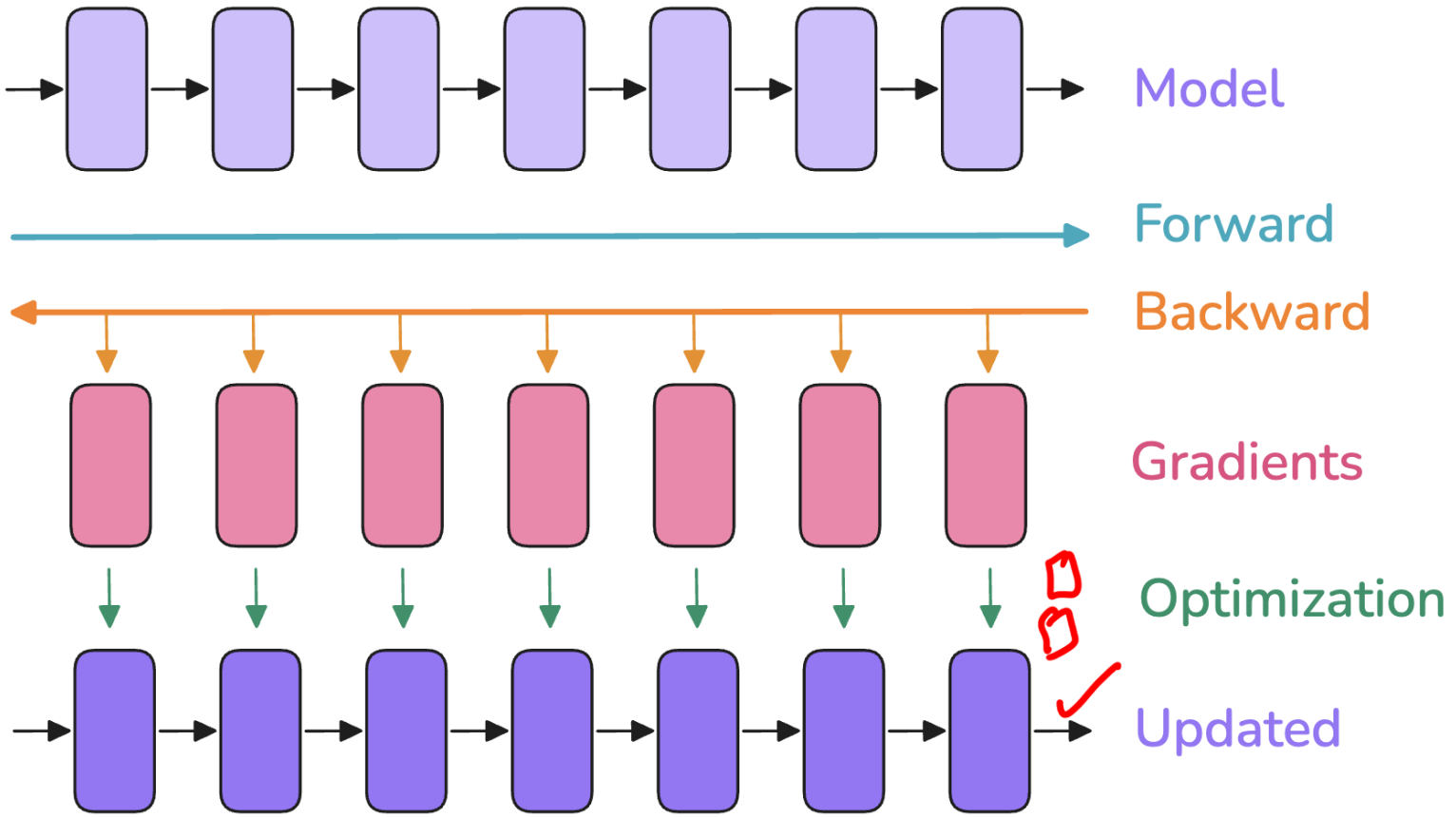
1. **Forward Pass:** pass inputs through the model to yield its outputs and compute Loss
2. **Backward Pass:** Compute the gradients

First Steps: Training on One GPU



1. **Forward Pass:** pass inputs through the model to yield its outputs and compute Loss
2. **Backward Pass:** Compute the gradients
3. **Optimization Step:** Update the parameters using the gradients

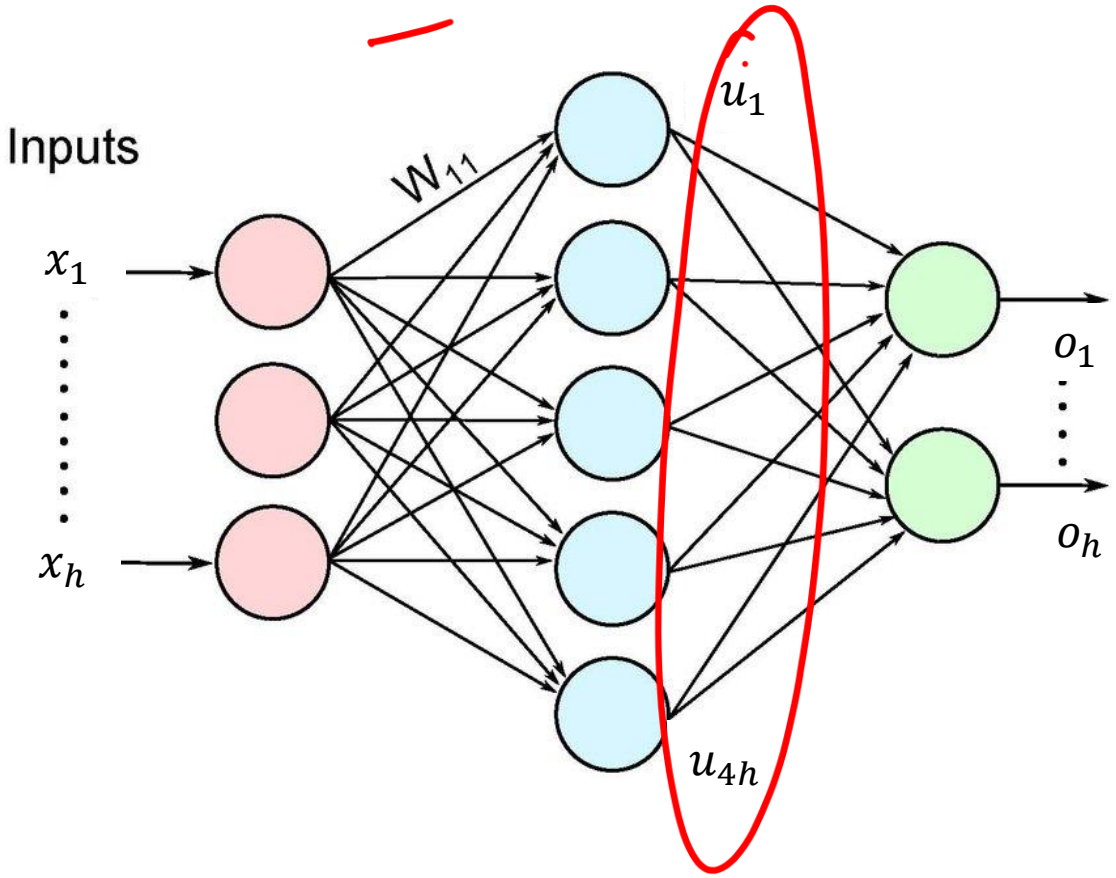
First Steps: Training on One GPU



1. **Forward Pass:** pass inputs through the model to yield its outputs and compute Loss
2. **Backward Pass:** Compute the gradients
3. **Optimization Step:** Update the parameters using the gradients

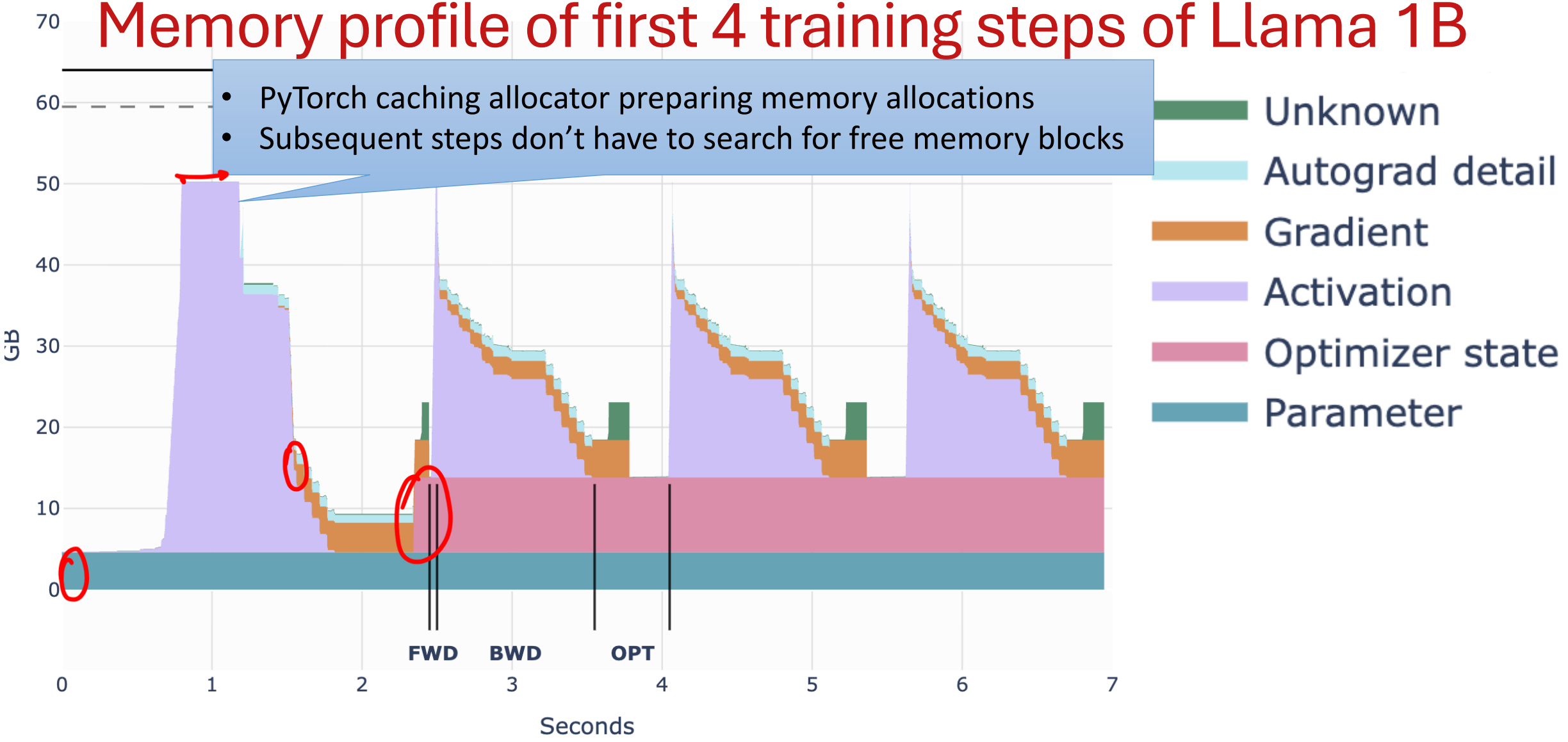
First Steps: Training on One GPU ?

- **Memory usage in transformers**
  - Model weights
  - Model gradients
  - Optimizer states
  - Activations needed to compute the gradients

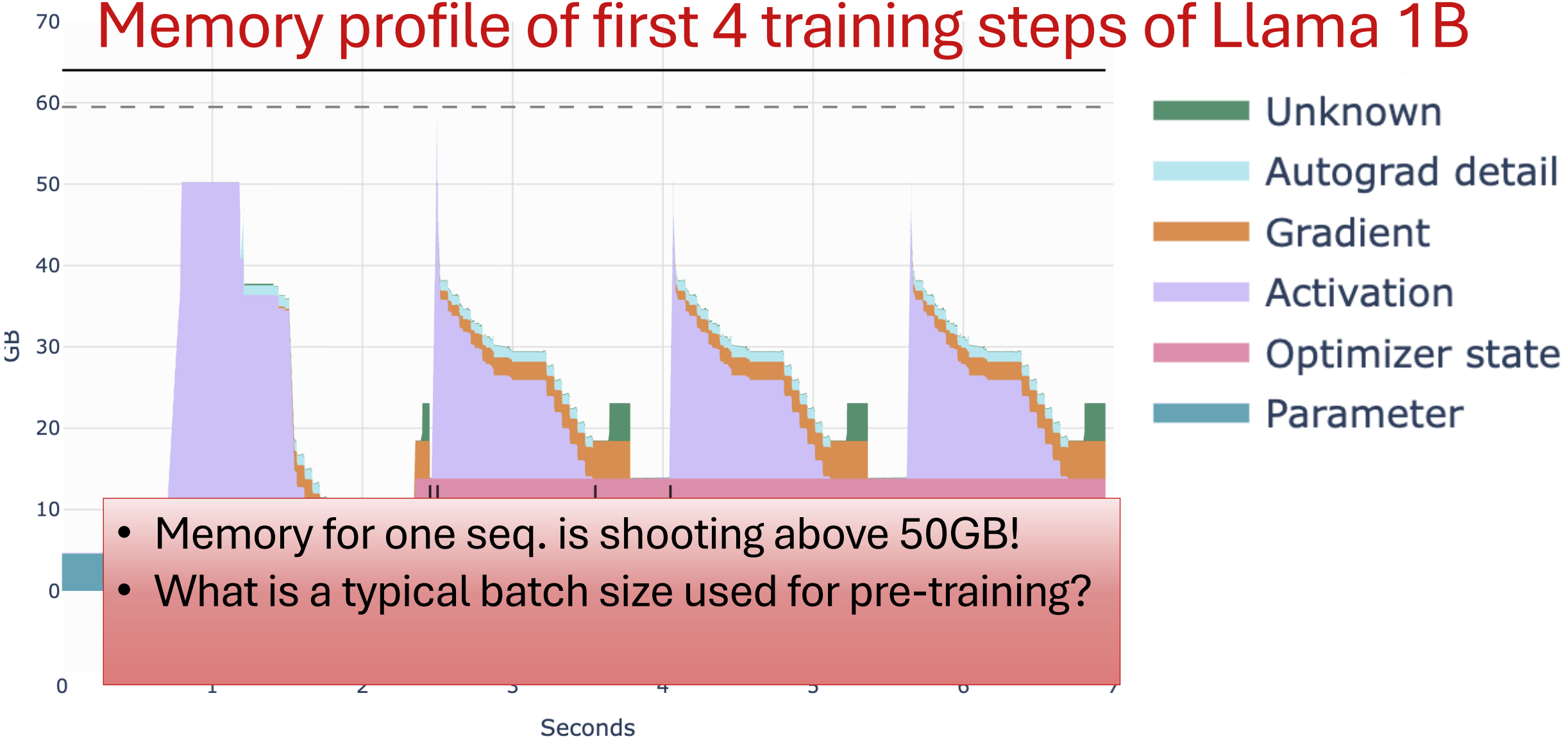


Memory profile of first 4 training steps of Llama 1B

- PyTorch caching allocator preparing memory allocations
- Subsequent steps don't have to search for free memory blocks



Memory profile of first 4 training steps of Llama 1B



- Memory for one seq. is shooting above 50GB!
- What is a typical batch size used for pre-training?

Batch Size

- Reported in ***#Tokens*** = Batch Size Tokens = *bst* = *bs* * *seq*

  Llama1 – batch size ~ 4M for 1.4T tokens; DeepSeek - batch size ~ 60M for 14T tokens

- Affects both model convergence and throughput

Small batch size:

- 👎 Noisy gradient estimates
- 👍 Quick to compute => each step is fast
- 👎 More optimizer steps for fixed data
- 👎 May evade optima due to noisy gradients

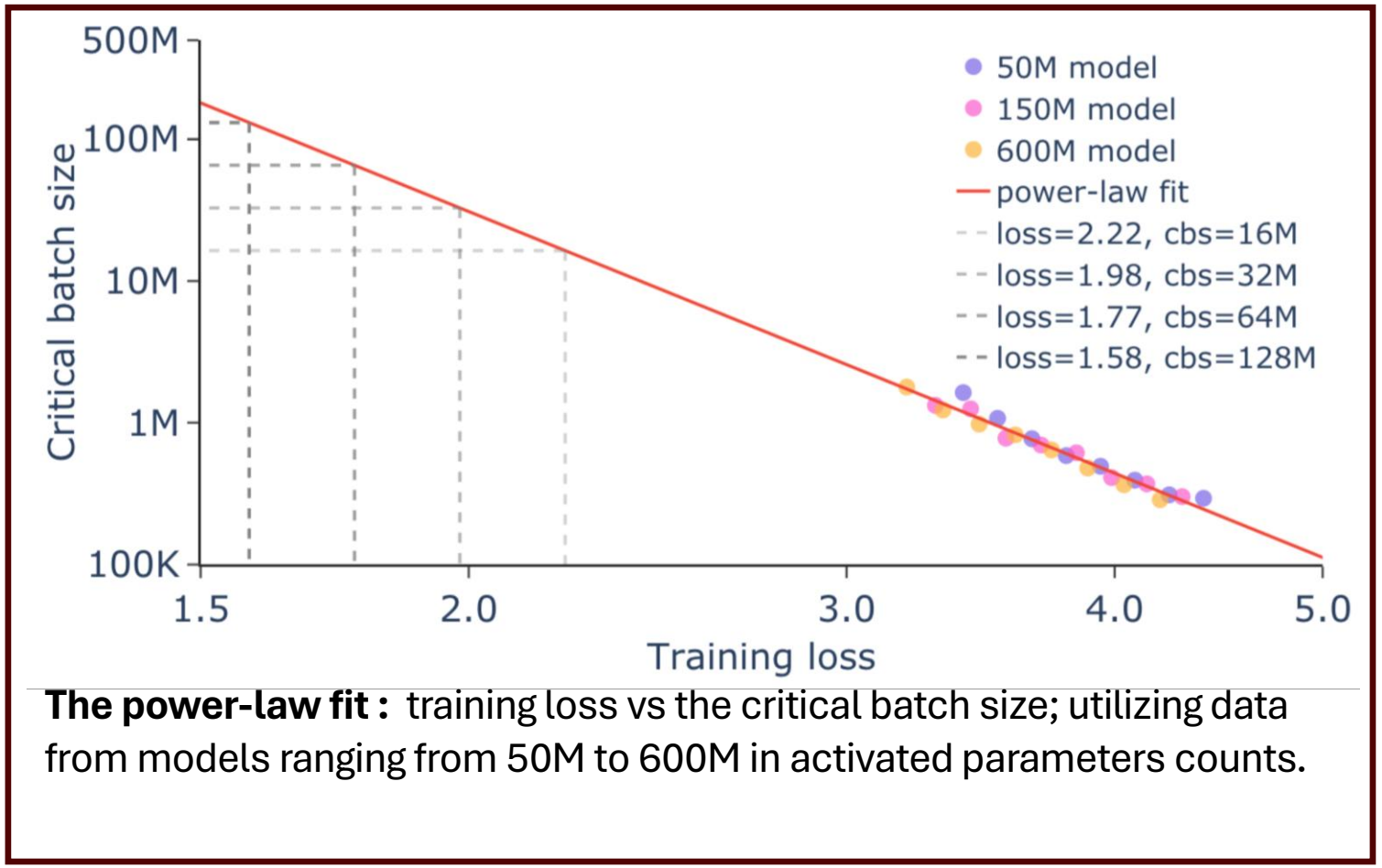
Large batch size:

- 👍 Stable gradient estimates
- 👎 More time to compute => each step is slow
- 👍 Less optimizer steps for fixed data
- 👍 Arrive at better optima due to stable gradients

Batch Size

- *Training at the **critical batch size** yields a near-optimal balance between training time and data efficiency*

McCandlish et al. 2018, An Empirical Model of Large-Batch Training



The power-law fit : training loss vs the critical batch size; utilizing data from models ranging from 50M to 600M in activated parameters counts.

Batch Size: 16M → 32M → 64M → 128M

 # Tokens Seen: 0 → 69B → 790B → 4700B

Gradient update steps →

Batch Size

Towards the beginning...

Small Batch Size

- Quickly move through the loss landscape (multiple noisy gradient update steps)

Large Batch Size

- Accurate gradient, but slower convergence, potentially wasting compute

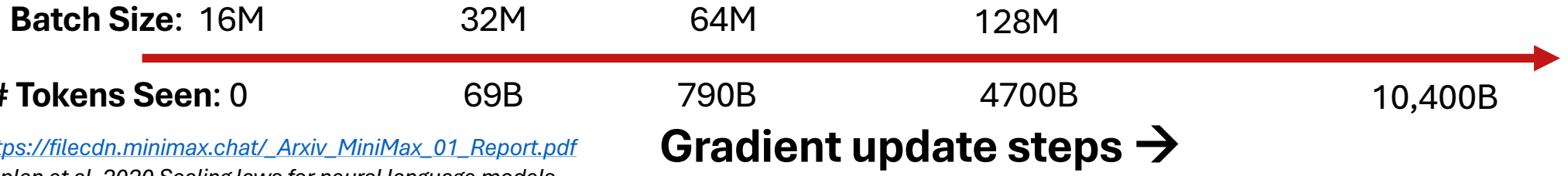
Towards the end...

Small Batch Size

Model may not converge to the most optimal performance (due to noisy gradients)

Large Batch Size

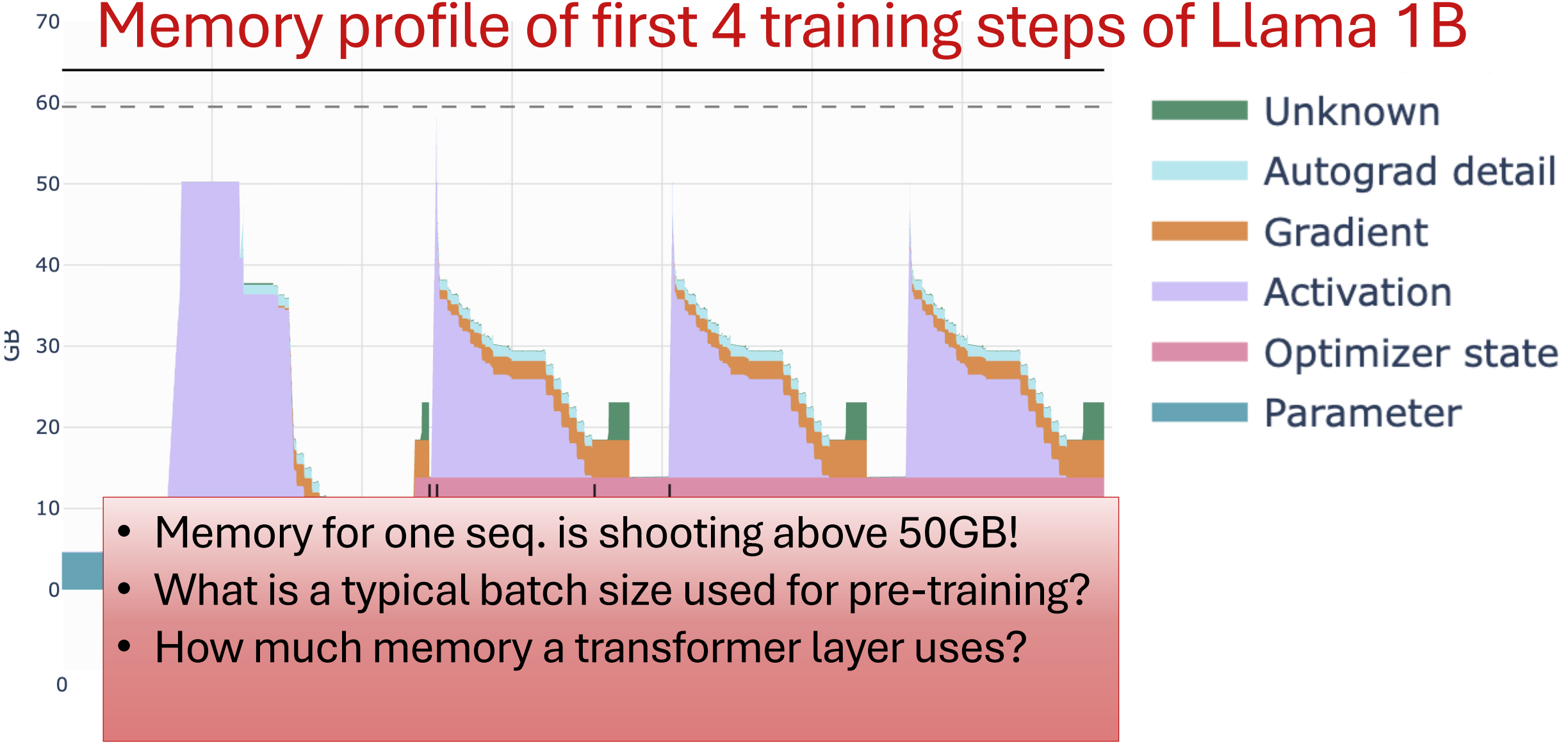
Accurate gradient, helps in reaching optimal performance



https://filecdn.minimax.chat/_Arxiv_MiniMax_01_Report.pdf

Kaplan et al. 2020 Scaling laws for neural language models.

Memory profile of first 4 training steps of Llama 1B

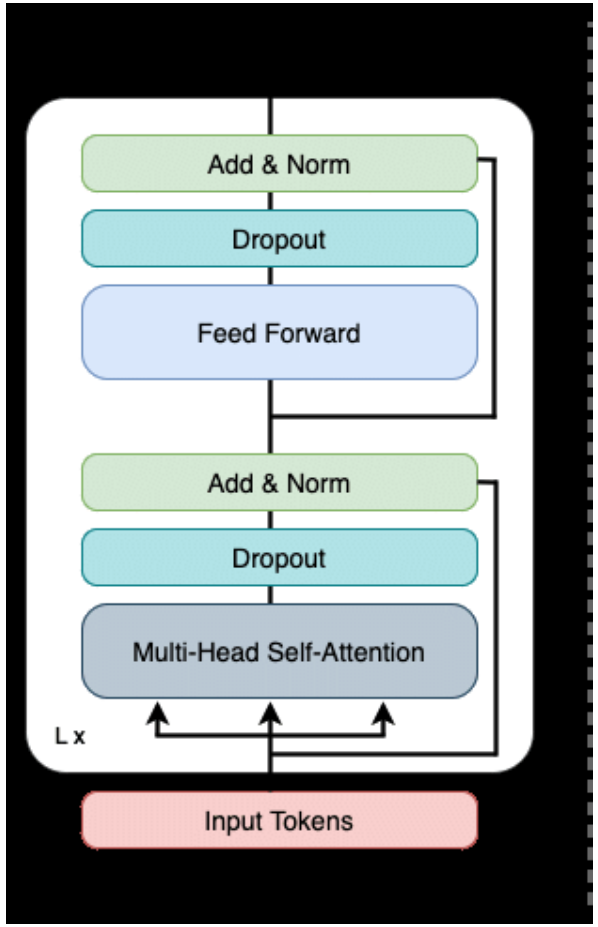


- Memory for one seq. is shooting above 50GB!
- What is a typical batch size used for pre-training?
- How much memory a transformer layer uses?

Memory for Weights, Grads, and Optim States

- h : hidden dimension,
- v : vocabulary size,
- L : number of layers.

Total	
Layer Norm	$2h$
MLP	$(h+4h+4h)+4h^2+h$
Layer Norm	$2h$
Projection	h^2+h
MHA	$3(h^2+h)$
Token Emb.	hv



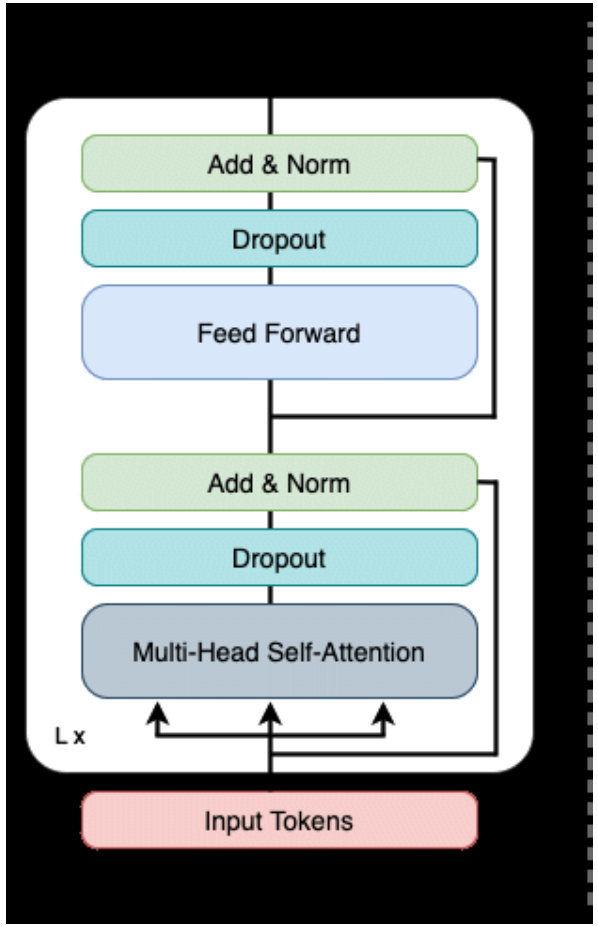
https://michaelwornow.net/2024/01/18/counting-params-in-transformer

Memory for Weights, Grads, and Optim States

- $N = h * v + L * (12 * h^2 + 13 * h) + 2 * h$

h : hidden dimension,
v : vocabulary size,
L : number of layers.

Total	
Layer Norm	$2 * h$
MLP	$h * 4h + 4h + 4h * h + h$
Layer Norm	$2 * h$
Projection	$h * h + h$
MHA	$3 * (h * h + h)$
Token Emb.	$h * v$



https://michaelwornow.net/2024/01/18/counting-params-in-transformer

Memory for Weights, Grads, and Optim States

- $N = h * v + L * (12 * h^2 + 13 * h) + 2 * h$

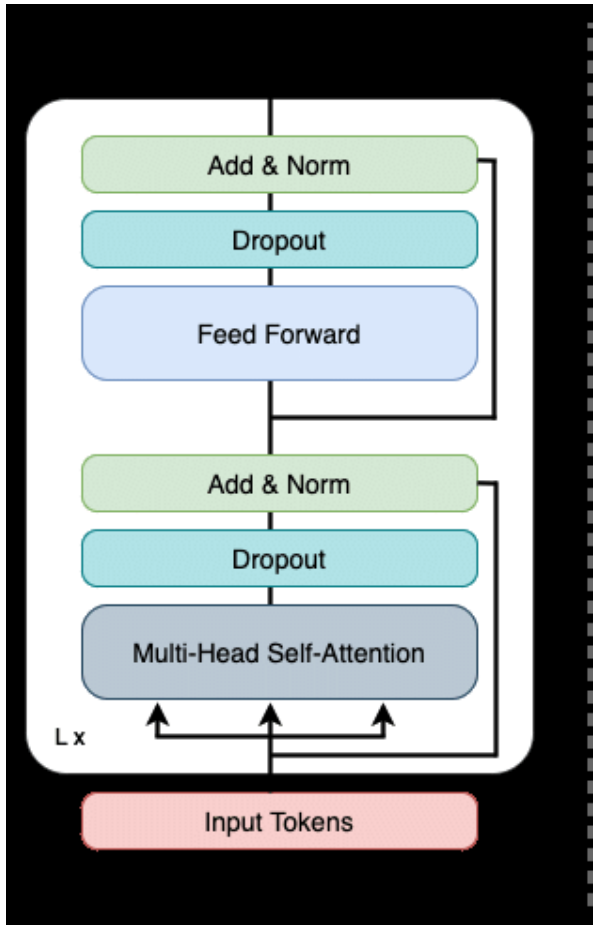
- FP32 representation (4 bytes per param)
  - $m_{params} = 4 * N$
  - $m_{grad} = 4 * N$
  - $m_{opt} = (4 + 4) * N$

Total: $16 * N\ bytes$

- Mixed precision: computation in bf16 (2 bytes); storage in FP32
  - $m_{params} = 2 * N$
  - $m_{grad} = 2 * N$
  - $m_{opt} = (4 + 4) * N$
  - $m_{params\_fp32} = 4 * N$ (master weights)

Total: $16 * N\ bytes$

Why use mixed precision if total memory is same?



Memory for Weights, Grads, and Optim States

- $N = h * v + L * (12 * h^2 + 13 * h) + 2 * h$

- FP32 representation (4 bytes per param)
  - $m_{params} = 4 * N$
  - $m_{grad} = 4 * N$
  - $m_{opt} = (4 + 4) * N$

Total: $16 * N\ bytes$

- Mixed precision: computation in bf16 (2 bytes); storage in FP32
  - $m_{params} = 2 * N$
  - $m_{grad} = 2 * N$
  - $m_{opt} = (4 + 4) * N$
  - $m_{params\_fp32} = 4 * N$ (master weights)

Total: $16 * N\ bytes$

Why use mixed precision if total memory is same?

1. Allows us to use optimized lower precision operations on the GPU, which are faster
2. Reduces the activation memory requirements during the forward pass

Memory for Weights, Grads, and Optim States

- $N = h * v + L * (12 * h^2 + 13 * h) + 2 * h$

- FP32 representation (4 bytes per param)
  - $m_{params} = 4 * N$
  - $m_{grad} = 4 * N$
  - $m_{opt} = (4 + 4) * N$

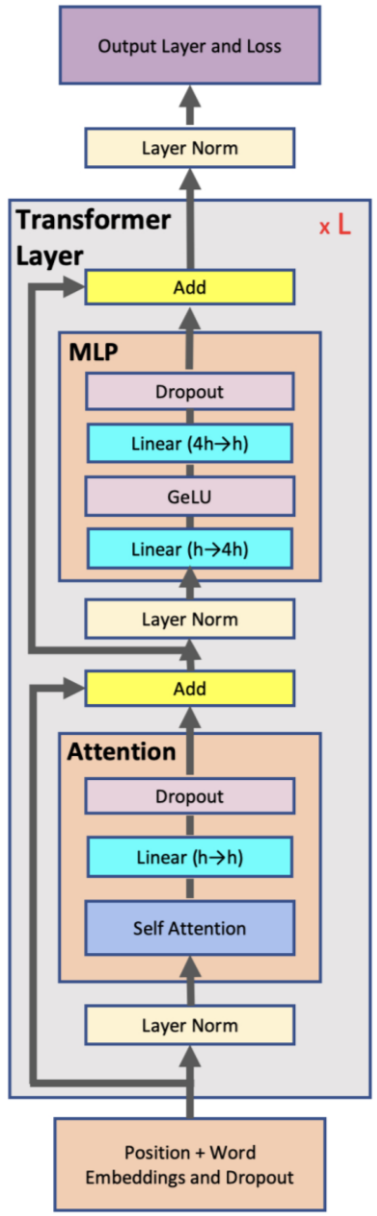
Total: $16 * N\ bytes$

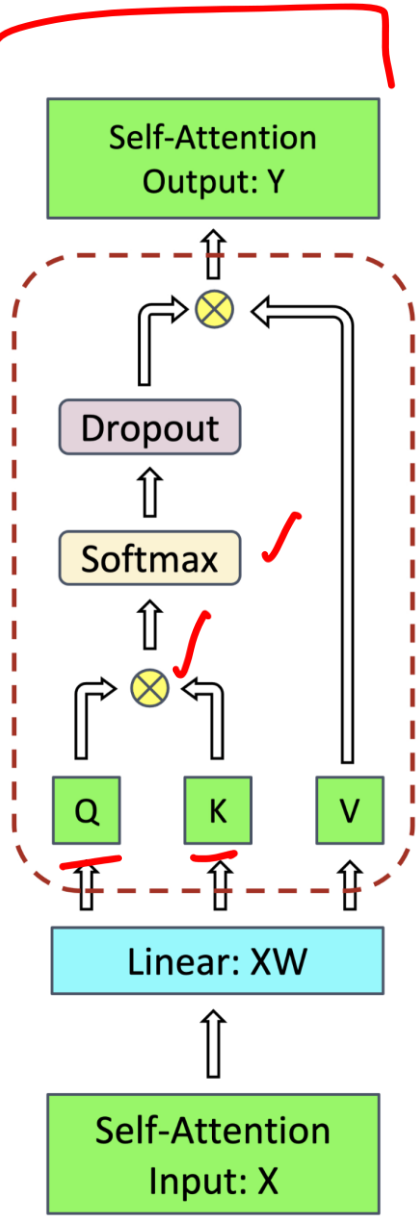
- Mixed precision: computation in bf16 (2 bytes); storage in FP32
  - $m_{params} = 2 * N$
  - $m_{grad} = 2 * N$
  - $m_{opt} = (4 + 4) * N$
  - $m_{params\_fp32} = 4 * N$ (master weights)

Total: $16 * N\ bytes$

Why use mixed precision if total memory is same?

Model parameters	Total Mem.
1B	16 GB
7B	112 GB
70B	1120 GB
405B	6480 GB

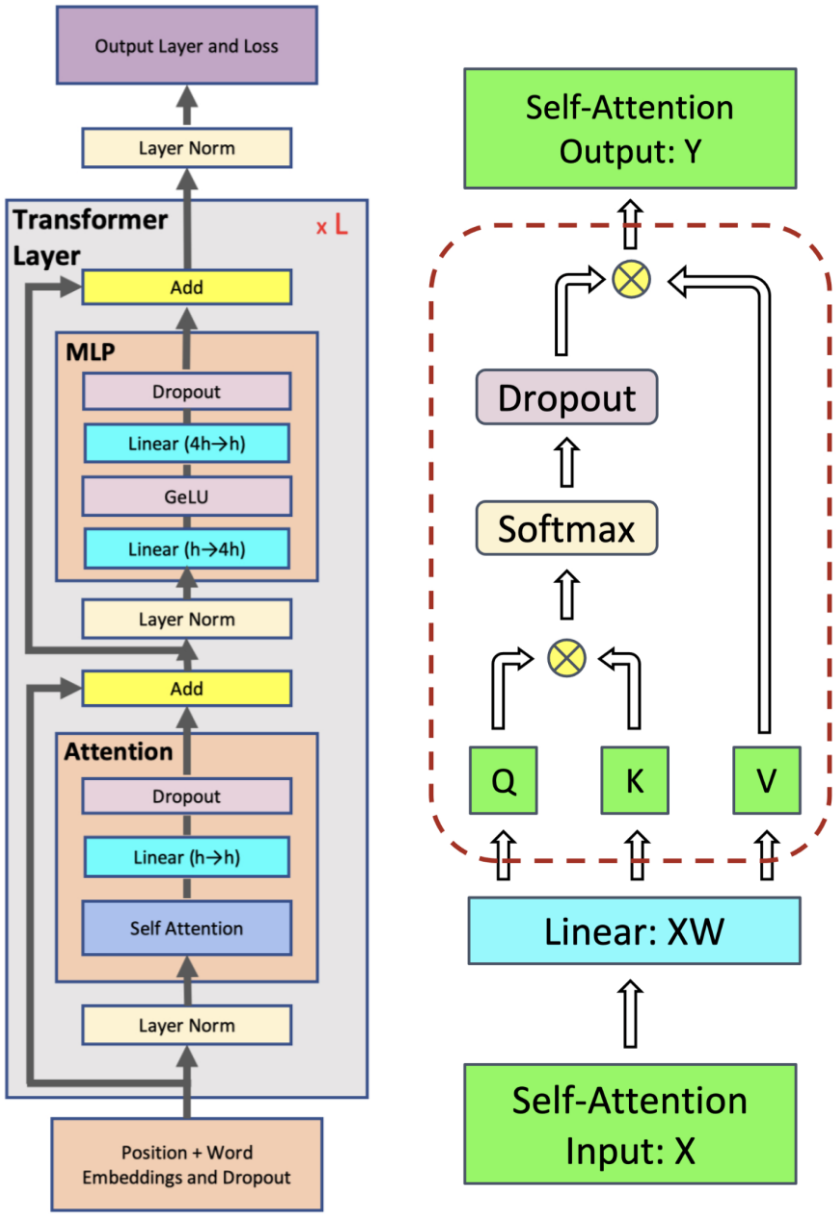




Memory for Activations

MHA	
Q, K, V input	$bs \times seq \times h$
QK^T	$O(bsh)$
Softmax	$O(n_{heads} seq^2 bs)$
Softmax d/o mask	
Prob*Values	

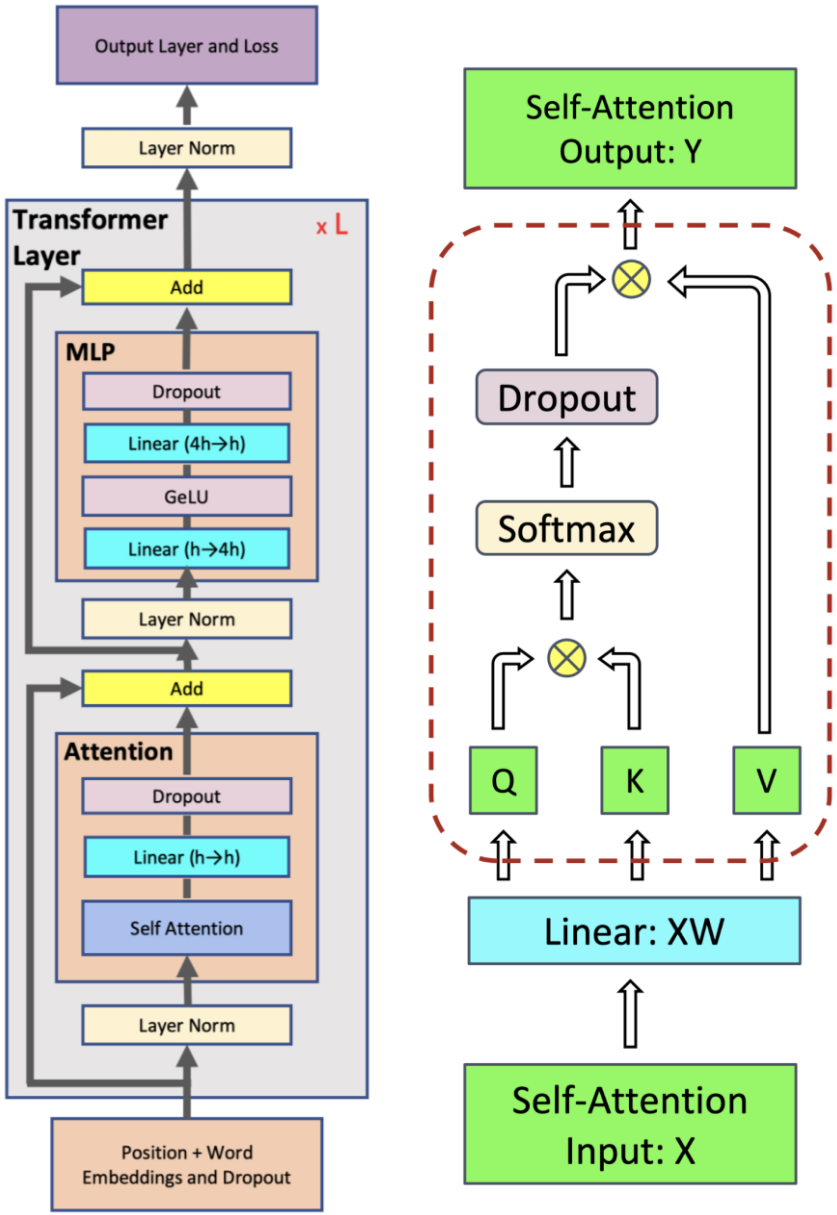
[Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models]



Memory for Activations

Attention Block	
Dropout mask	
Linear Proj:	
MHA	$8 * seq * bs * h + 5 * n_{heads} * seq^2 * bs$
Q, K, V **input**	$2 * seq * bs * h$
QK^T	$4 * seq * bs * h$
Softmax	$2 * n_{heads} * bs * seq * seq$
Softmax d/o mask	$n_{heads} * bs * seq * seq$
Prob*Values	$2 * n_{heads} * bs * seq * seq + 2 * seq * bs * h$

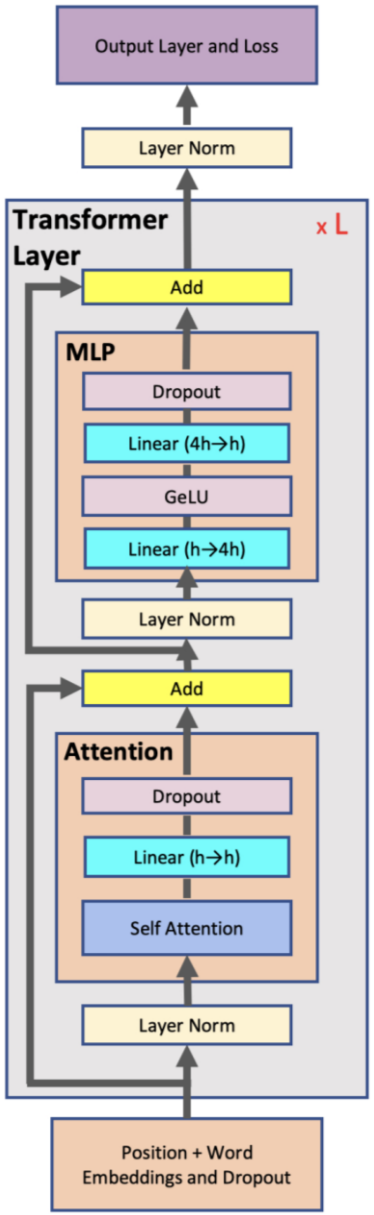
Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models



Memory for Activations

Attention Block	$11 * seq * bs * h + 5 * n_{heads} * seq^2 * bs$
Dropout mask	$seq * bs * h$
Linear Proj:	$2 * seq * bs * h$
MHA	$8 * seq * bs * h + 5 * n_{heads} * seq^2 * bs$
Q, K, V **input**	$2 * seq * bs * h$
QK^T	$4 * seq * bs * h$
Softmax	$2 * n_{heads} * bs * seq * seq$
Softmax d/o mask	$n_{heads} * bs * seq * seq$
Prob*Values	$2 * n_{heads} * bs * seq * seq + 2 * seq * bs * h$

Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models

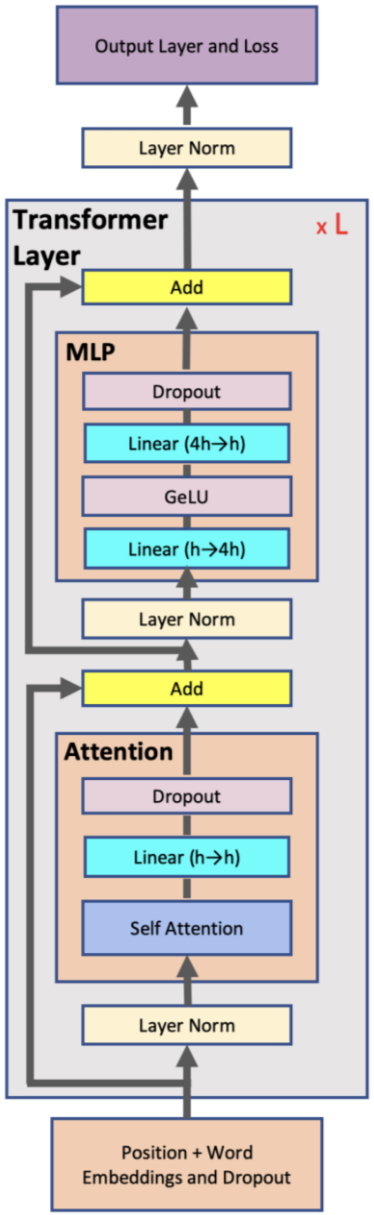


MLP Block	
D/o mask	
Linear *(4h -> h)*	
GELU	
Linear *(h -> 4h)*	
Layer Norm	
Attention Block	$11 * seq * bs * h + 5 * n_{heads} * seq^2 * bs$
Layer Norm	

Memory for Activations

Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models

Memory for Activations



Total	$\mathbf{34 * seq * bs * h + 5 * n_{heads} * seq^2 * bs}$
MLP Block	$\mathbf{19 * seq * bs * h}$
D/o mask	$1 * seq * bs * h$
Linear *(4h -> h)*	$8 * seq * bs * h$
GELU	$8 * seq * bs * h$
Linear *(h -> 4h)*	$2 * seq * bs * h$
Layer Norm	$\mathbf{2 * seq * bs * h}$
Attention Block	$\mathbf{11 * seq * bs * h + 5 * n_{heads} * seq^2 * bs}$
Layer Norm	$\mathbf{2 * seq * bs * h}$

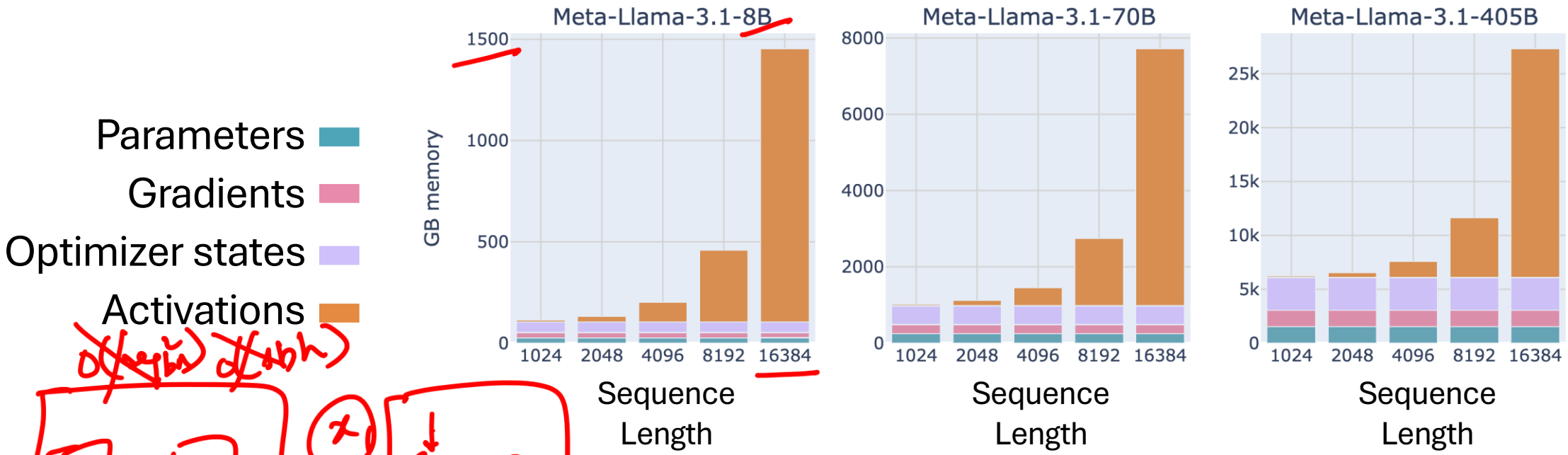
$$m_{act} = L * \left(34 * seq * bs * h + 5 * n_{heads} * seq^2 * bs \right)$$

- Scales linearly with batch size
- Quadratically with the sequence length

Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models

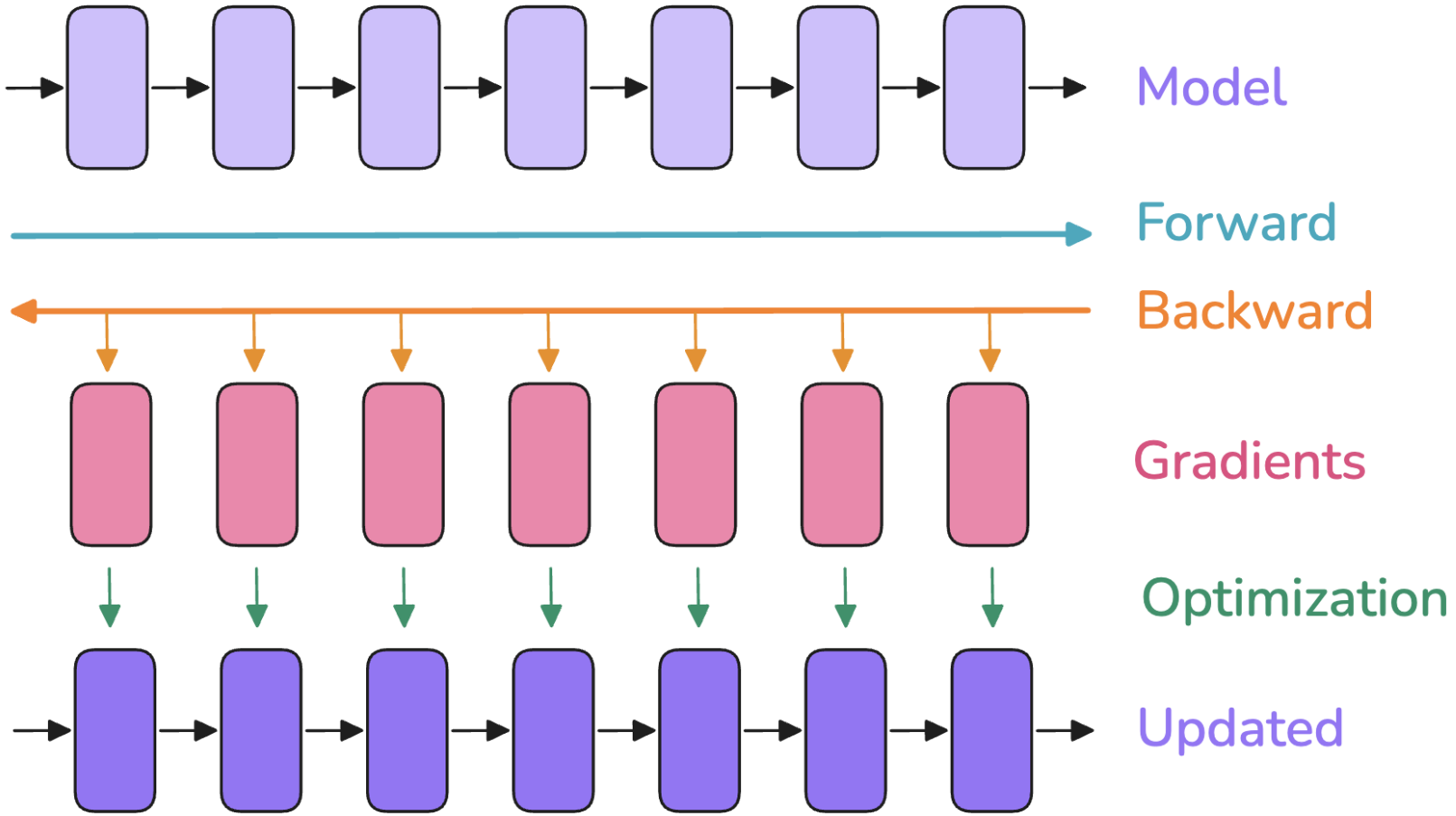
Memory usage in Transformers

Batch Size: 1



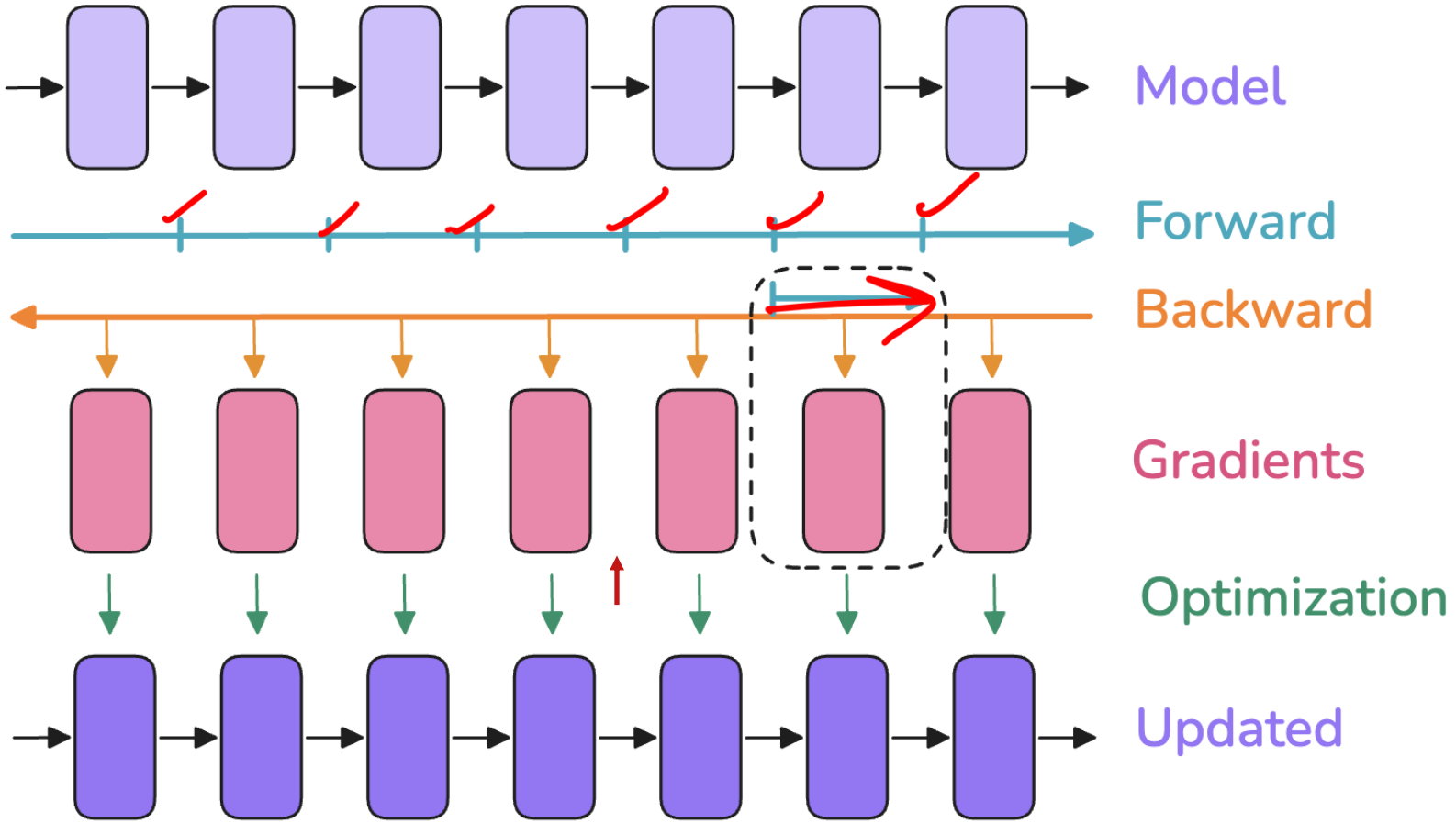
For large numbers of input tokens (i.e., large batch sizes / sequences), activations become by far the largest memory burden.

Activation re-computation *aka* Gradient Checkpointing



Activation re-computation *aka* Gradient Checkpointing

- Discard some activations during the forward pass to save memory
- Spend some extra compute to recompute these on the fly during the backward pass.

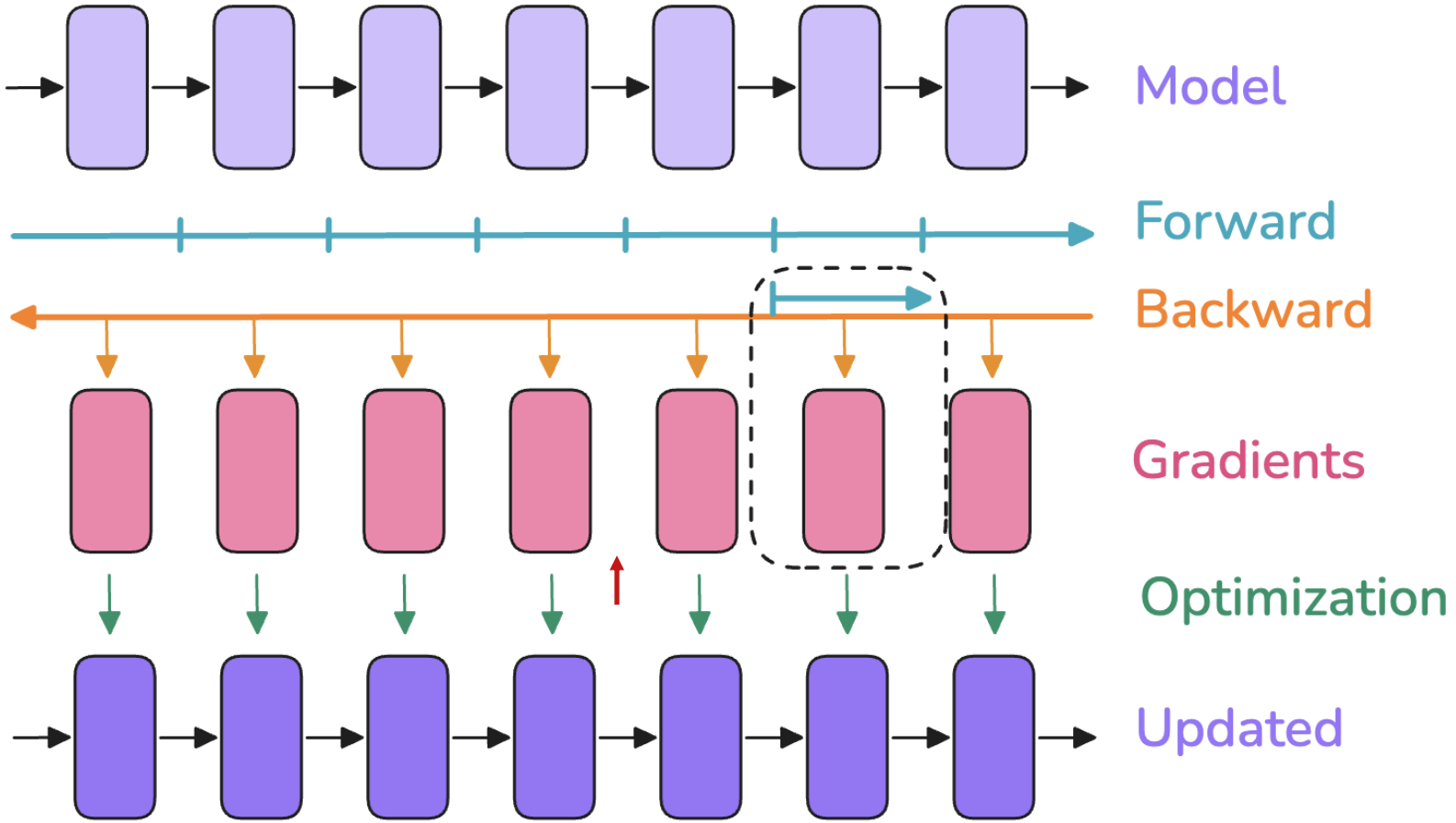


[Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models]

Activation re-computation *aka* Gradient Checkpointing

- Discard some activations during the forward pass to save memory
- Spend some extra compute to recompute these on the fly during the backward pass.

Which activations to store?



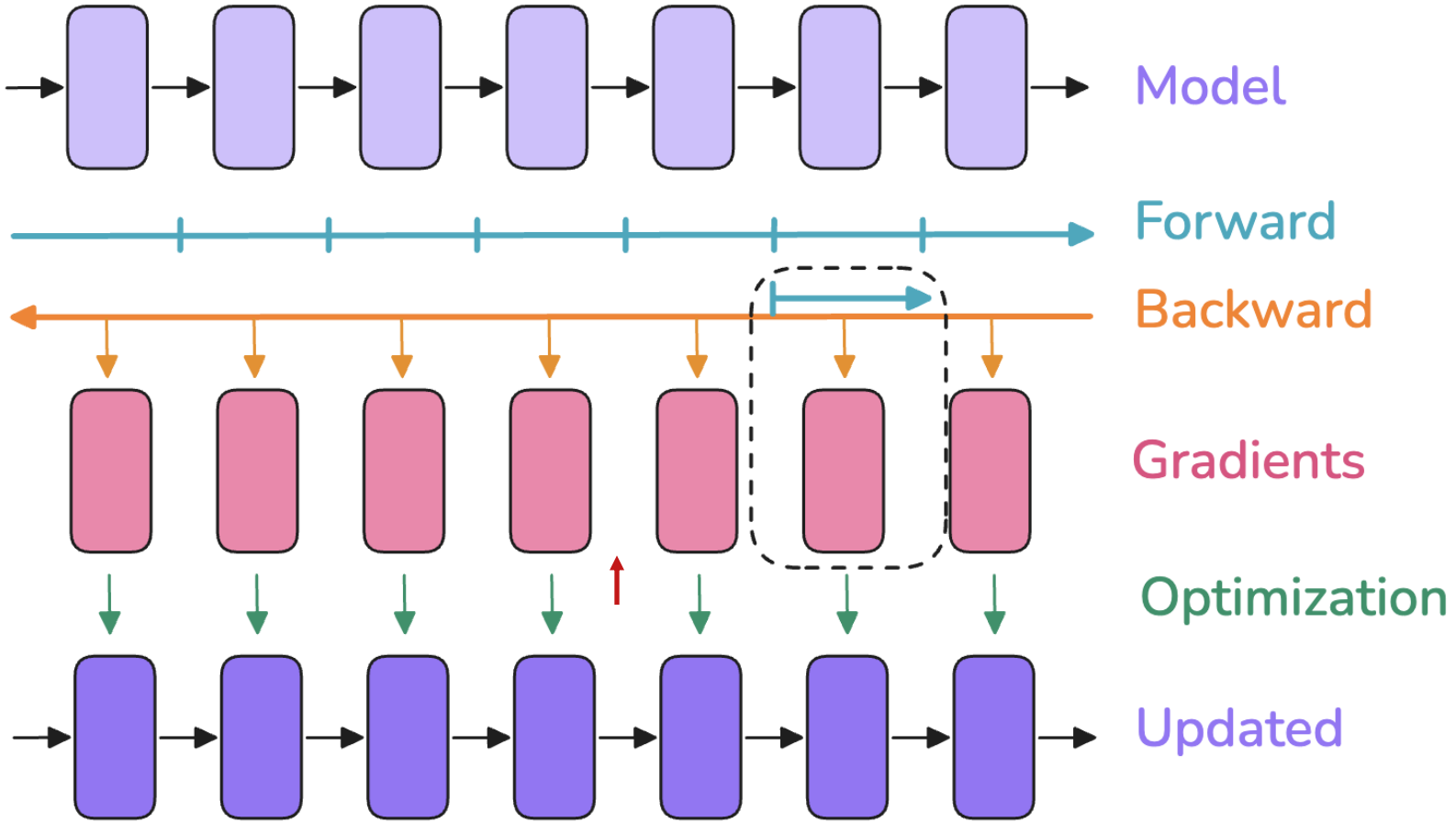
[Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models]

Activation re-computation *aka* Gradient Checkpointing

- Discard some activations during the forward pass to save memory
- Spend some extra compute to recompute these on the fly during the backward pass.

Which activations to store?

- **Full:** Save at the transition of layers
  - 👍 Saves most memory
  - 👎 Most expensive – 30-40%



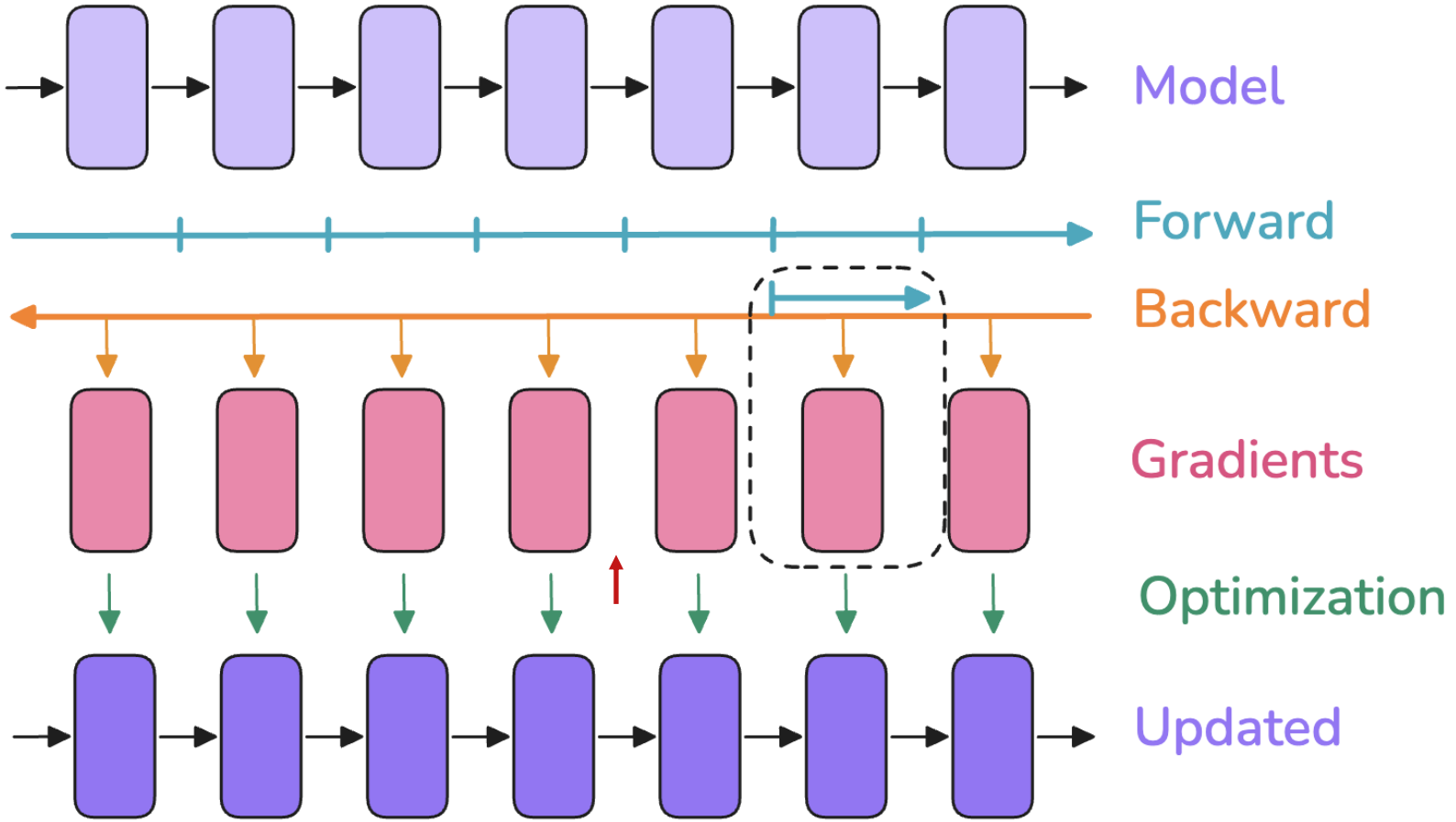
Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models

Activation re-computation *aka* Gradient Checkpointing

- Discard some activations during the forward pass to save memory
- Spend some extra compute to recompute these on the fly during the backward pass.

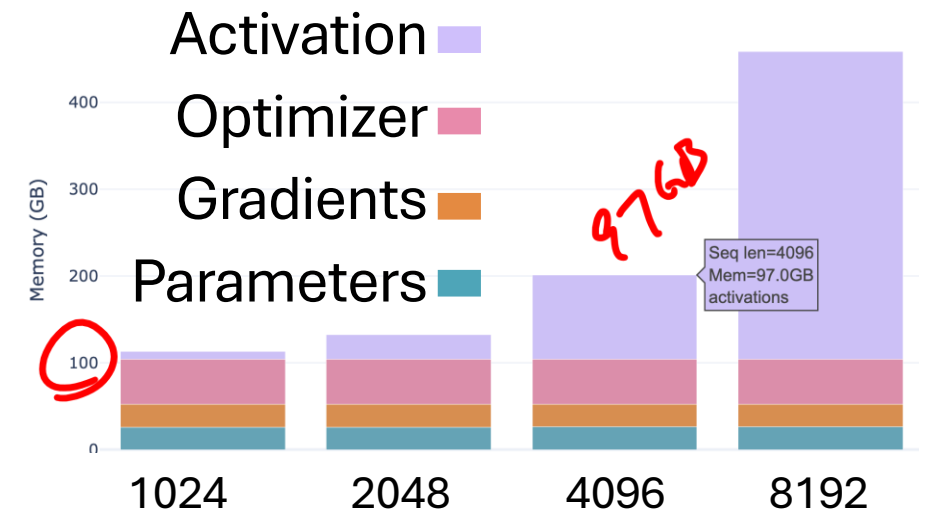
Which activations to store?

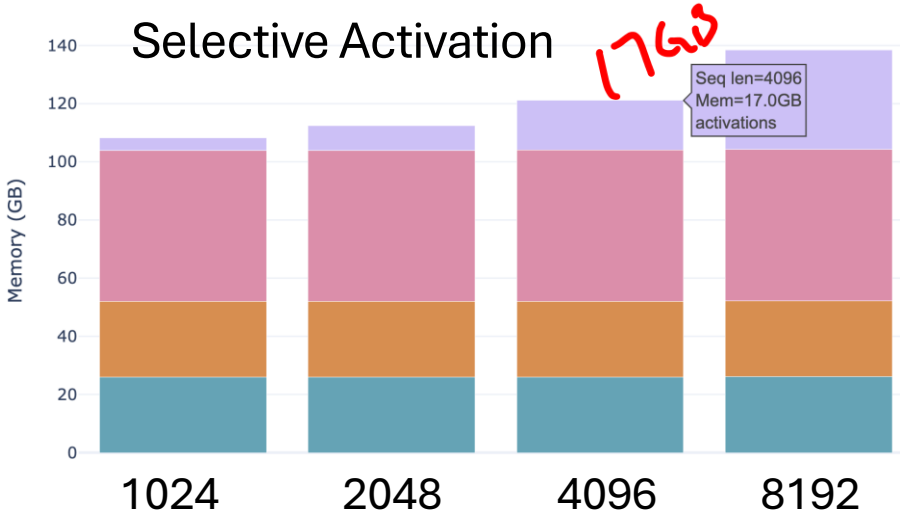
- **Full:** Save at the transition of layers
  - 👍 Saves most memory
  - 👎 Most expensive – 30-40%
- **Selective:** Save MLP; discard attn
  - **70%** memory reduction ✓
  - **2.7%** compute cost ✓

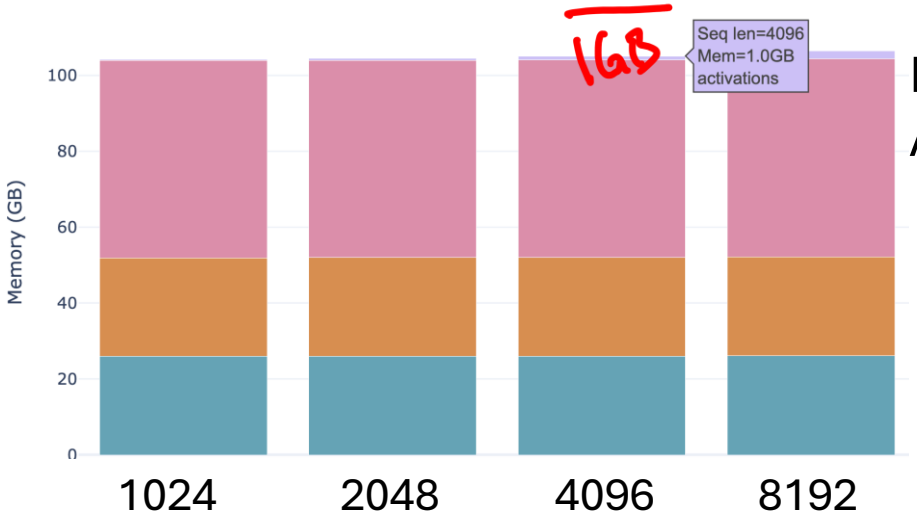


Korthikanti etal. 2022, Reducing Activation Recomputation in Large Transformer Models

Memory usage in Transformers – 8B Model , *bs* = 1

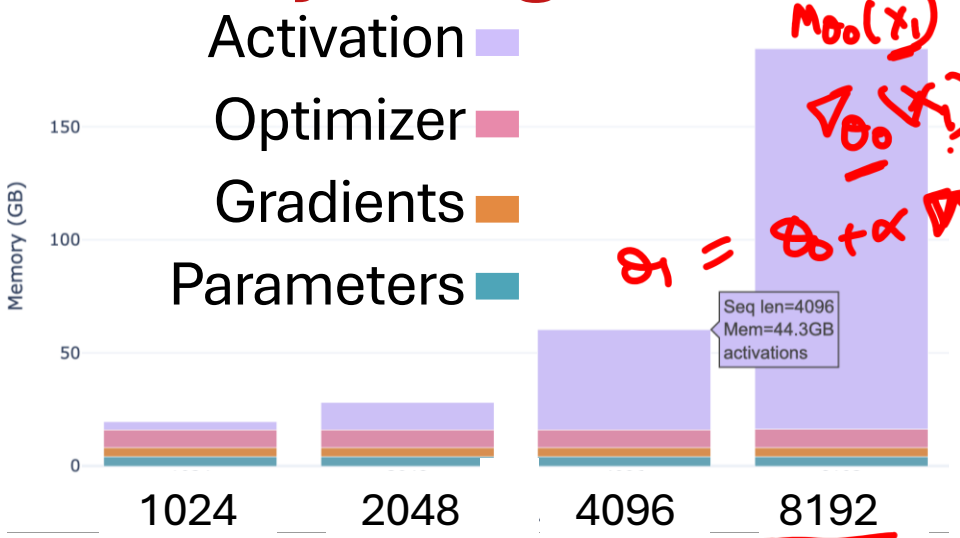


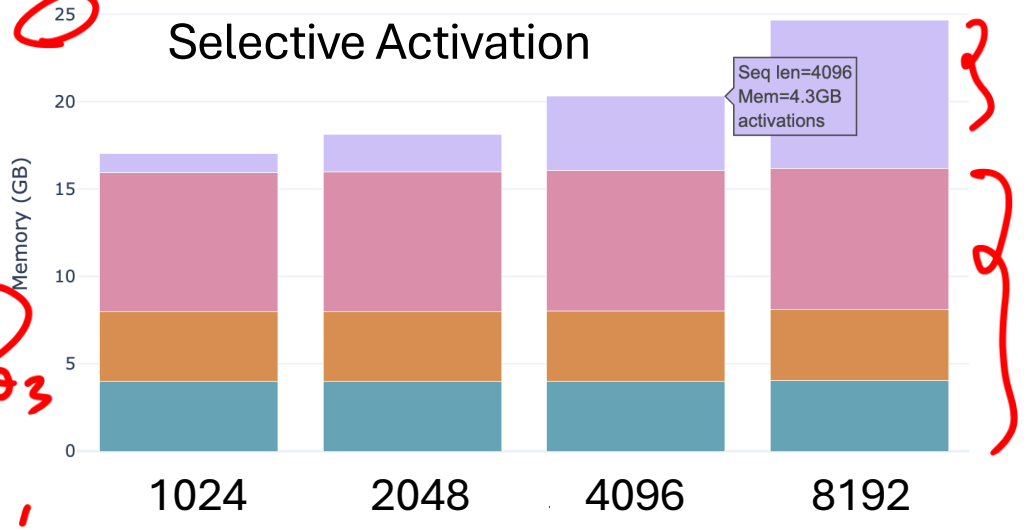


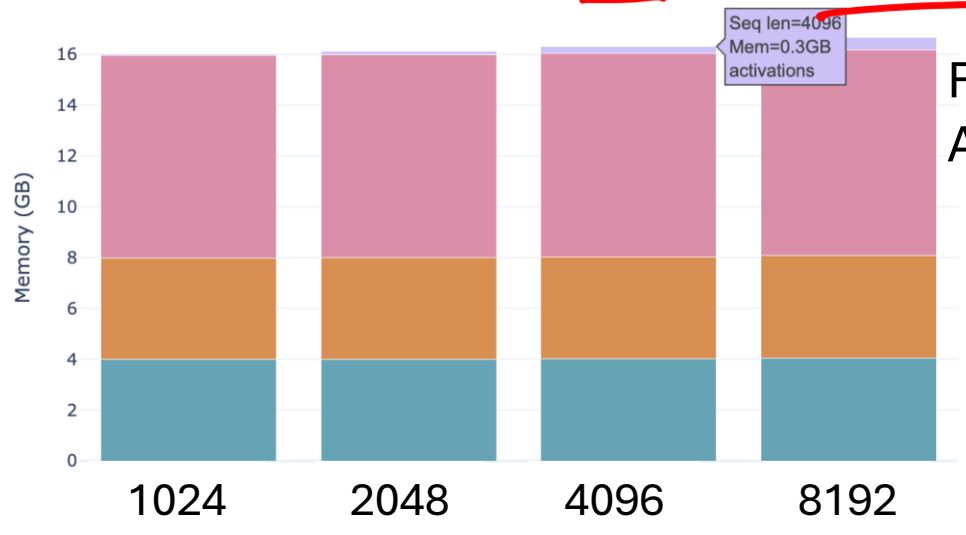


Full Activation

Memory usage in Transformers – 1B Model , $bs = 1$







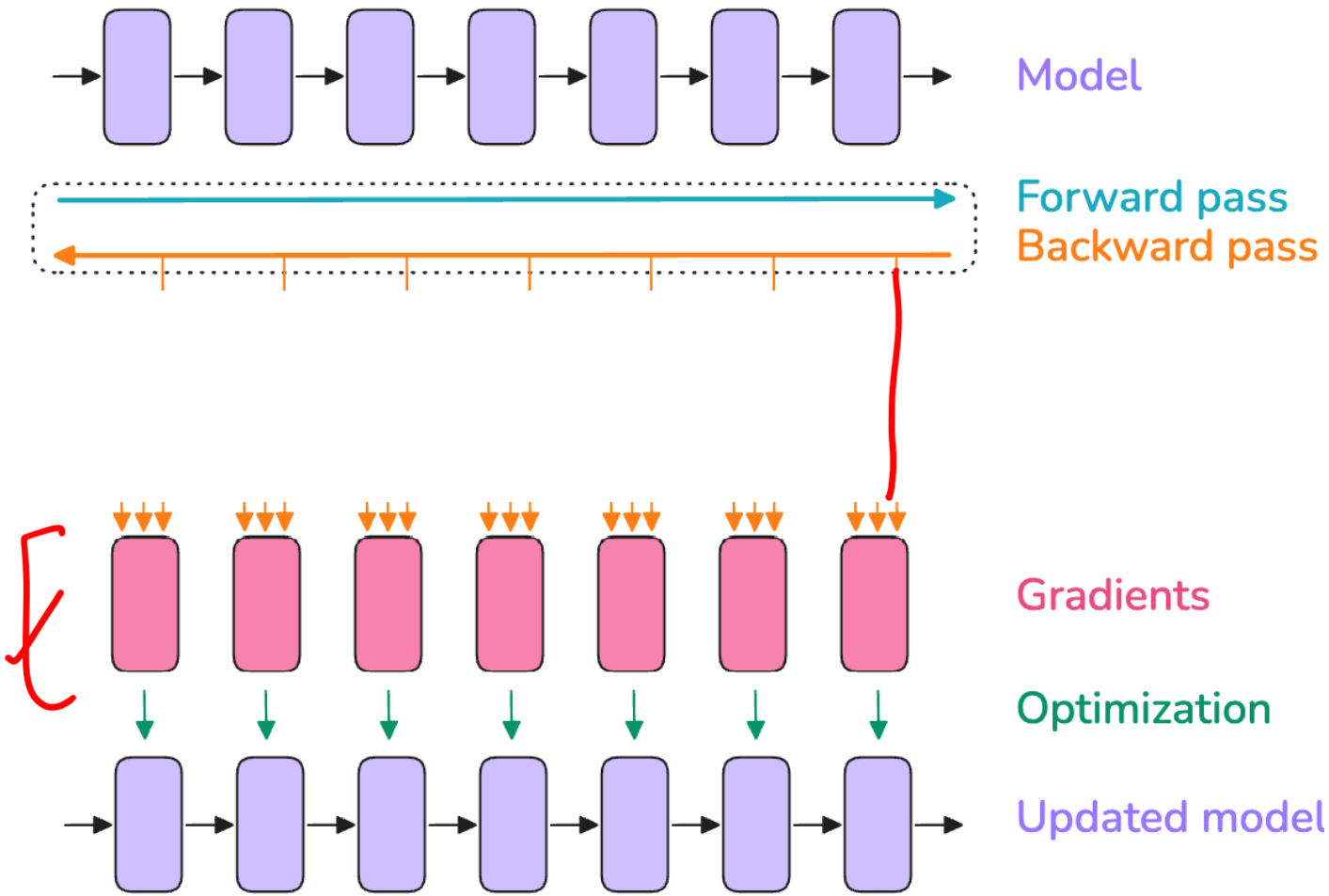
Full Activation

$$m_{act} = L * \left(\frac{34 * seq * bs * h}{+ 5 * n_{heads} * seq^2 * bs} \right)$$

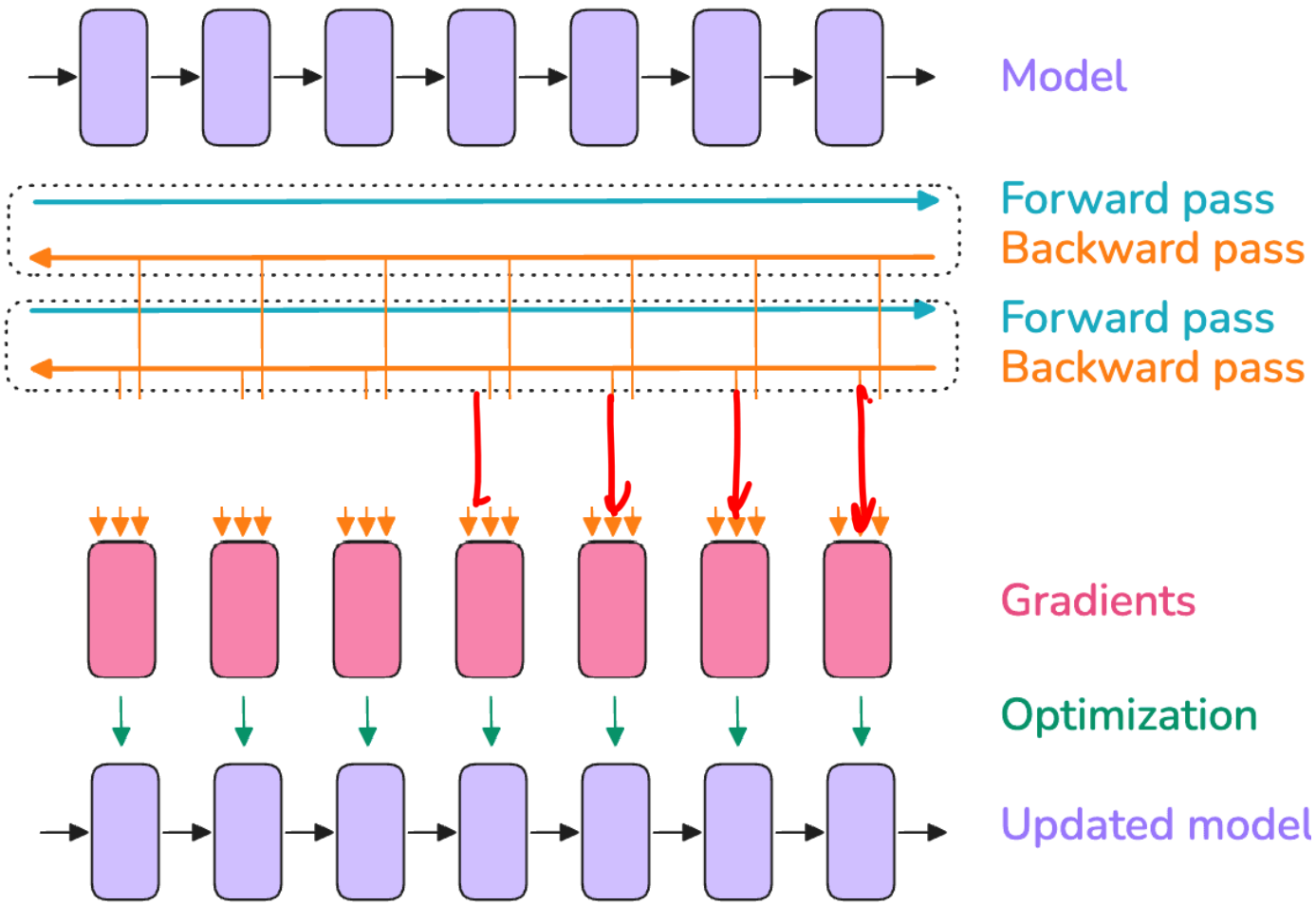
Activation memory – grows linearly with batch size

How to increase the batch size?

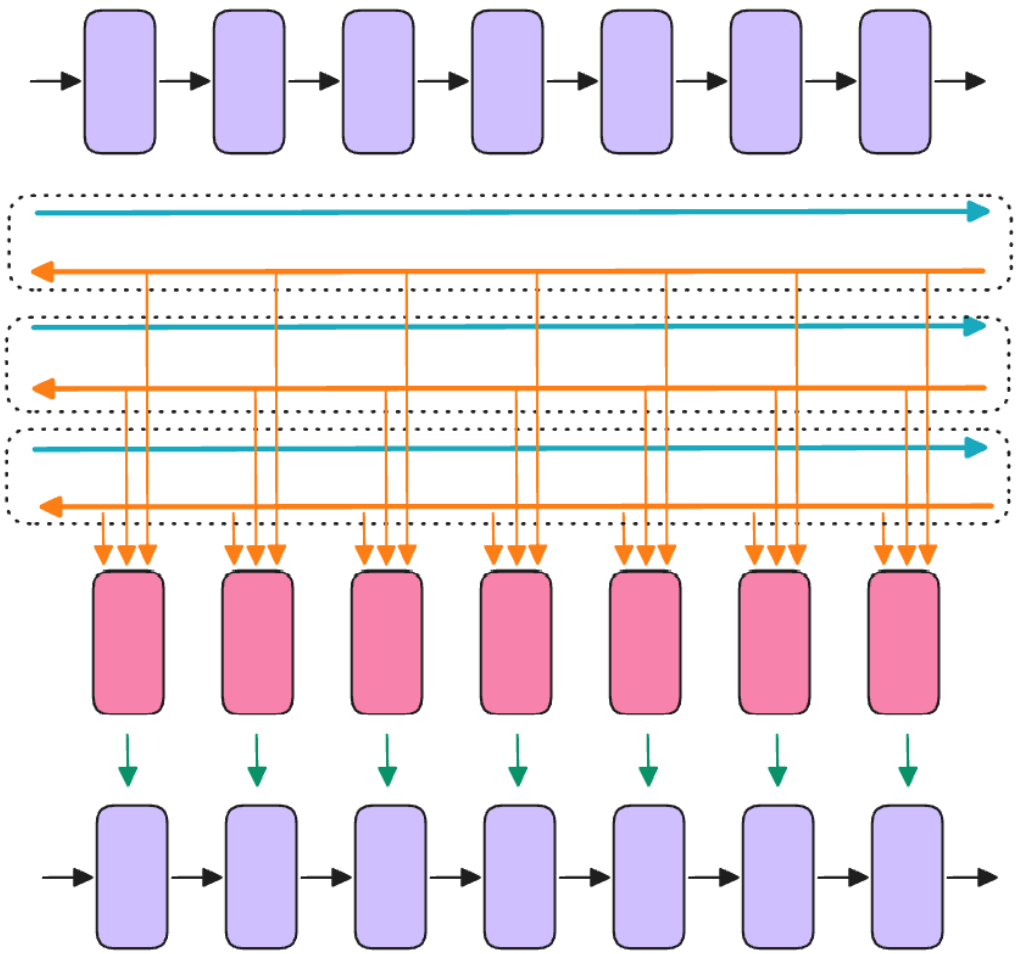
How to increase batch size? – *Gradient Accumulation*



How to increase batch size? – *Gradient Accumulation*



How to increase batch size? – *Gradient Accumulation*

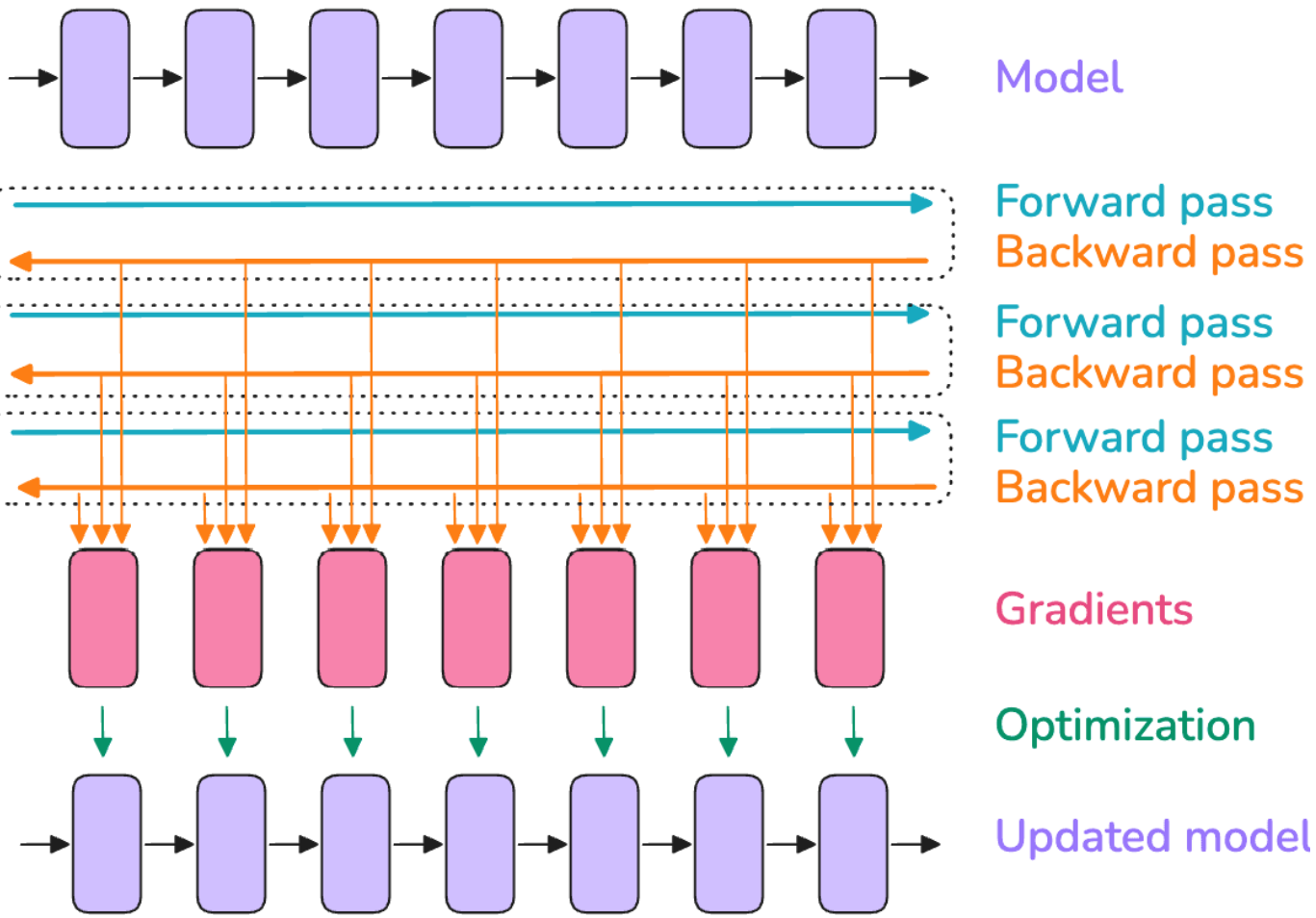


opt:

opti

optim.

How to increase batch size? – *Gradient Accumulation*



- Process smaller micro-batches sequentially
- $bs = gbs = mbs * grad\_acc$

Only one micro-batch's worth of activations needs to be kept in memory at a time

Requires multiple consecutive forward/backward passes per optimization step

How to speedup?