AI
SWEDEN

Building LLMs in Practice

Part 1

Amaru Cuba Gyllensten
2025-05-19

AI
SWEDEN

AGENDA

I. Model Training Basics

II. Basic strategies:
   - Gradient accumulation
   - Activation recomputation
   - Data parallelism
   - Mixed Precision

III. Benchmarking, finding a training configuration

Many examples and illustrations taken from the excellent ultra-scale playbook:
https://huggingface.co/spaces/nanotron/ultrascale-playbook

Model training basics

What does it take to train an LLM?

AI SWEDEN

Outline

- What are the resource requirements to train an LLM?
- How does memory and compute scale w.r.t. hyperparameters?

Model training basics

AI SWEDEN

- LLMs require vast amounts of compute resources to train.
- It is necessary to distribute training across multiple GPUs.
- To do that efficiently, a number of strategies are required to deal with issues related to
    - memory
        - Hard limit!
    - compute
    - communication

Model training basics

AI SWEDEN

- LLMs require vast amounts of compute resources to train.
- It is necessary to distribute training across multiple GPUs.
- To do that efficiently, a number of strategies are required to deal with issues related to
  - memory
  - compute
  - communication
- But first, what are the resource requirements?

Model training basics (Compute)

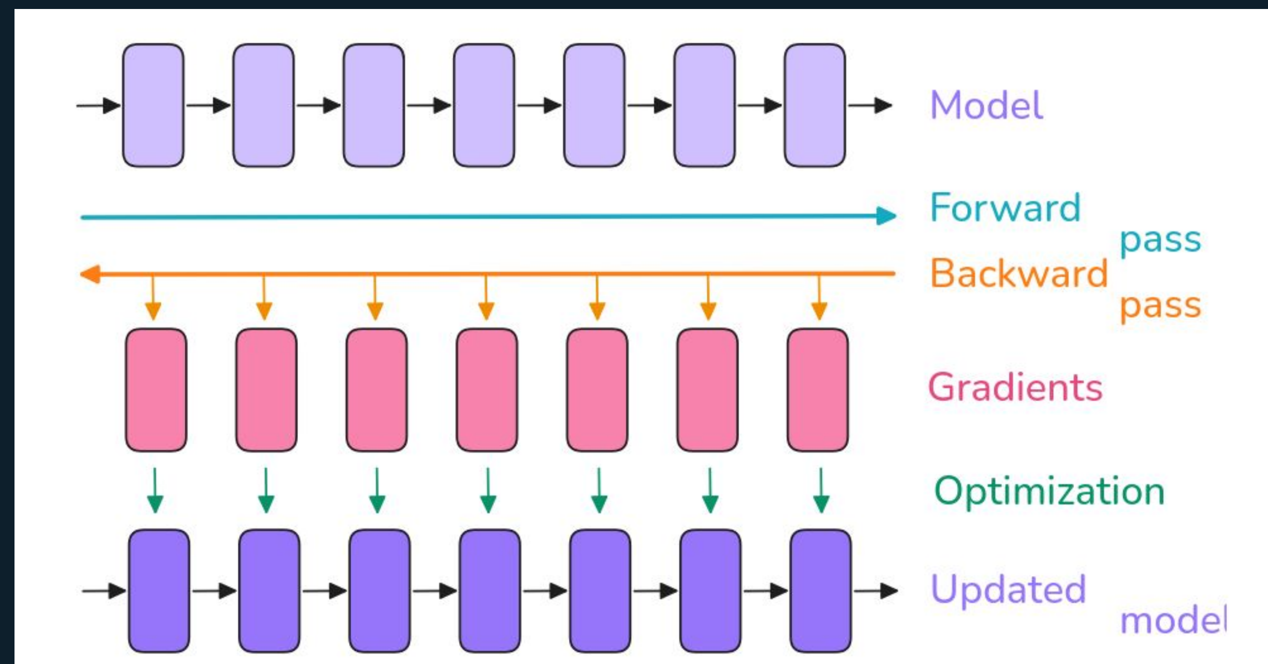
- LLama 3 8B:
  - Trained on 15 Trillion tokens
  - Compute: ~8e23 FLOPS
  - With 40% Utilization:
    - ~71600 A100 days
    - 150 Berzelius days. (60 x 8 A100)

We need lots of GPUs, and to train efficiently on lots of GPUs.

Model training basics (Memory)

AI SWEDEN

- What happens during training:
  - Forward pass
  - Backward pass
  - Optimizer step
- What do we need to keep track of?
  - Parameters
  - Gradients
  - Activations
  - Optimizer state



Requirements: High-level

- What’s going on inside an LLM?

Requirements: High-level

- What's going on inside an LLM?
  - **(batched) Matrix multiplications**

Requirements: High-level

- What's going on inside an LLM?
  - **(batched) Matrix multiplications**
- A K by N matrix consists of KN parameters.
- Multiplying an M by K matrix with a K by N matrix requires 2MNK FLOPS.

Requirements: High-level

AI SWEDEN

- MLP:
  - Two linear transformations, i.e. two matrix multiplications.
- Self-attention:
  - QKV projection.
  - Attention computation.
  - Value aggregation
  - Out Projection

- Classification head:
  - Embedding => Vocab project. i.e. matrix multiplication.
  - (We ignore this, for the sake of simplicity, has limited impact when scaling up)

Requirements: High-level

AI SWEDEN

- MLP:
  - Two linear transformations, i.e. two matrix multiplications.
- Self-attention:
  - QKV projection.
  - Attention computation.
  - Value aggregation
  - Out Projection

B	Batch size
S	Sequence length
D	Embedding dimension
L	Layers

Layer	FLOPS	Parameters	Activations
MLP	$O(B \times S \times D^2)$	$O(D^2)$	$O(B \times S \times D)$
Attention	$O(B \times S \times D \times (S + D))$	$O(D^2)$	$O(B \times S \times (D + S))$
Transformer	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times B \times S \times (D + S))$

This uses naive self attention, not flash-attention!

We're ignoring constants here, which are quite important to get the full picture. For more exact derivations, see this paper by NVIDIA: **Reducing Activation Recomputation in Large Transformer Models (https://arxiv.org/abs/2205.05198)**

Requirements: High-level

AI SWEDEN

Param	FLOPS	Parameters	Activations
B	Linear		Linear
S	Quadratic		Quadratic
D	Quadratic	Quadratic	Linear
L	Linear	Linear	Linear

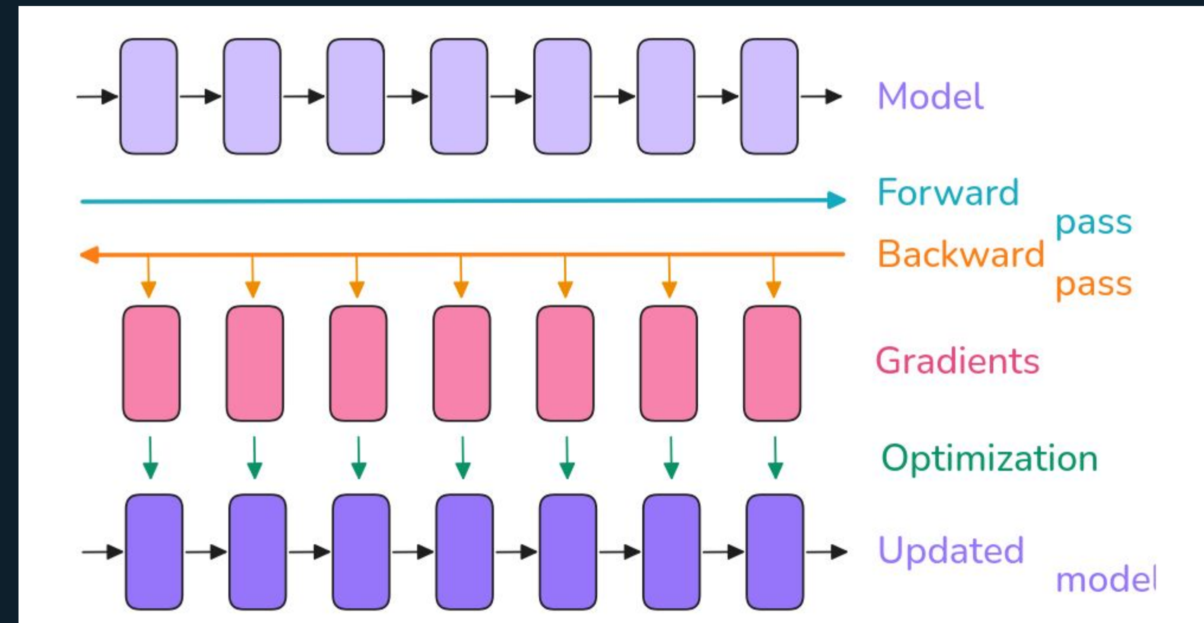
Layer	FLOPS	Parameters	Activations
MLP	$O(B \times S \times D^2)$	$O(D^2)$	$O(B \times S \times D)$
Attention	$O(B \times S \times D \times (S + D))$	$O(D^2)$	$O(B \times S \times (D + S))$
Transformer	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times B \times S \times (D + S))$

AI SWEDEN

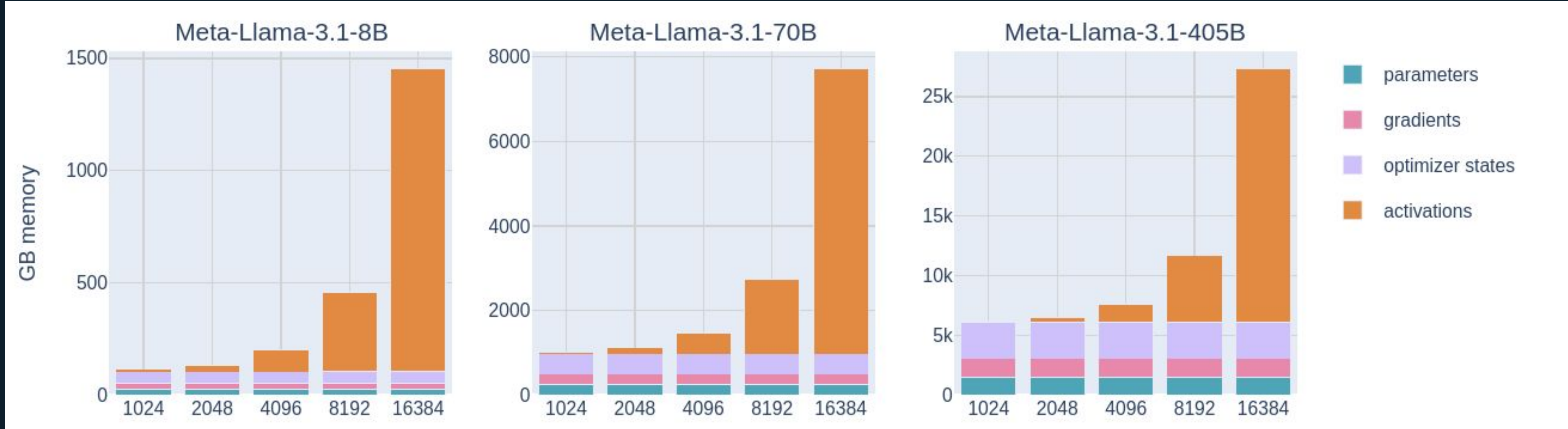
- FP32 8B Parameter model:
  - parameters: 32 GB
  - gradient: 32 GB
  - optimizer state: 64 GB
    - (Adam: mean + “variance”)

Just the model needs 128 GB of memory!

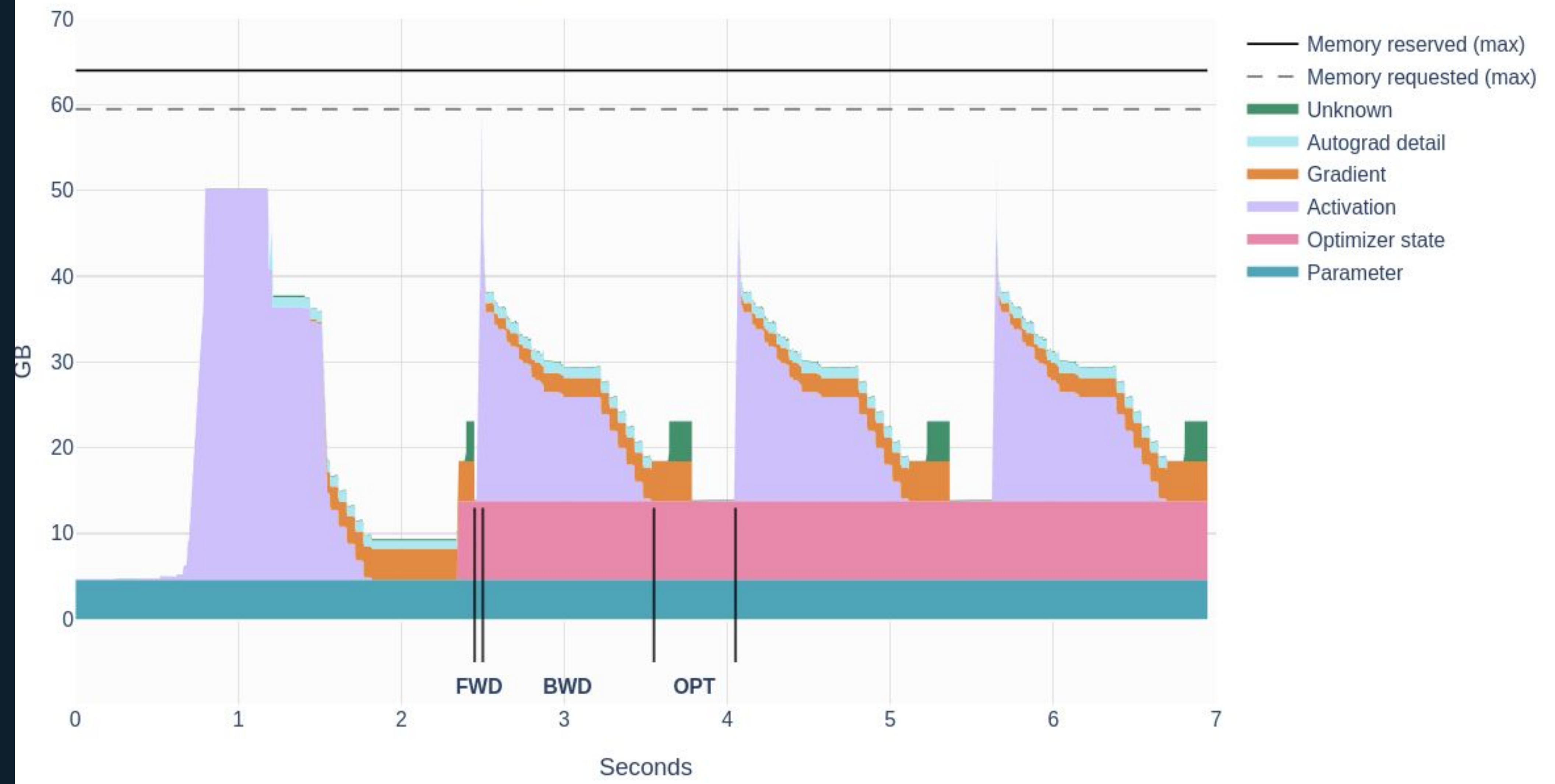
Activations are added on top of this.



AI
W E D E N



Layer	FLOPS	Parameters	Activations
MLP	$O(B \times S \times D^2)$	$O(D^2)$	$O(B \times S \times D)$
Attention	$O(B \times S \times D \times (S + D))$	$O(D^2)$	$O(B \times S \times (D + S))$
Transformer	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times B \times S \times (D + S))$


Memory profile of the first 4 training steps of Llama 1B
70
60
50
40
30
20
10
0
GB
0
1
2
3
4
5
6
7
Seconds
FWD
BWD
OPT
Memory reserved (max)
Memory requested (max)
Unknown
Autograd detail
Gradient
Activation
Optimizer state
Parameter

Mitigation strategies

How can we tackle the resource requirements?

AI
SWEDEN

Outline

AI SWEDEN

- Single GPU strategies:
  - Activation recomputation
  - Flash attention
  - Gradient accumulation
  - Mixed precision
- Multi GPU strategies:
  - Data Parallelism
  - Next lecture: Model parallelism
    - Tensor & Pipeline parallelism
    - FSDP

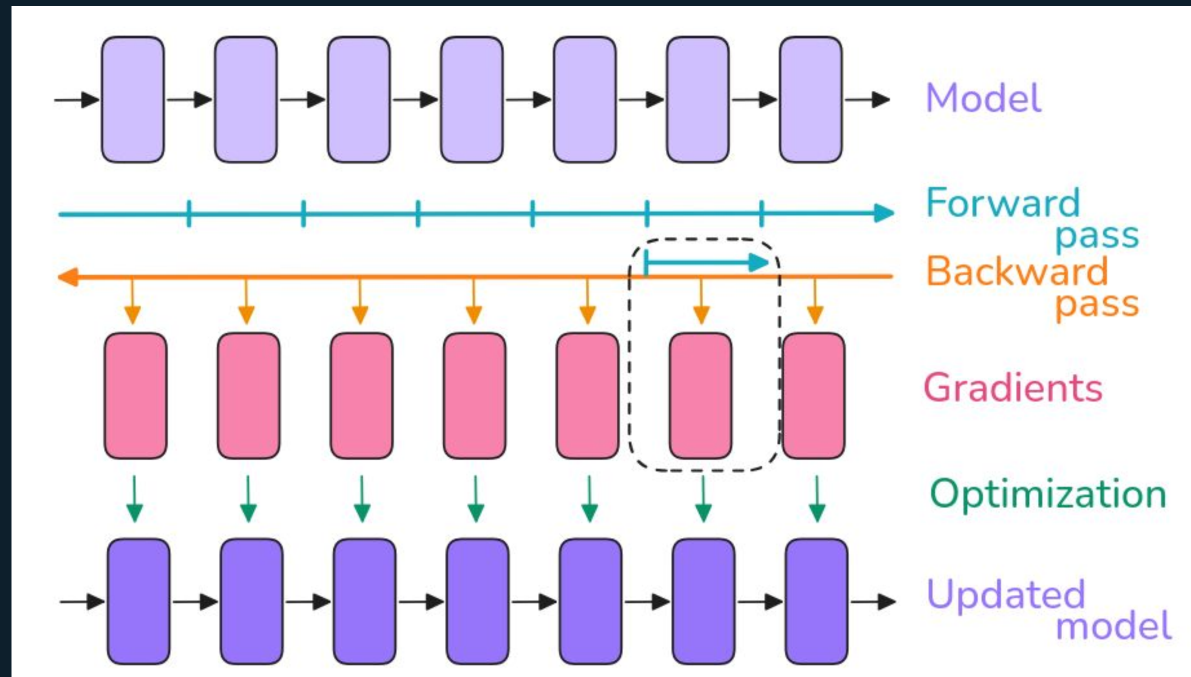
Activation recomputation

- Do we really need to store all those activations?

Activation recomputation

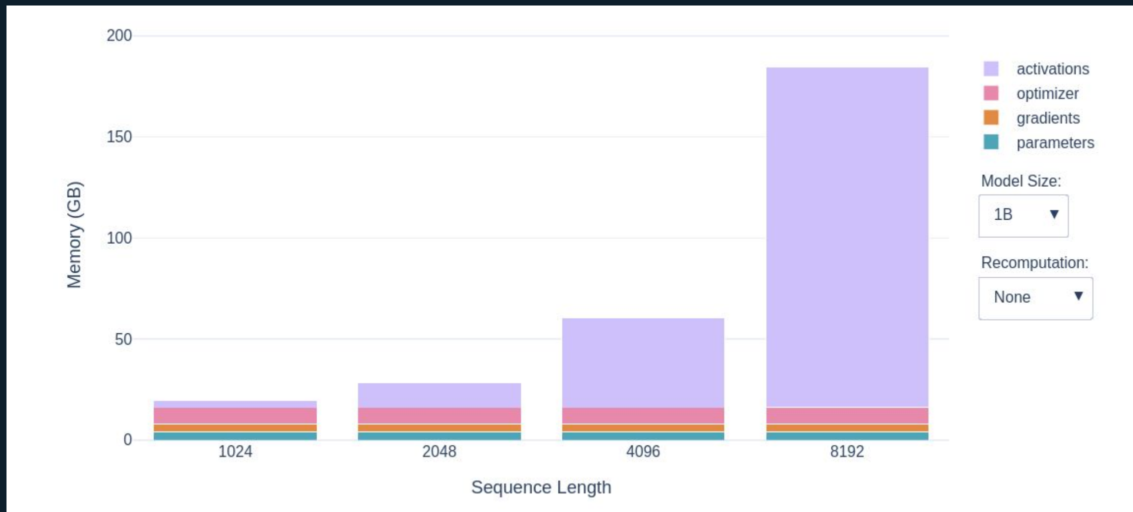
AI SWEDEN

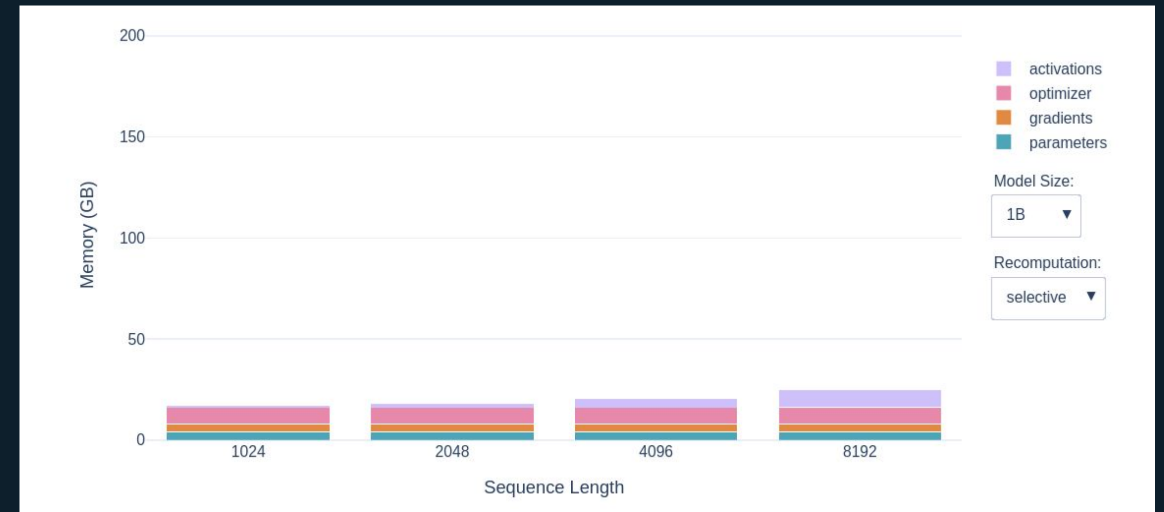
- Do we really need to store all those activations? No!
- Recompute activations the backward pass.
- E.g. Don't save the attention activations.



Activation recomputation

AI SWEDEN





Activation recomputation

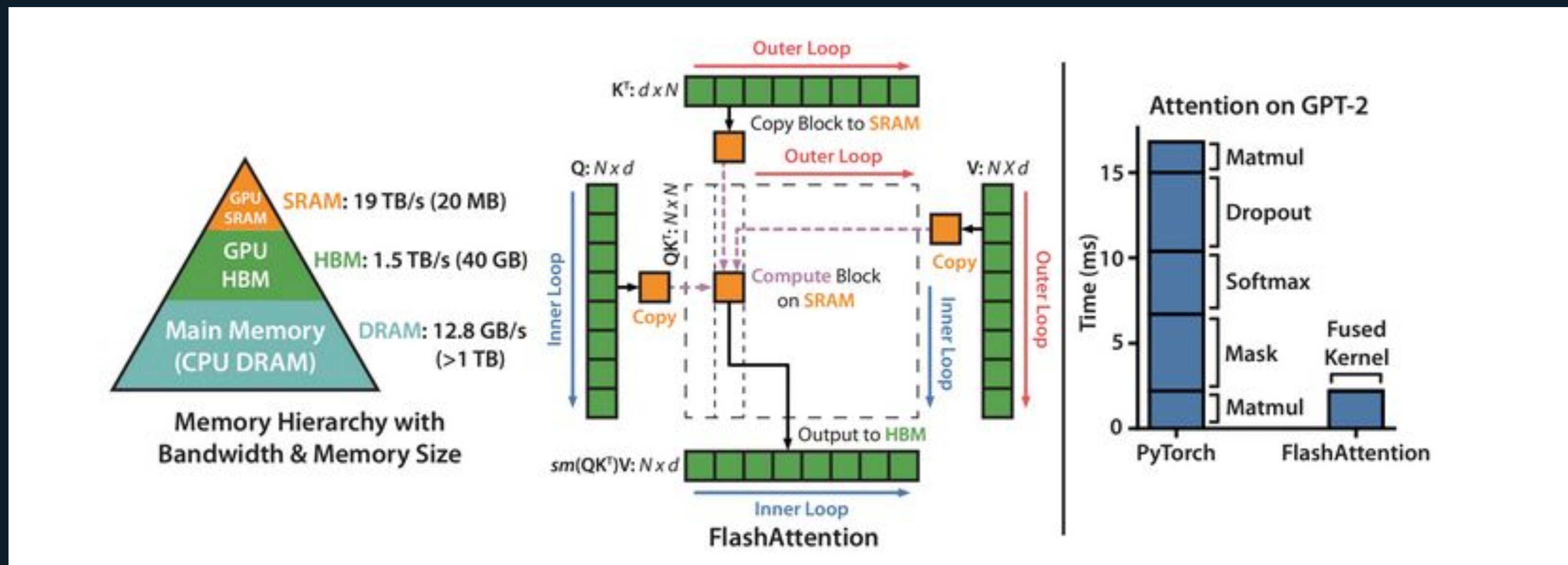
Activation recomputation enables to effectively pay extra compute to reduce memory requirements.

Note: not all activations are equal!

For more: **Reducing Activation Recomputation in Large Transformer Models (https://arxiv.org/abs/2205.05198)**

Flash attention

- Avoid materializing the attention matrix.
- Uses activation recomputation.
- You’re probably already using it.



Flash attention

AI SWEDEN

Layer	FLOPS	Parameters	Activations
MLP	$O(B \times S \times D^2)$	$O(D^2)$	$O(B \times S \times D)$
Attention	$O(B \times S \times D \times (S + D))$	$O(D^2)$	$O(B \times S \times (D + S))$
Transformer	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times B \times S \times (D + S))$

Layer	FLOPS	Parameters	Activations
MLP	$O(B \times S \times D^2)$	$O(D^2)$	$O(B \times S \times D)$
Attention (flash)	$O(B \times S \times D \times (S + D))$	$O(D^2)$	$O(B \times S \times D)$
Transformer (flash)	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times B \times S \times D)$

Flash attention

AI
SWEDEN

Param	FLOPS	Parameters	Activations
B	Linear		Linear
S	Quadratic		Linear
D	Quadratic	Quadratic	Linear
L	Linear	Linear	Linear

Linear Attention?

AI SWEDEN

- If we want to train a model over a fixed amount of tokens T, how does total compute scale w.r.t. sequence length?

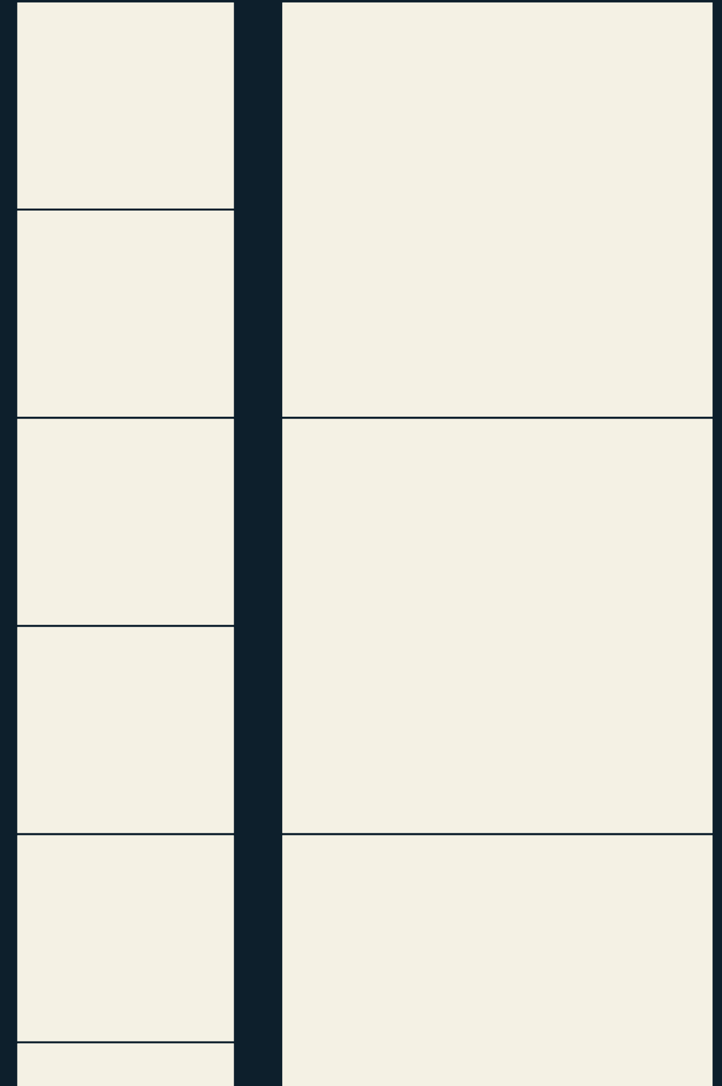
Linear Attention?

AI SWEDEN

- If we want to train a model over a fixed amount of tokens T, how does total compute scale w.r.t. sequence length?
- Per sample scaling is quadratic.

Linear Attention?

- If we want to train a model over a fixed amount of tokens T, how does total compute scale w.r.t. sequence length?
- Per sample scaling is quadratic.
- Per token scaling is linear.
- Longer sequence length => fewer samples.

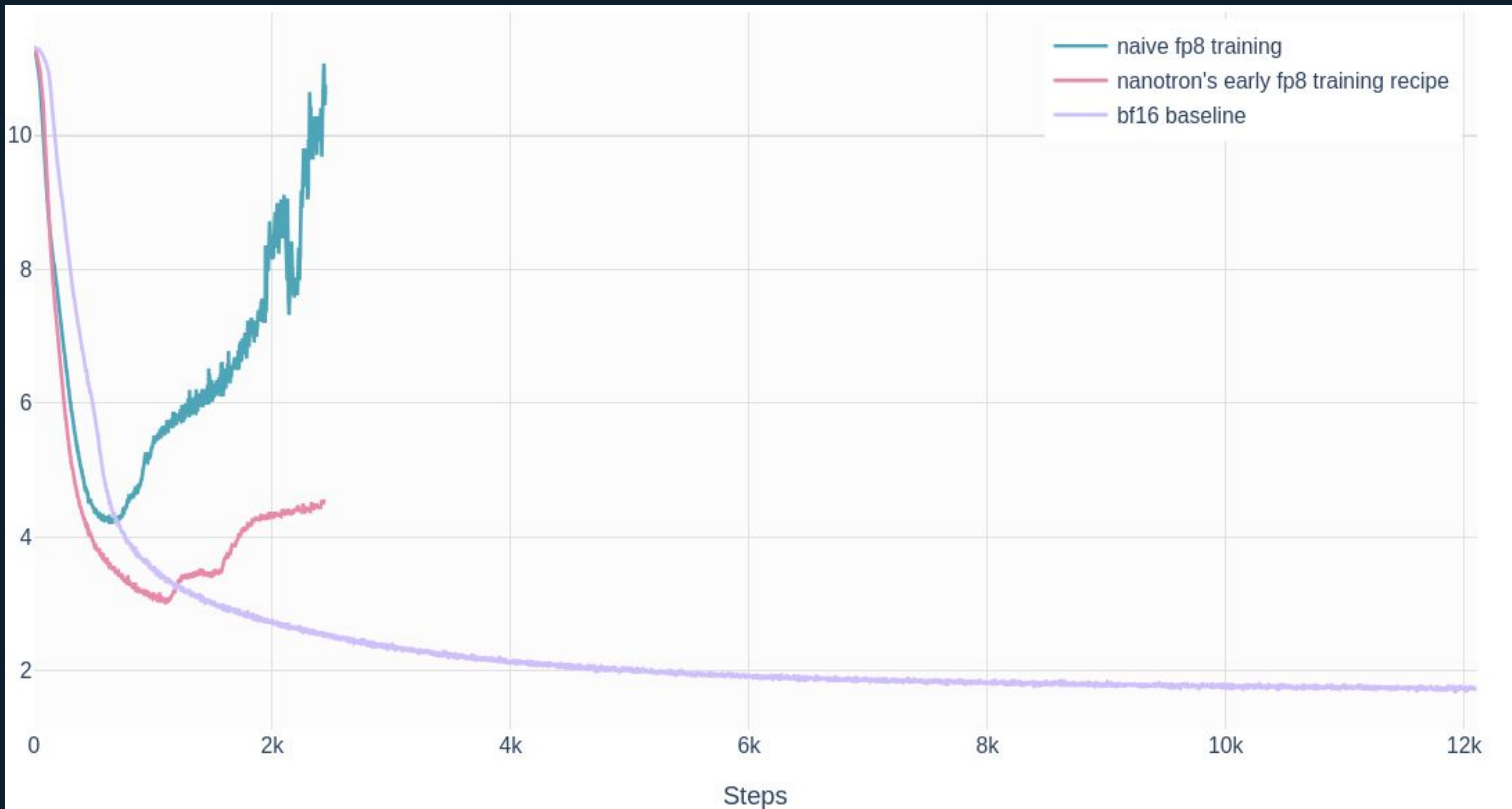


Mixed precision

AI SWEDEN

- **FP32 uses 4 bytes per value. We can reduce precision to improve both memory footprint and training speed.**

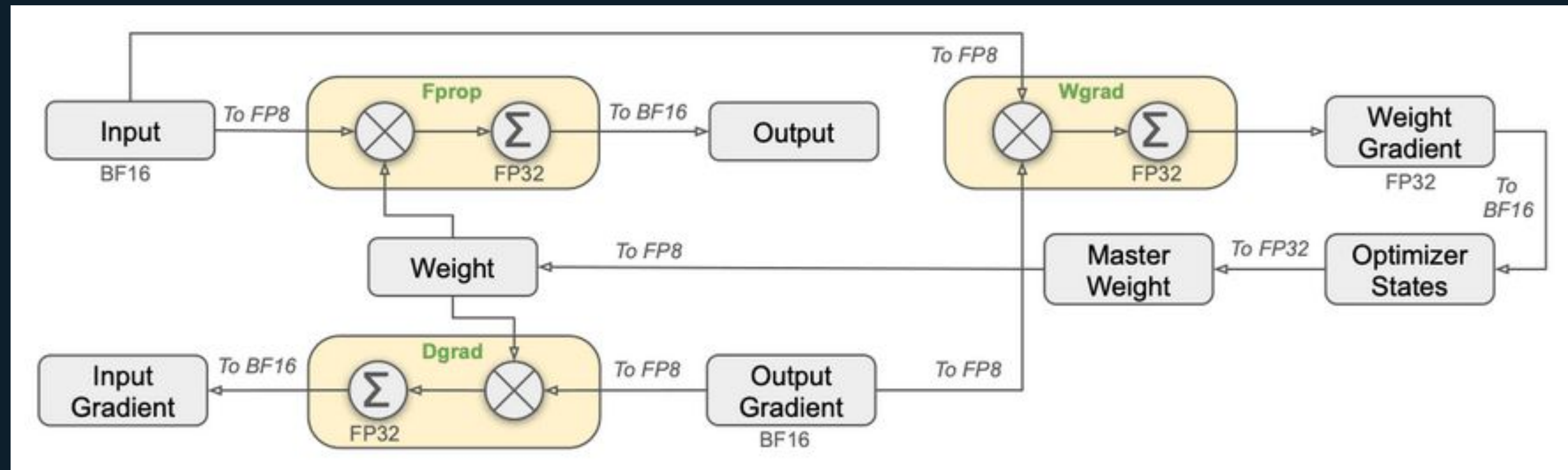
Tensor Float 32 (TF32)	**156 TFLOPS | 312 TFLOPS***
BFLOAT16 Tensor Core	**312 TFLOPS | 624 TFLOPS***
FP16 Tensor Core	**312 TFLOPS | 624 TFLOPS***
INT8 Tensor Core	**624 TOPS | 1248 TOPS***


naive fp8 training
nanotron's early fp8 training recipe
bf16 baseline
10
8
6
4
2
0
2k
4k
6k
8k
10k
12k
Steps

Mixed precision

AI
SWEDEN

- FP32 uses 4 bytes per value. We can reduce precision to improve both memory footprint and training speed.
- Lots of strategies.
- Very active field of research/engineering.
- Hardware sensitive.

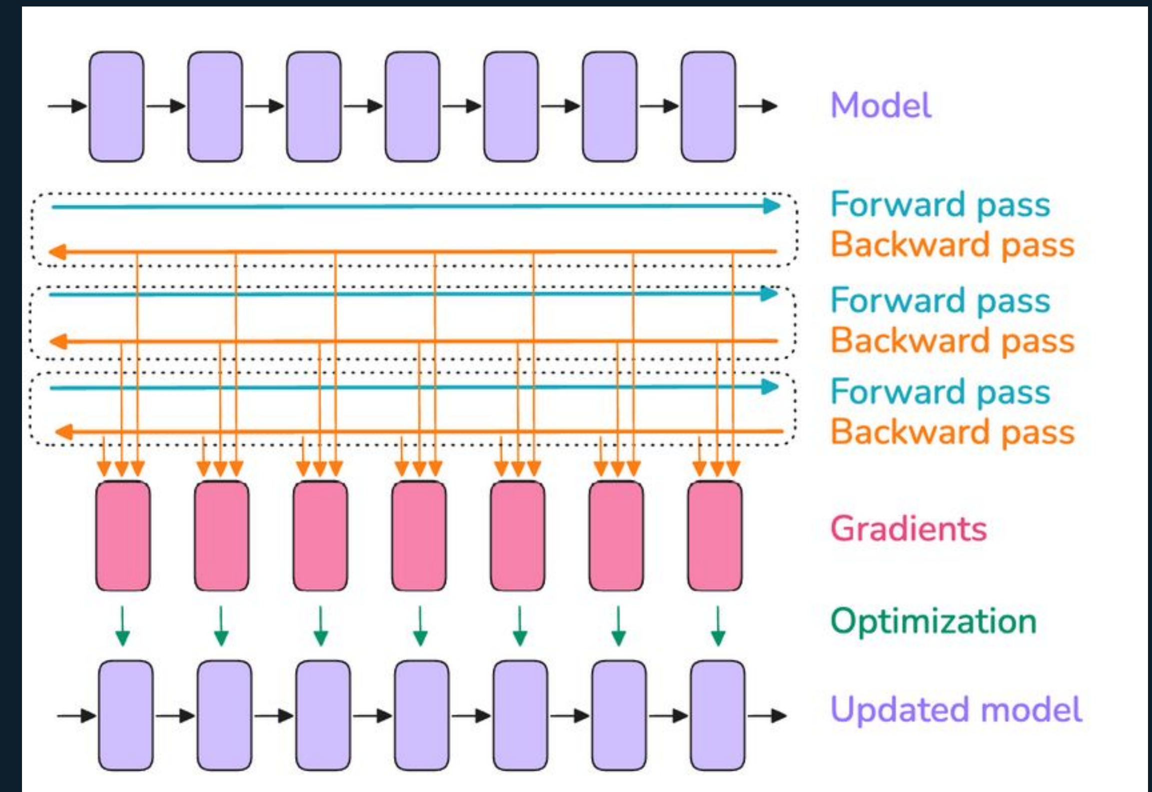


From DeepSeek-V3 Technical Report:
https://arxiv.org/abs/2412.19437v1

Gradient accumulation

AI SWEDEN

- Idea: split up your batch into micro-batches.
- Activation footprint scales by micro-batch size b, rather than global batch size B.
- Possibly down to b=1



Gradient accumulation

AI SWEDEN

- Idea: split up your batch into micro-batches.
- Activation footprint scales by micro-batch size b, rather than global batch size B.
- Possibly down to b=1

$$b = \frac{B}{\text{gradient accumulation steps}}$$

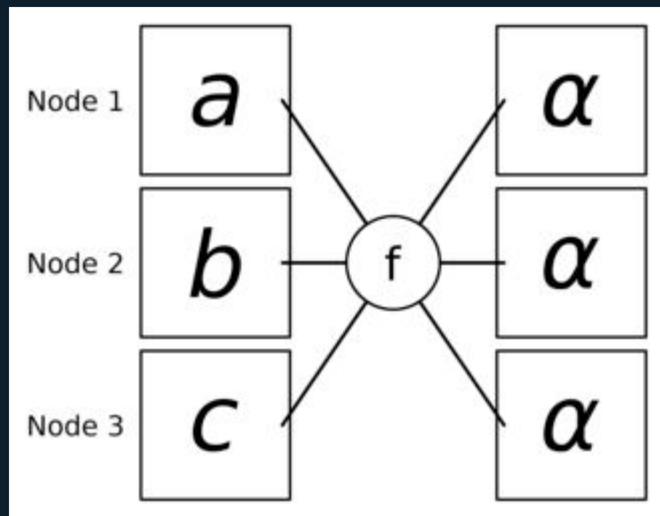
Param	FLOPS	Parameters	Activations
B	Linear		
b			Linear
S	Quadratic		Linear
D	Quadratic	Quadratic	Linear
L	Linear	Linear	Linear

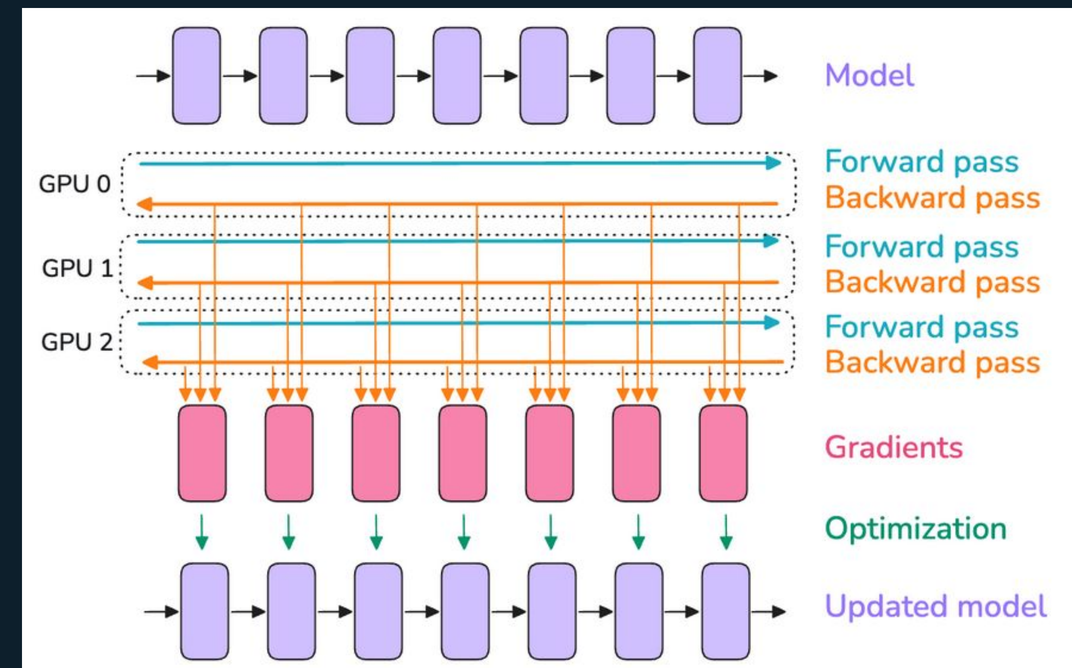
	FLOPS	Parameters	Activations
Transformer	$O(L \times B \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times b \times S \times D)$

Data parallelism

AI SWEDEN

- Idea: Essentially distributed gradient accumulation!
- One replica of the model lives on each GPU
- Each model replica computes the gradient on its own batch.
- Global gradient is computed using an all-reduce operation.





Data parallelism

AI SWEDEN

- Idea: Essentially distributed gradient accumulation!
- One replica of the model lives on each GPU
- Each model replica computes the gradient on its own batch.
- Global gradient is computed using an all-reduce operation.
- Can be combined with gradient accumulation.

$$b = \frac{B_D}{\text{gradient accumulation steps}}$$

$$B = B_D \times \text{Data Parallelism}$$

	FLOPS / device	Parameters / device	Activations / device
Transformer	$O(L \times B_D \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times b \times S \times D)$

Data parallelism

AI SWEDEN

- Does not reduce memory footprint*.
- *Does* increase training speed by enabling multi-GPU training.

$$b = \frac{B_D}{\text{gradient accumulation steps}}$$

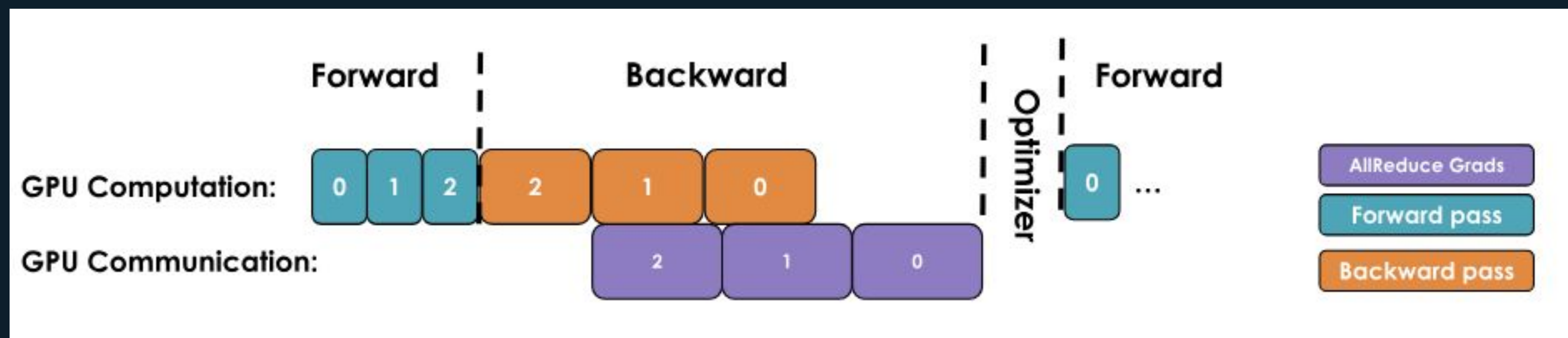
$$B = B_D \times \text{Data Parallelism}$$

	FLOPS / device	Parameters / device	Activations / device
Transformer	$O(L \times B_D \times S \times D \times (S + D))$	$O(LD^2)$	$O(L \times b \times S \times D)$

Data parallelism

AI SWEDEN

- Does not reduce memory footprint*.
- *Does* increase training speed by enabling multi-GPU training.
- *Does* come with communication overhead:
  - All-reduce requires communication and synchronization between devices.



Efficiency and training configuration.

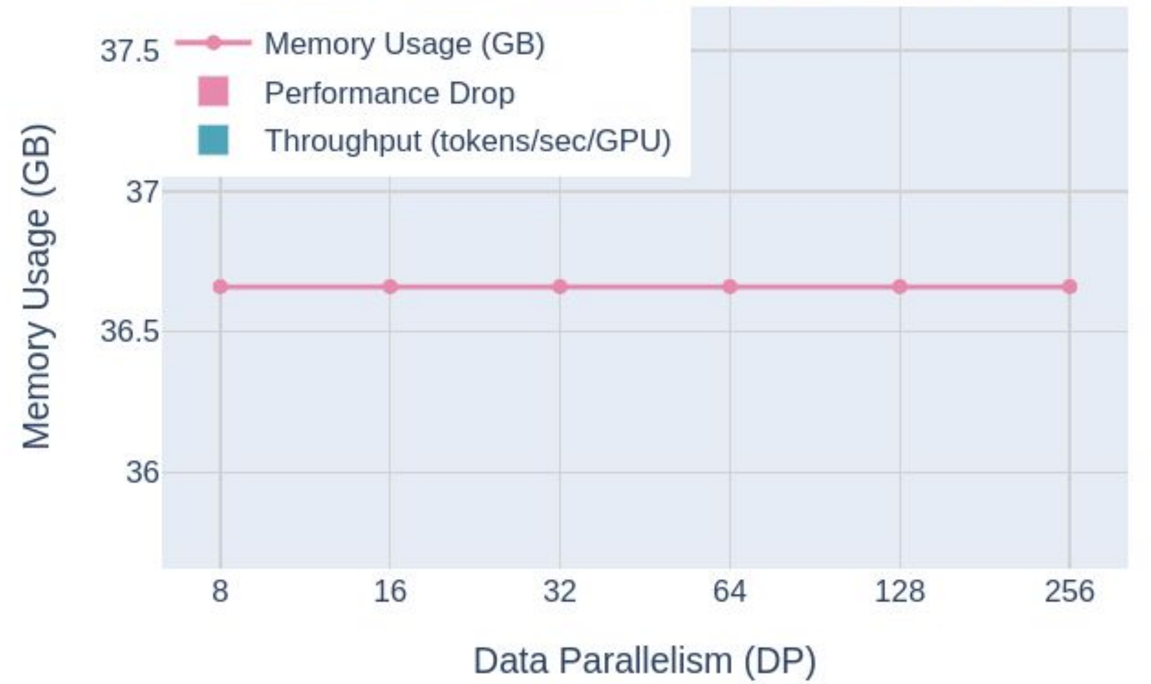
AI SWEDEN

Training run

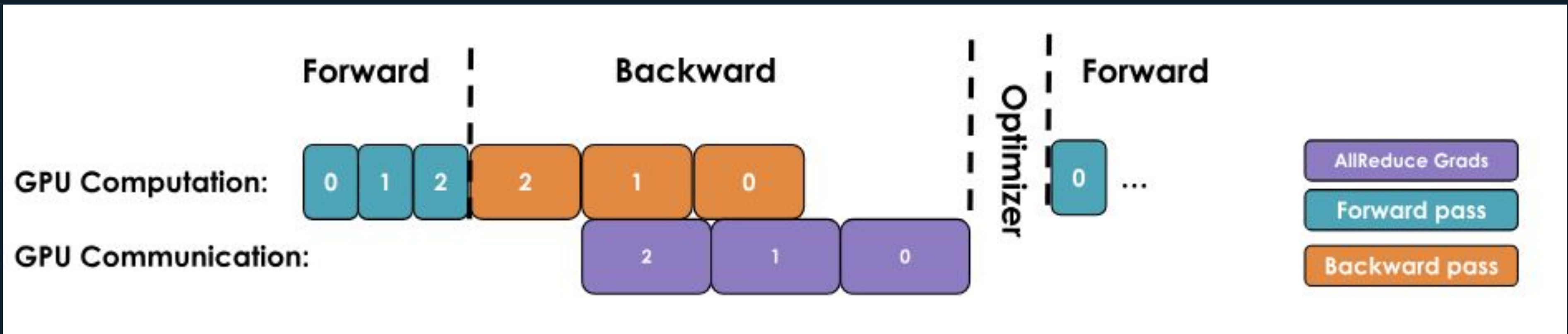
AI
SWEDEN

- Decide on model hyperparameters and number of training tokens.
- Based on the model size and available literature/scaling experiments, you can determine the optimal *Global* batch size (B).
    - (e.g. 4-60 million tokens per step)
- Based on the model hyperparameters, figure out the largest mini batch size (b) for one device before OOM*.
- DP * b * gradient accumulation = B.
- B * Sequence Length = Tokens/step
- More GPUs => Faster training*

Throughput Scaling with Data Parallelism
Throughput (tokens/sec/GPU)
40k
30k
20k
10k
0
-6.3%
-6.0%
-12.0%
-15.0%
-40.6%
8
16
32
64
128
256
Data Parallelism (DP)



Memory Usage Scaling with Data Parallelism
Memory Usage (GB)
Performance Drop
Throughput (tokens/sec/GPU)
37.5
37
36.5
36
8
16
32
64
128
256
Data Parallelism (DP)



Forward
Backward
Optimizer
Forward
GPU Computation:
0
1
2
2
1
0
0
...
GPU Communication:
2
1
0
AllReduce Grads
Forward pass
Backward pass

How to measure utilization?

AI SWEDEN

- Throughput: Tokens / second
  - Simple
  - But does not tell you how close to optimal we're running!

How to measure utilization?

AI SWEDEN

- Model FLOPs Utilization (MFU)
  - idea: Compare the peak FLOPs (PF) of your hardware to Model FLOPs.
  - Model FLOPs (MF): A theoretical lower bound on the necessary FLOPs (per training batch or token).

How to measure utilization?

AI SWEDEN

- FLOPs utilization!
  - PF: Peak FLOPs [FLOP/s]
  - MF: Model FLOPs [FLOP/token]
  - T: Throughput [token/s]
  - MFU: MF * T / PF

$$Model\ FLOPs = layers \times dim^2 \times \begin{pmatrix} 12 \\ + \left(12 \times \frac{query\ groups}{attention\ heads}\right) \\ + \left(18 \times \frac{dim_{ff}}{dim}\right) \\ + \left(12 \times \frac{sequence\ length}{dim}\right) \\ + \left(6 \times \frac{vocab\ size}{dim}\right) \end{pmatrix}$$

AI SWEDEN

How to measure utilization?

- MFU varies. 40-60% is usually *good*.

$$Model\ FLOPs = layers \times dim^2 \times \begin{pmatrix} 12 \\ + \left(12 \times \frac{query\ groups}{attention\ heads}\right) \\ + \left(18 \times \frac{dim_{ff}}{dim}\right) \\ + \left(12 \times \frac{sequence\ length}{dim}\right) \\ + \left(6 \times \frac{vocab\ size}{dim}\right) \end{pmatrix}$$

Practice

AI
SWEDEN

- 1B model
- 4096 sequence length
- 4 A100s