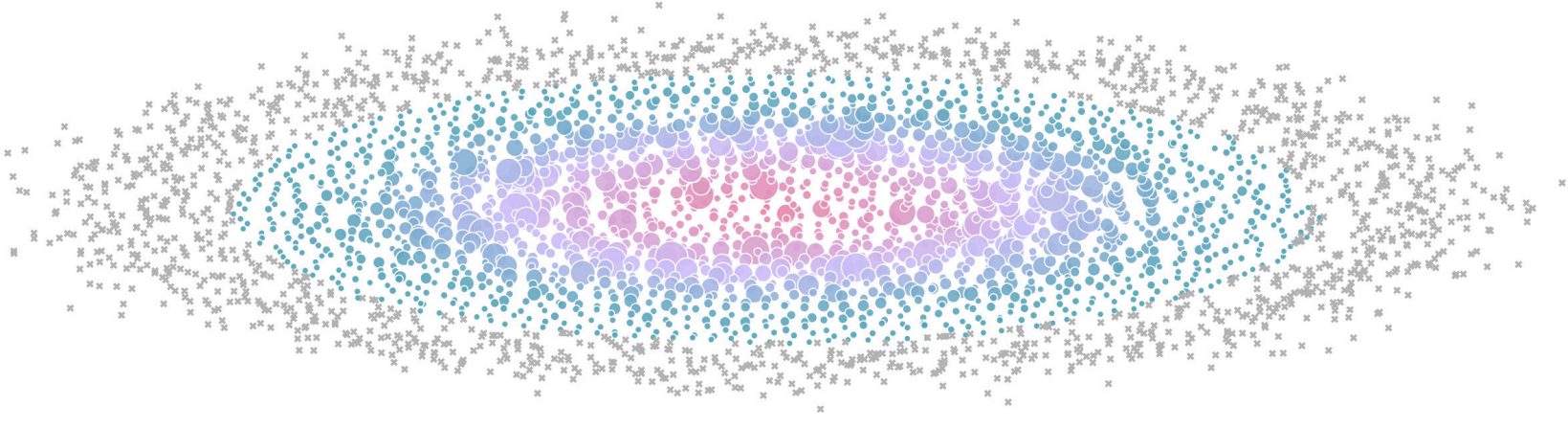
The Ultra-Scale Playbook: Training LLMs on GPU Clusters

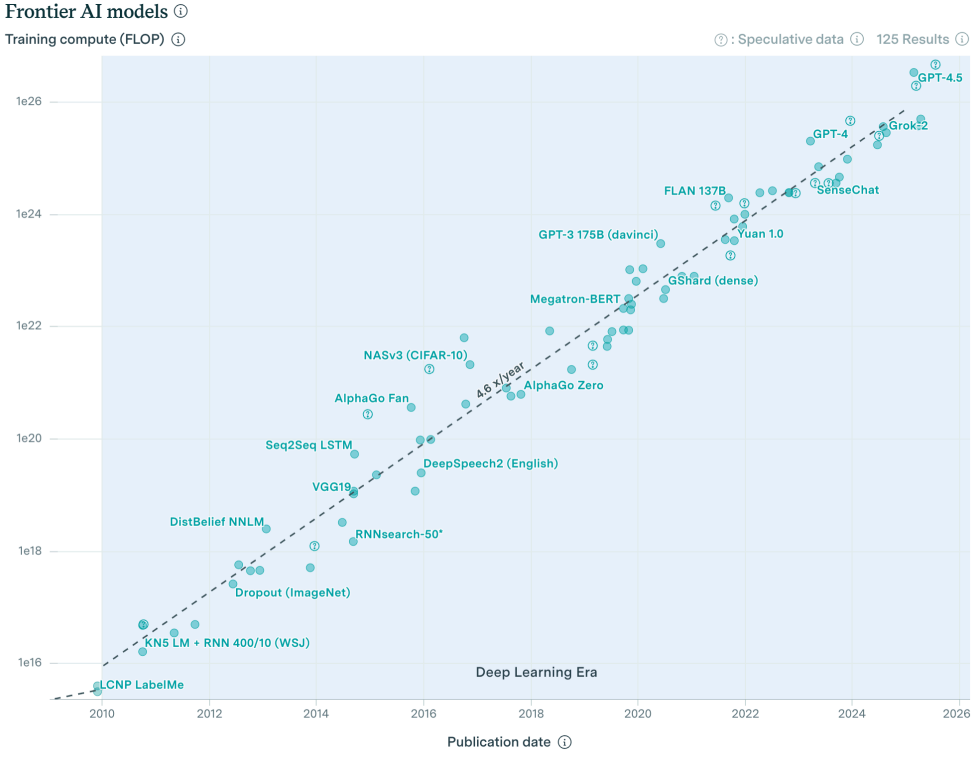


We ran over 4,000 scaling experiments on up to 512 GPUs and measured throughput (size of markers) and GPU utilization (color of markers). Note that both are normalized per model size in this visualization.

Week 1: the plan

1. Intro on scaling in 2025
2. Scaling frame: the three constraints
3. One training batch
4. What actually fills HBM & memory maths
5. Activation recomputation
6. Gradient accumulation

Scaling in 2025

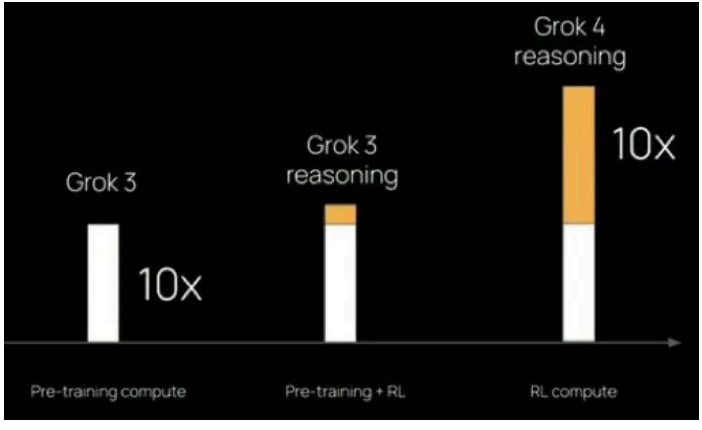


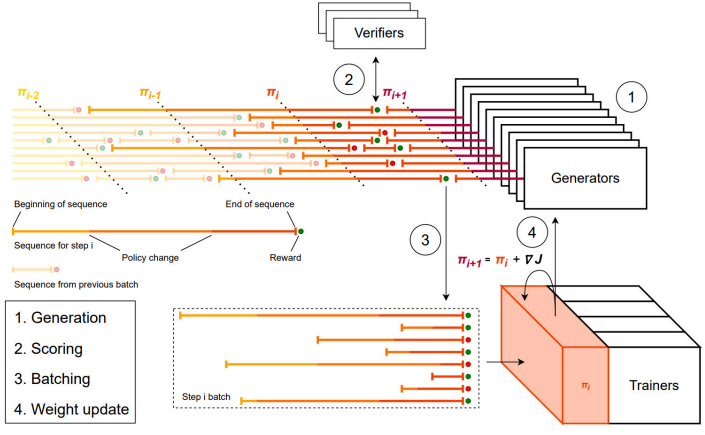
Current infrastructure	30+ GW new build $1.4T in total

Historically: pretrain FLOPs

Now: ?

Scaling in 2025

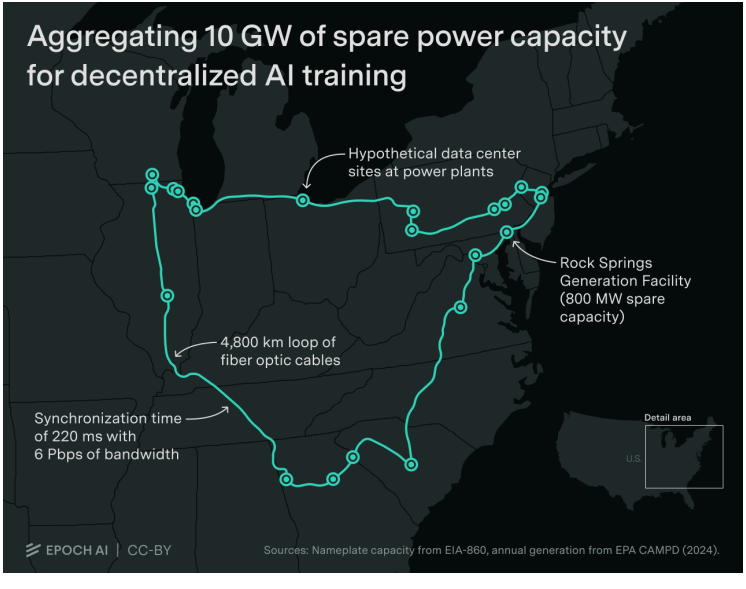


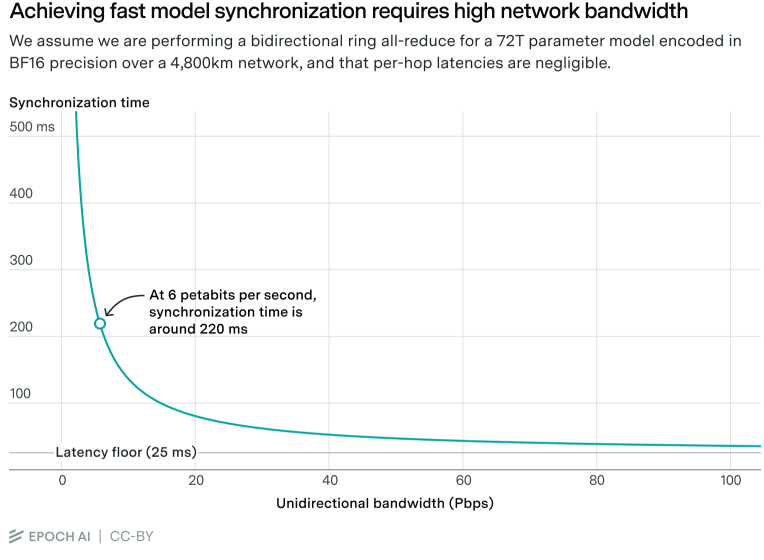


What's cool & different:

- we might use different GPUs (*)
- less latency / sync dependent
- multi-DC is easier
- verification / execution adds additional "bubbles" in compute

Scaling in 2025





Read here: epoch.ai

Scaling frame: the three constraints

All the techniques we'll cover in this book tackle one or several of the following three key challenges, which we'll bump into repeatedly:

1. **Memory usage:** This is a hard limitation - if a training step doesn't fit in memory, training cannot proceed.
2. **Compute efficiency:** We want our hardware to spend most time computing, so we need to reduce time spent on data transfers or waiting for other GPUs to perform work.
3. **Communication overhead:** We want to minimize communication overhead, as it keeps GPUs idle. To achieve this, we will try to make the best use of intra-node (fast) and inter-node (slower) bandwidths and to overlap communication with compute as much as possible.

In many places, we'll see that we can trade one of these (computation, communication, memory) off against another (e.g., through recomputation or tensor parallelism). Finding the right balance is key to scaling training.

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

One training batch

How to compute BS / BST?

> A small batch size can be useful early in training to quickly move through the training landscape to reach an optimal learning point. However, further along in the model training, small batch sizes will keep gradients noisy, and the model may not be able to converge to the most optimal final performance.

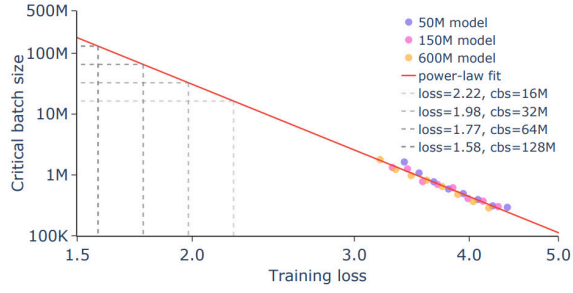


Figure 13 | **The power-law fit for the training loss and the critical batch size, utilizing data from models ranging from 50M to 600M in activated parameters counts.** We mark the points where the batch size is doubled with dashed gray lines.

and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and the weight decay is set to 0.1. The training sequence length is 8192, and the batch size is progressively scaled from an initial size of 16M to 32M at 69B tokens, to 64M at 790B tokens, and finally to 128M at 4.7T tokens, where it remains until the end of training. The schedule is designed based on the correlation between training loss and the critical batch size (McCandlish et al., 2018). It is argued that training at the critical batch size yields a near-optimal balance between training time and data efficiency (Kaplan et al., 2020). Following this, we fit a power-law relationship between the loss and the critical batch size on data from smaller models, as shown in Figure 13. The batch size is doubled when the corresponding loss is reached.

One training batch

In the simplest case, training on a single machine, the bs (in samples) and bst can be computed from the model input sequence length (seq) as follows:

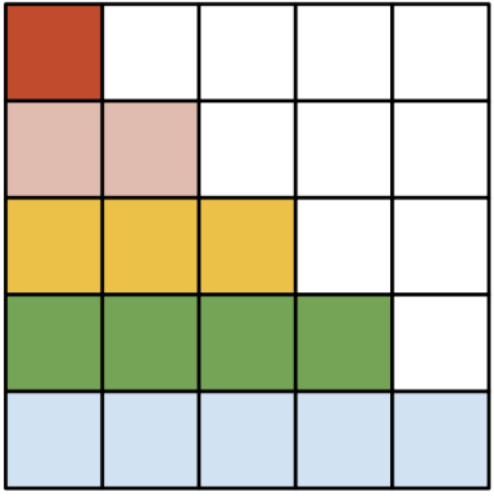
$$bst = bs * seq$$

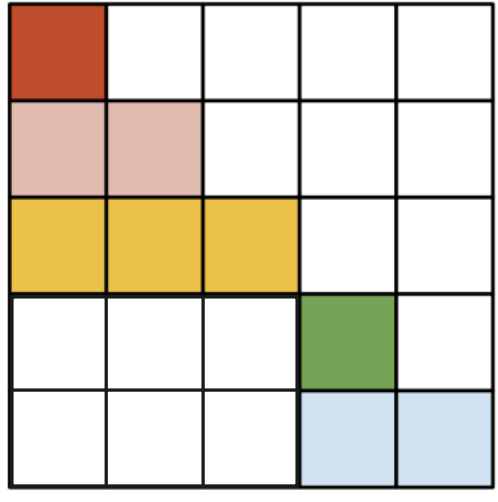
BS doesn't really tell the story in full:

During training we always train on sequences of the full $n_{\text{ctx}} = 2048$ token context window, packing multiple documents into a single sequence when documents are shorter than 2048, in order to increase computational efficiency. Sequences with multiple documents are not masked in any special way but instead documents within a sequence are delimited with a special end of text token, giving the language model the information necessary to infer that context separated by the end of text token is unrelated. This allows for efficient training without need for any special sequence-specific masking.

Question to keep for the further discussion: cross-document masking

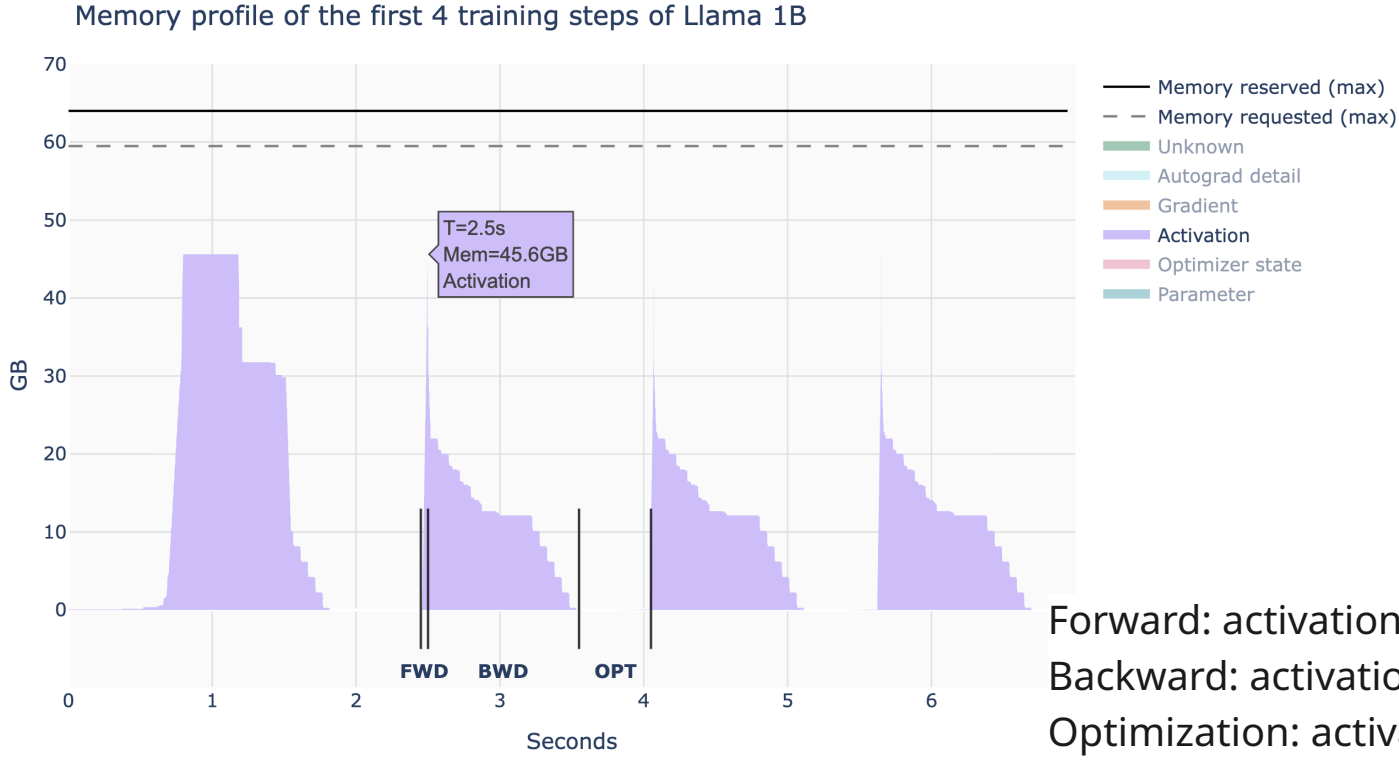
One training batch





Question to keep for the further discussion: cross-document masking

Profiling the memory usage

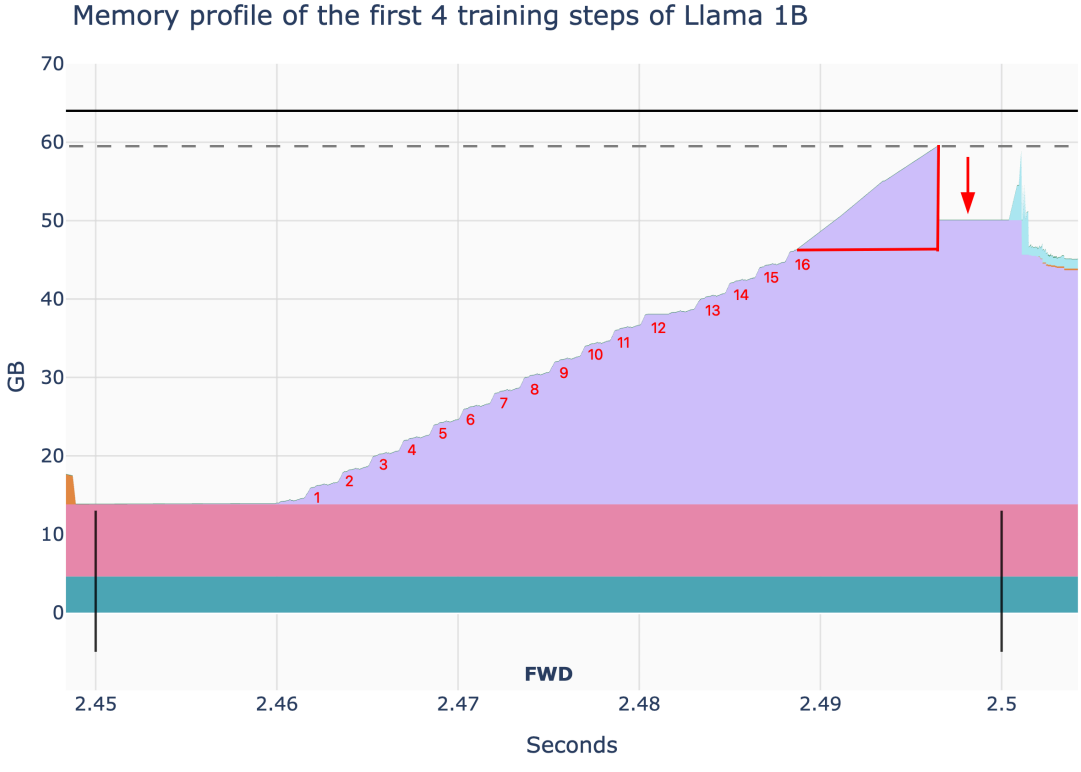


Forward: activations ↑

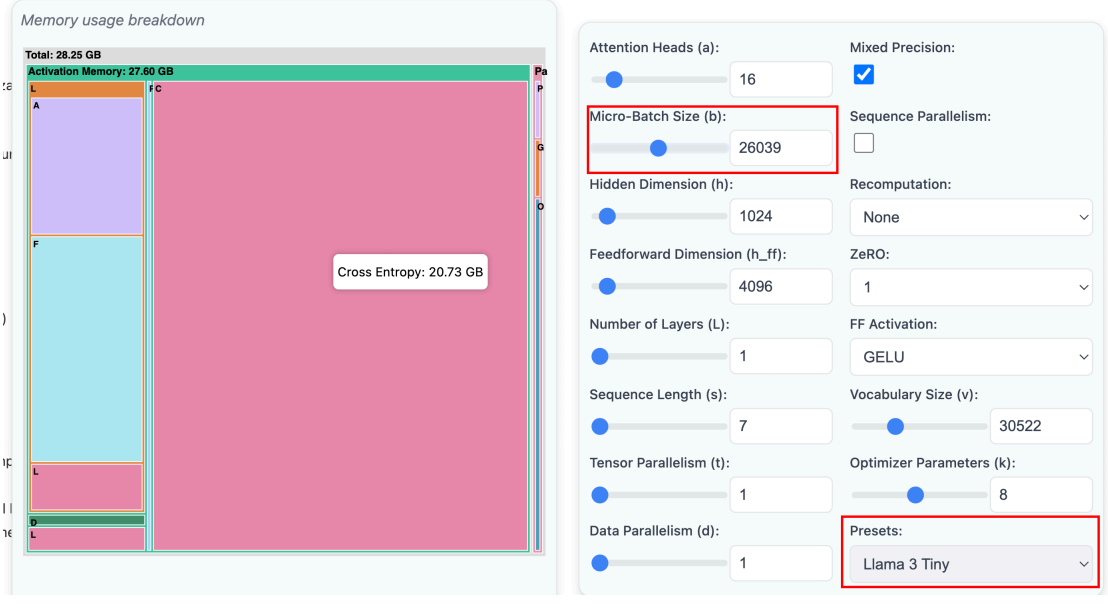
Backward: activations ↓

Optimization: activations 0️⃣

Profiling the memory usage



Profiling the memory usage



Relevant reading:

[Cut Your Losses in Large-Vocabulary Language Models]

Memory for Weights

For a simple transformer LLM, the number of parameters is given by the following formula:

$$C = E(V + P) + L(12E^2 + 13E) + 2E$$

Per-layer breakdown

Node	Parameters	Contribution to E^2	Contribution to E
LayerNorm 1	$\gamma \in \mathbb{R}^E,\ \beta \in \mathbb{R}^E$	0	$2E$
Attention: c_attn	weights $E \times 3E$, bias $3E$	$3E^2$	$3E$
Attention: c_proj	weights $E \times E$, bias E	E^2	E
LayerNorm 2	γ, β as above	0	$2E$
MLP: c_fc	weights $E \times H$, bias H	EH	H
MLP: c_proj	weights $H \times E$, bias E	HE	E
Activation, Dropout	no parameters	0	0

(with $H = 4E$)

$$\underbrace{(4+8)}_{12}E^2 + \underbrace{(4+4+5)}_{13}E.$$

* You will often see `H = 4 * E`. Why `4 * E`? I think this is simply by convention. The original paper uses `H = 4 * E`, and it seems like most implementations follow suit.

Memory for Weights

From LLAMA-1 paper:

SwiGLU activation function [PaLM]. We replace the ReLU non-linearity by the SwiGLU activation function, introduced by Shazeer (2020) to improve the performance. We use a dimension of $\frac{2}{3}4d$ instead of $4d$ as in PaLM.

* You will often see `H = 4 * E`. Why `4 * E`? I think this is simply by convention. The original paper uses `H = 4 * E`, and it seems like most implementations follow suit.

Mixed Precision: the trick(y) part

Interestingly, mixed precision training itself doesn't save memory; it just distributes the memory differently across the three components, and in fact adds another 4 bytes over full precision training if we accumulate gradients in FP32. It's still advantageous, though, as computing the forward/backward passes in half precision (1) allows us to use optimized lower precision operations on the GPU, which are faster, and (2) reduces the activation memory requirements during the forward pass, which as we saw in the graph above is a large part of the memory usage.

$$
\begin{aligned}
m_{params} &= 2 * N \\
m_{grad} &= 2 * N \\
m_{params\_fp32} &= 4 * N \\
m_{opt} &= (4 + 4) * N
\end{aligned}
$$

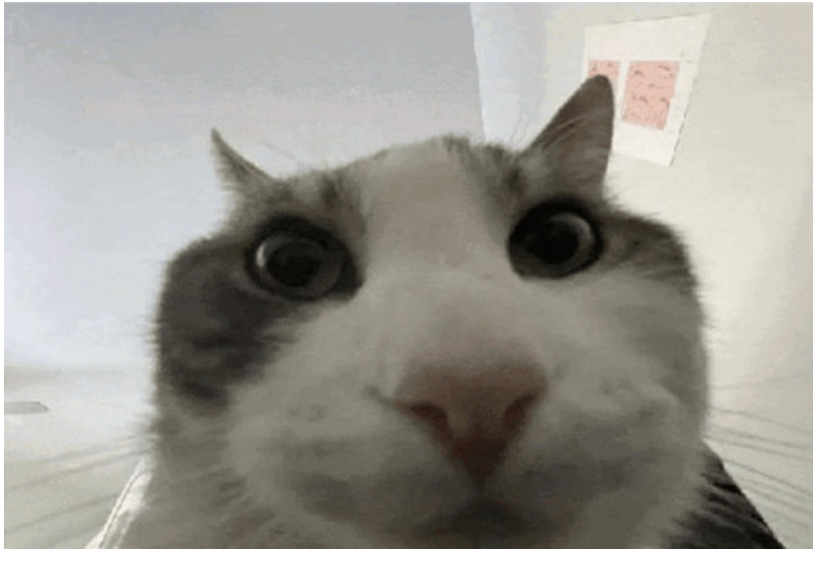
Instead of 0 + 4 + 4 + (4+4)

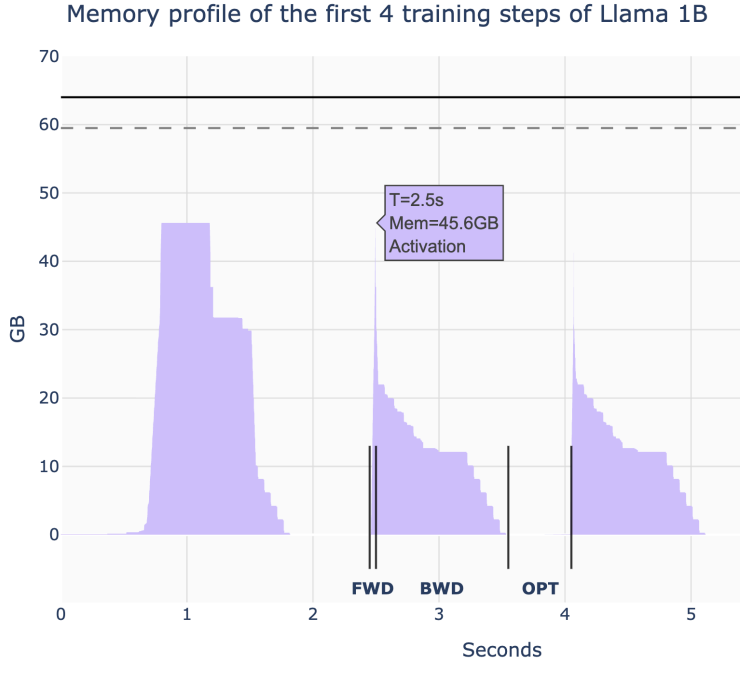
Gradient checkpointing

> look inside

> there's no checkpointing, in fact, we delete stuff

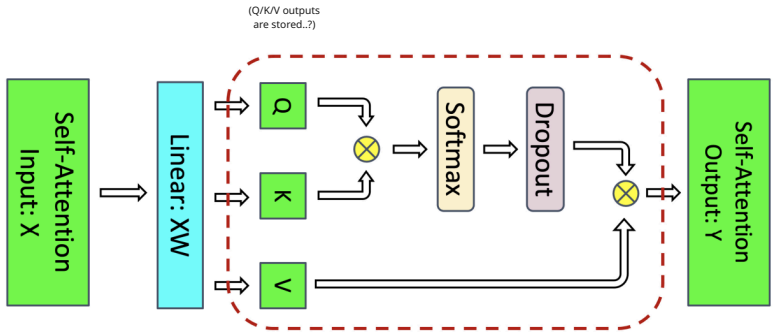
> works with activations





Gradient checkpointing

Key intuition from the paper: «we propose to checkpoint and recompute only parts of each transformer layer that take up a considerable amount of memory but are not computationally expensive to recompute».



For GPT-3, the activations ratio is 80 : 34 (compute: 3 : 97)

Figure 3: Self-attention block. The red dashed line shows the regions to which selective activation recomputation is applied (see Section 5 for more details on selective activation recomputation).

No implementation in Picotron?

Gradient checkpointing

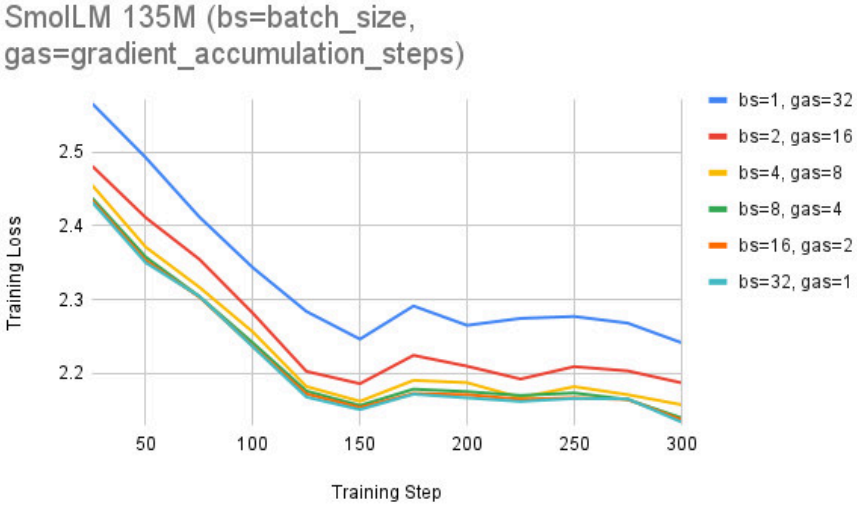
Are numbers & conclusions from “Reducing Activation Recomputation” valid for today’s MOE-based LLMs?

Gradient accumulation

«In practice, the optimization step is conducted not on the sum but on the average of the gradients, so that the result is independent of the number of gradient accumulation steps».

unsloth

Blogpost link



$$\ell(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq \text{ignore\_index}\}} l_n, & \text{if reduction} = \text{'mean';} \\ \sum_{n=1}^{N} l_n, & \text{if reduction} = \text{'sum'.} \end{cases}$$

This means naively averaging over each gradient accumulation step is wrong, but instead we must derive the denominator beforehand.

Questions & Discussion

My q: HUF and MUF