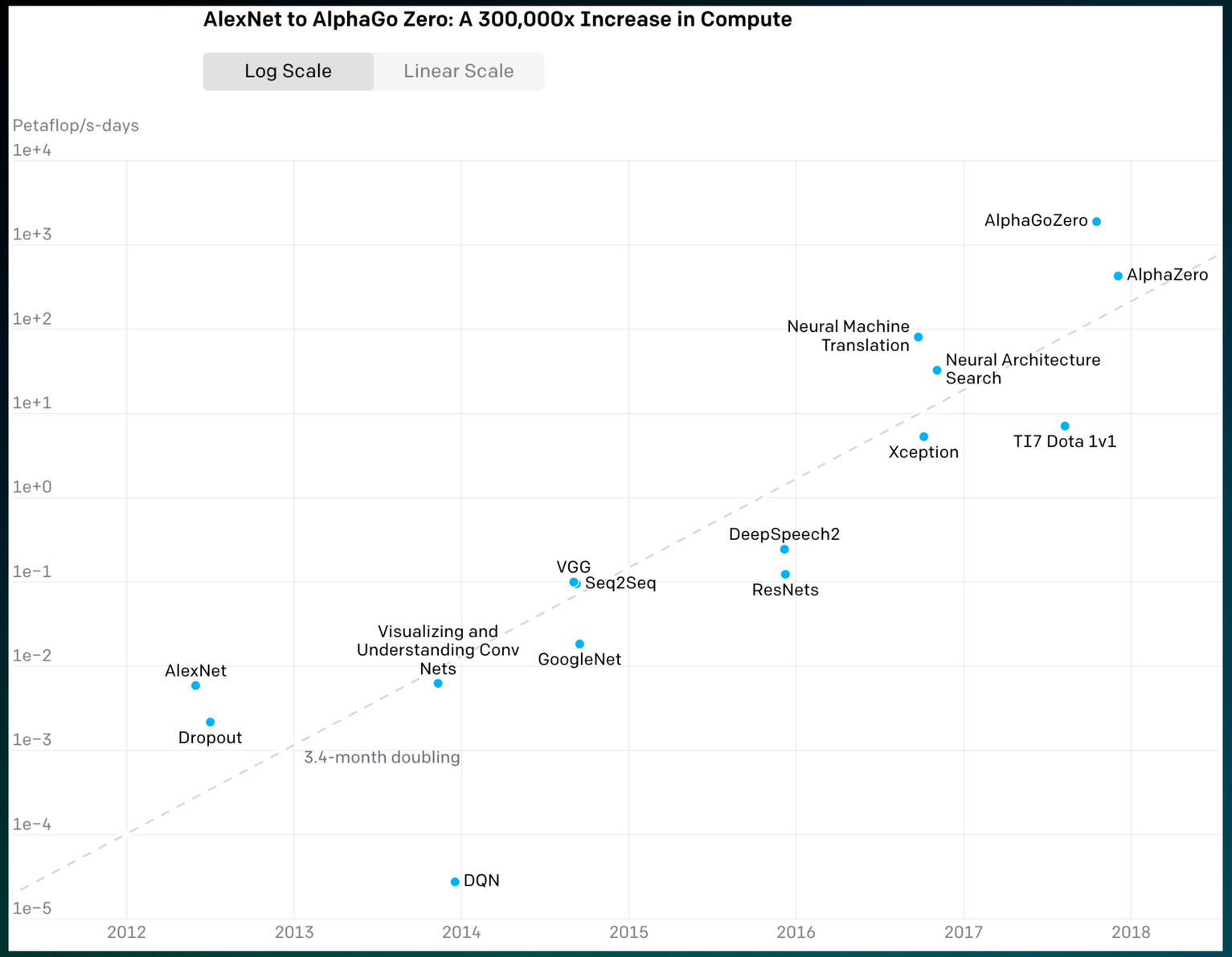
Optimizing Large-Scale GPT Models Pretraining

Oleh Shliazhko, NLP R&D, SberDevices

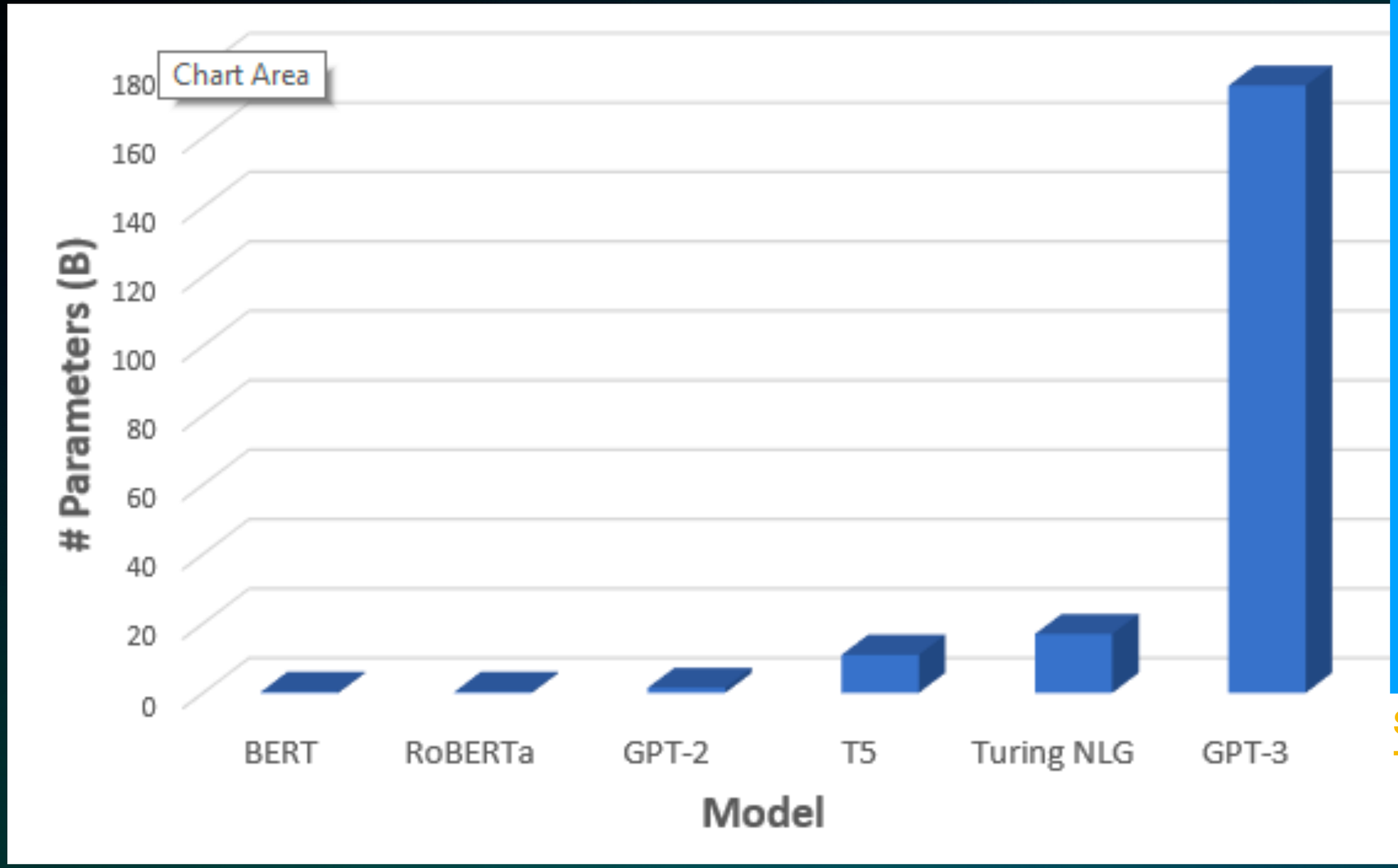
Large Scale Deep Learning NLP Models



1. **Amount of SOTA compute doubling each 3.4 months**
2. **And this is ain't gonna stop anytime soon**

SBER DEVICES

Large Scale Deep Learning NLP Models



GPT-2 – 1.5 Billion weights
Megatron LM – 8.3 Billion
T5 – 11 Billion
Turing NLG – 17 Billion
GPT-3 – 175 Billion

Switch Transformer - 1.6 TRILLION 😱 😱 😱

Switch Transformer

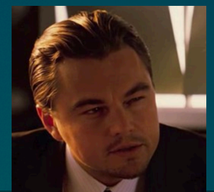
https://arxiv.org/pdf/2101.03961.pdf

Large Scale Modeling limited by:

- Memory constraints
  - Model size
  - Batch size
  - Optimizer choice
- Time constraints
  - Lot of samples
  - Long step time

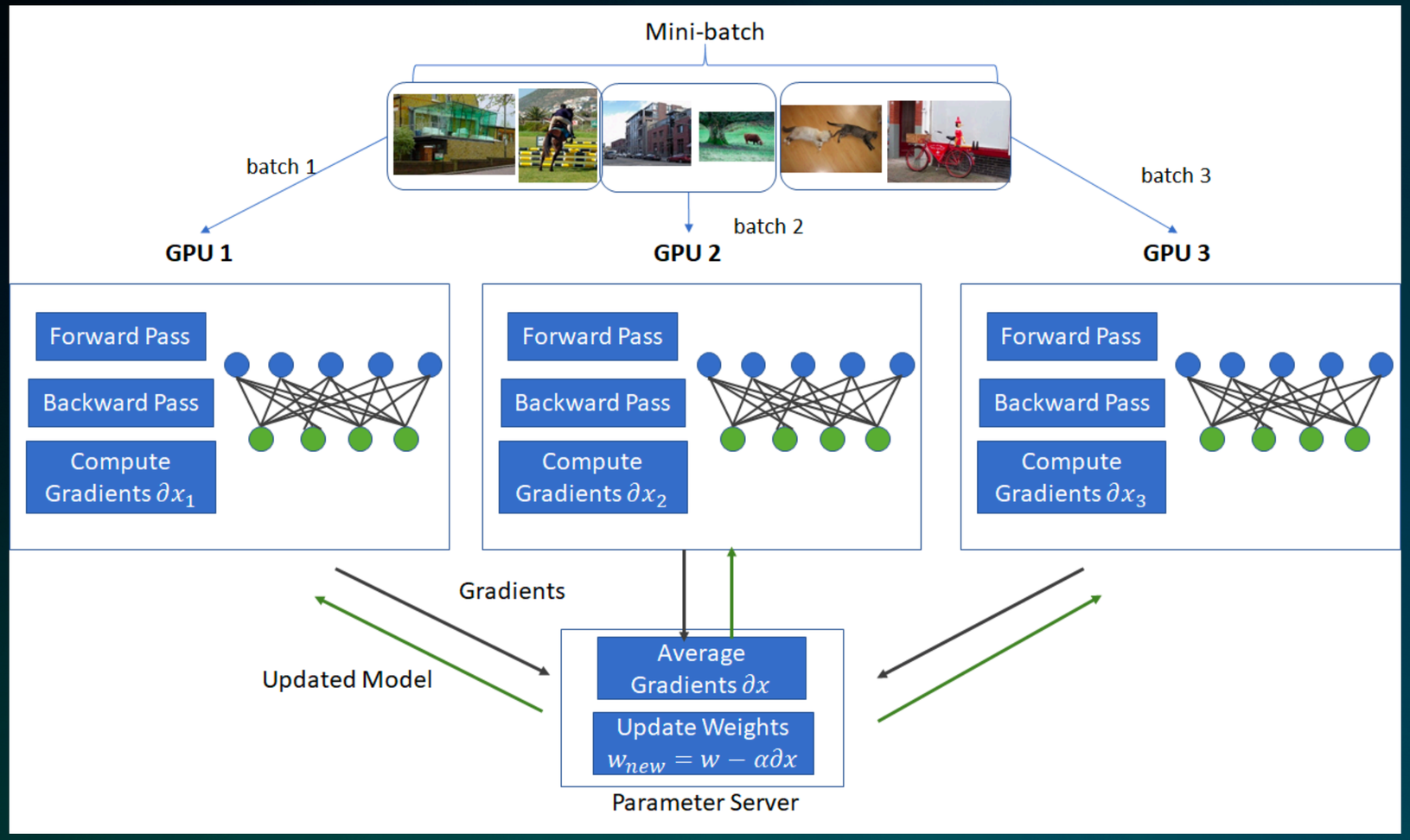
Our path to 13B GPT-3

1. GPT-2 small, 117M
2. GPT-2 small, + data parallel
3. GPT-2 medium, 345M
4. GPT-3-scale dataset
5. GPT-3 medium 345M
6. GPT-3 large, 750M, + deepspeed optimizer
7. GPT-3 sparse layer selection
8. GPT-3 XL, 1.3B, + checkpoint activations
9. GPT-3 13B, + model parallel
10. ....We need to go Deeper



Large Scale Deep Learning NLP Models

Usual workflow, distributed Data-Parallel Training



Model replicated on each GPU
Optimizer state replicated on each GPU

Memory constraints

Modern GPU Memory constraints:

Nvidia V100 - 32GB
Nvidia 2080TI - 11GB

.

.

Nvidia A100 - 80GB 🙏

Weights → Bytes approximation

Training with FP32 Adam:

32 bit weights + 32 × 2 optimizer state =

12 bytes

Training with FP16 Mixed precision:

16 weights + 16 × 2 + 32 × 2 optimizer =

14 bytes!

Training with pure FP16 training:

16 weights + 16 × 2 optimizer = 6 bytes

*But not available out of the box 😞

Upper bound model size

+ some memory for input batches (megabytes)
+ some memory for activations (few gigabytes)

Practical upper bound of GPT model size per GPU:

- V100 - 2 billion weights
- 2080TI - 700 million weights

Data parallelism is not enough.

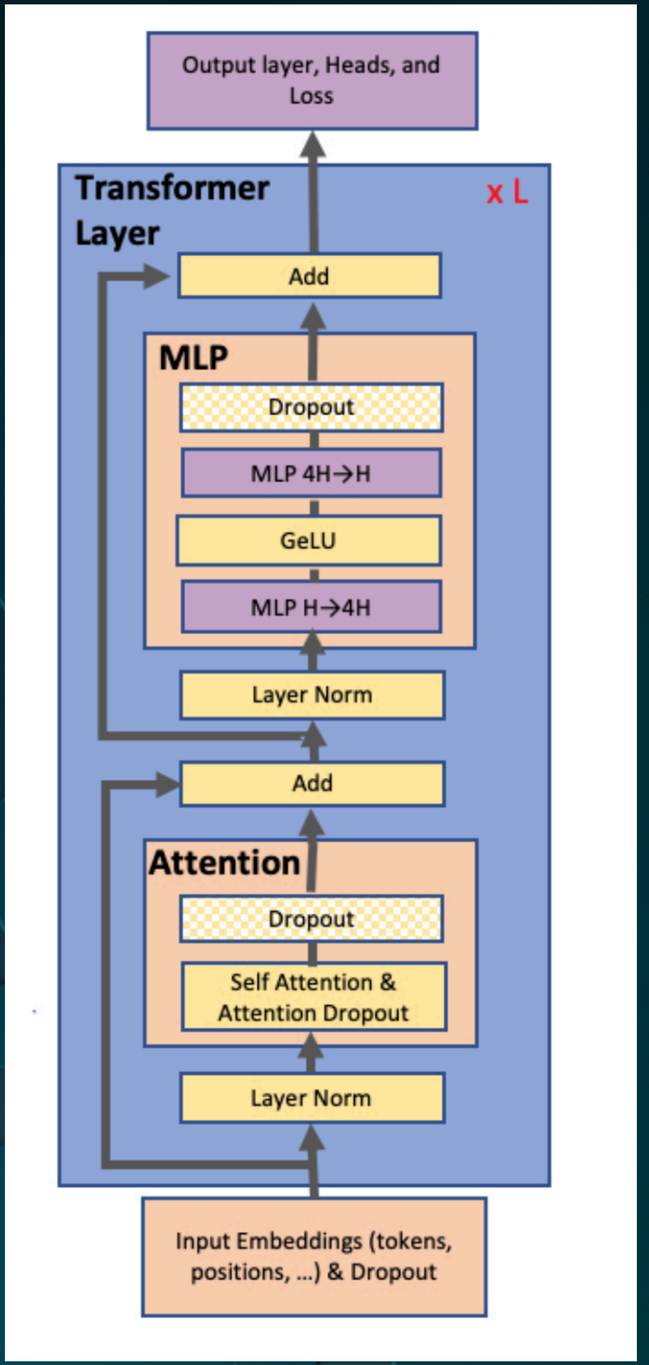
Ways to go:

- Split the model between GPUs
- Split the optimizer between GPUs
- Recompute activations instead of store

Model parallelism

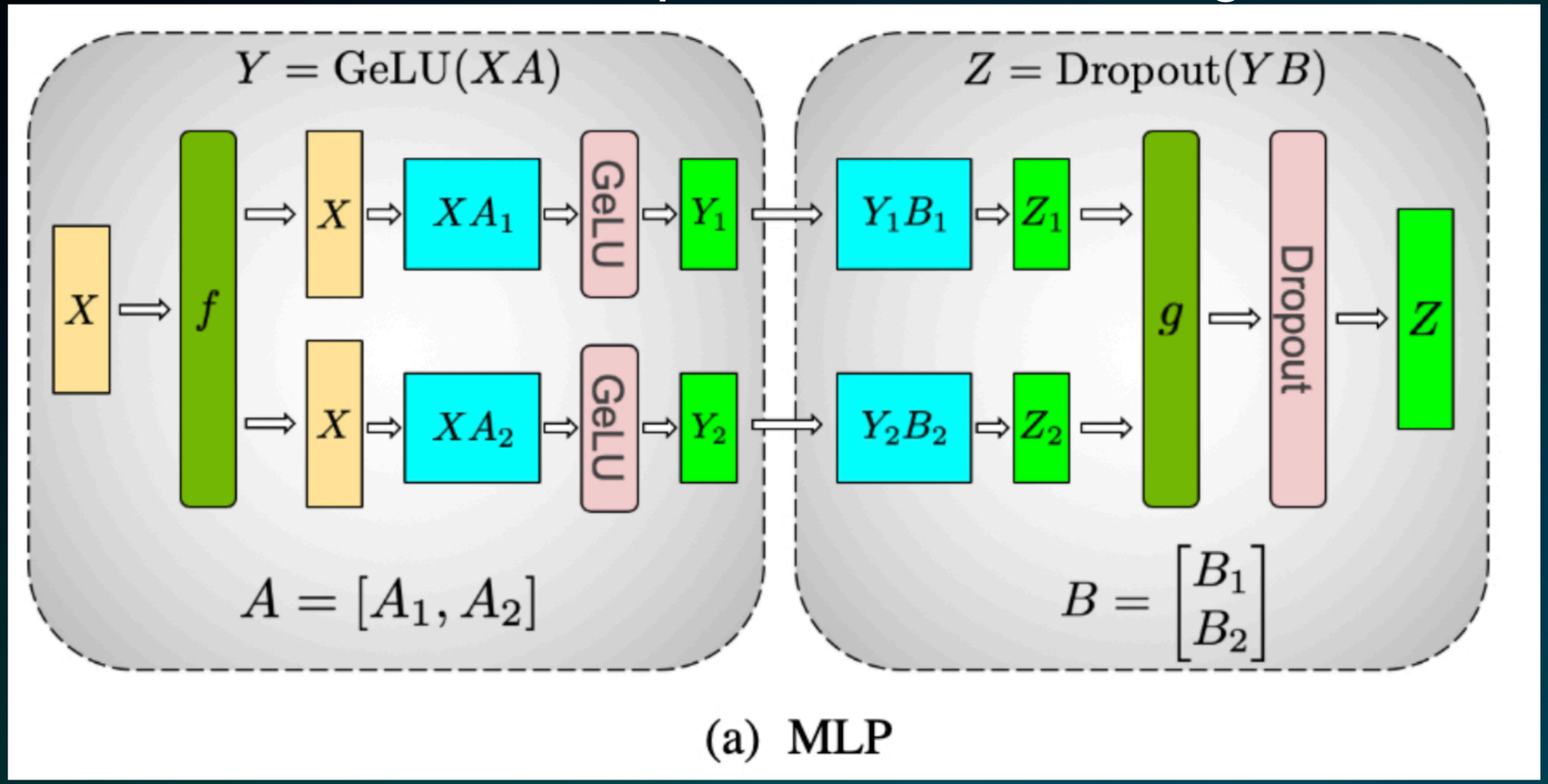
- Pipeline (layer-wise) parallelism
- Distributed tensor computation

Transformer Reminder



Split the model between GPUs

Distributed tensor computation. Nvidia Megatron

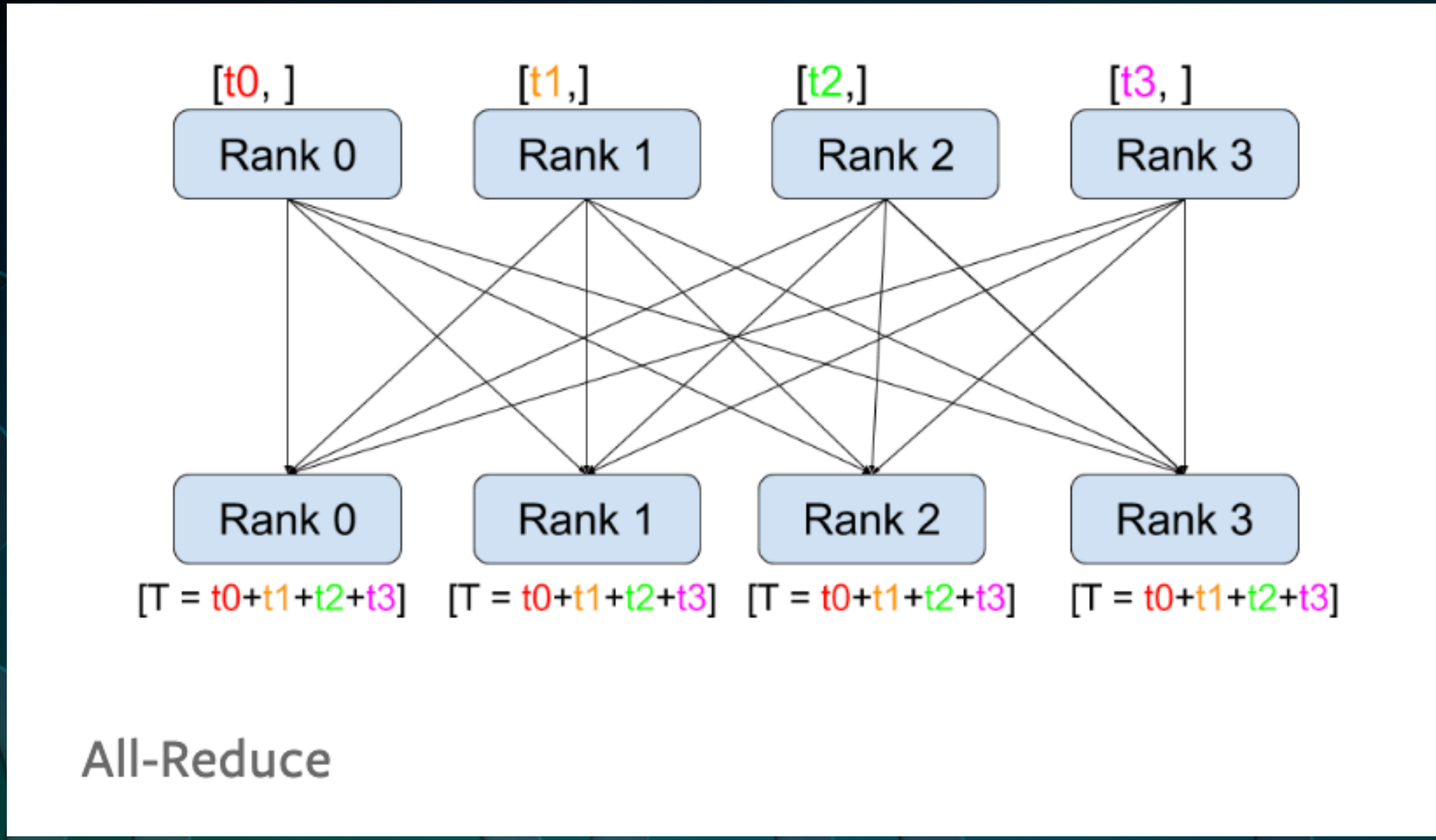


(a) MLP

```python
class f(torch.autograd.Function):
    def forward(ctx, x):
        return x
    def backward(ctx, gradient):
        all_reduce(gradient)
        return gradient
```

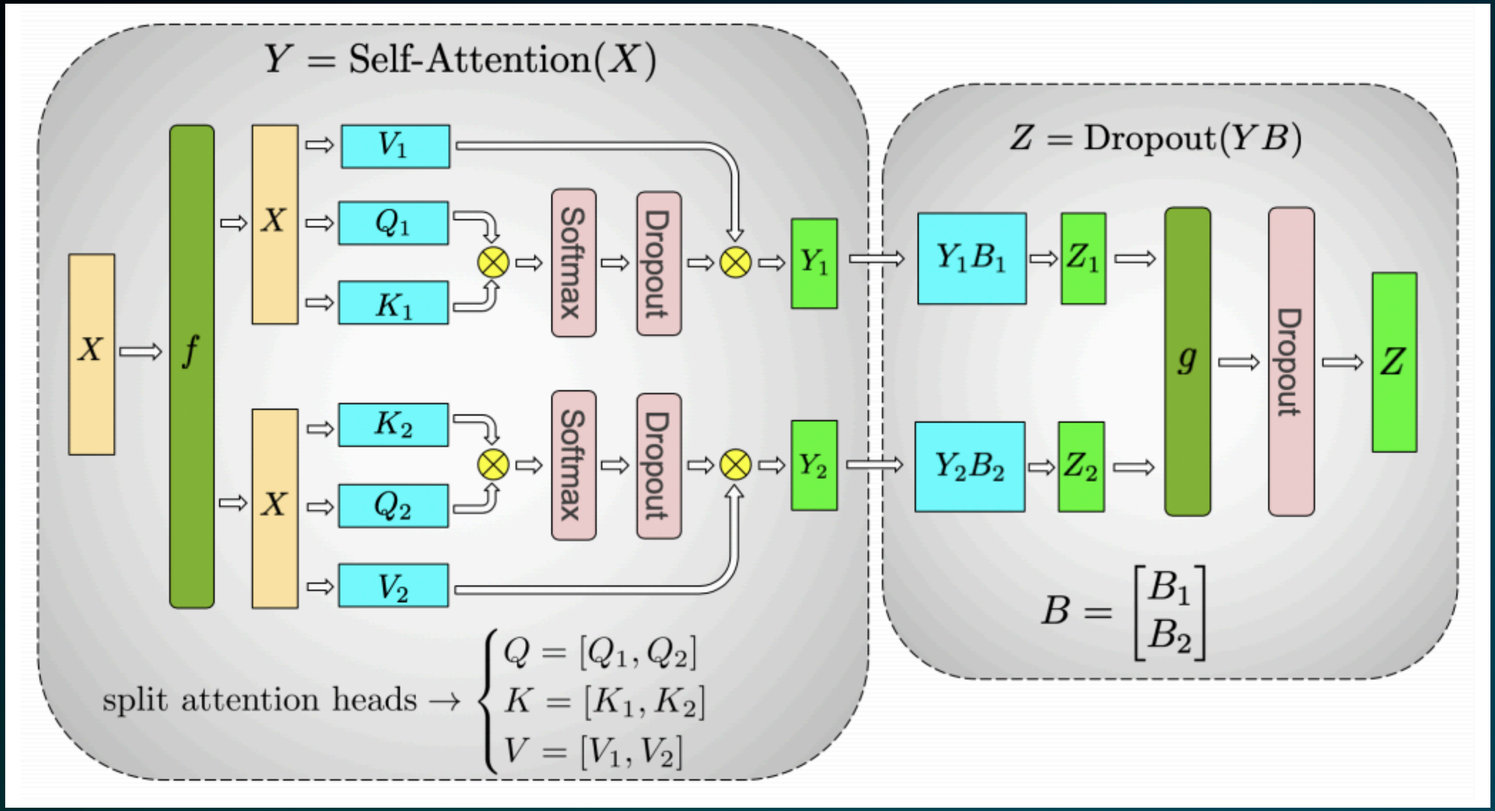

https://arxiv.org/pdf/1909.08053.pdf

All-Reduce Reminder



Split the model between GPUs

Distributed tensor computation. Nvidia Megatron

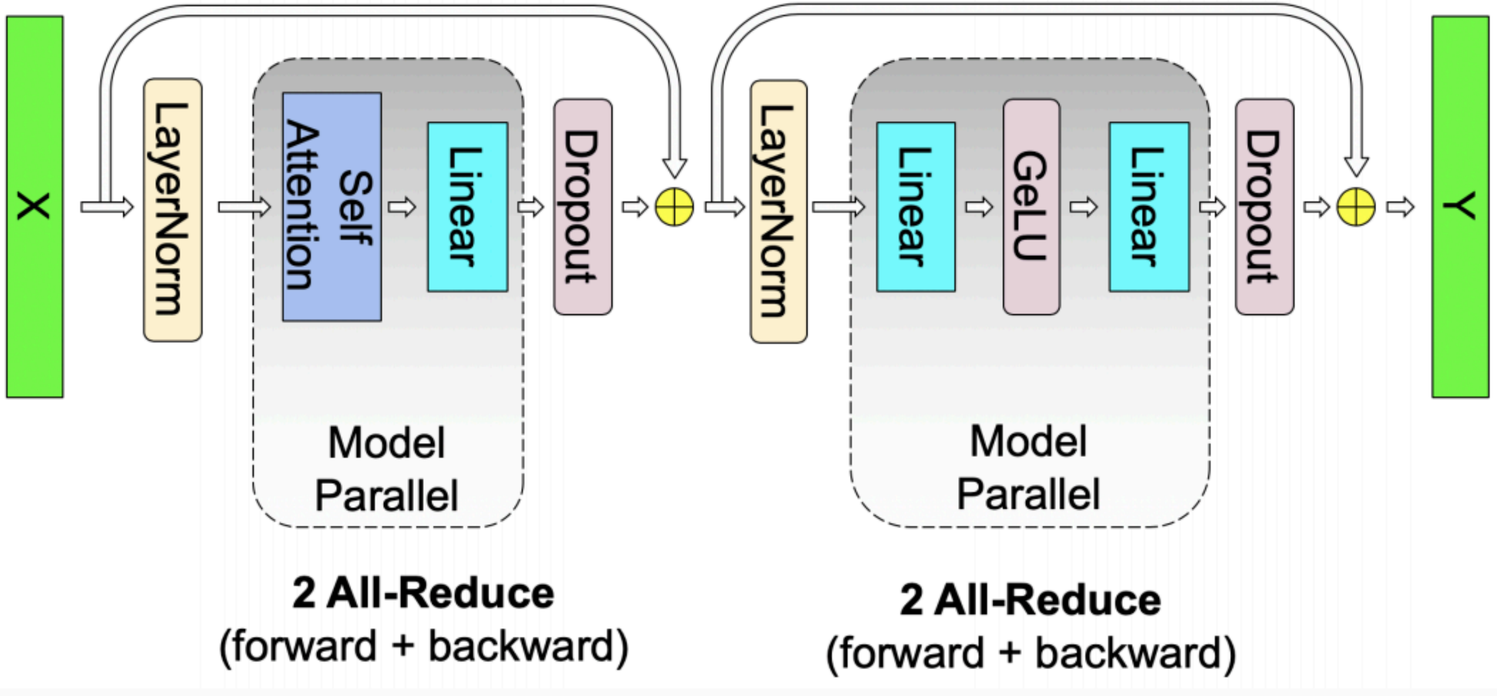


Partitioning the Q, K, V in a column parallel fashion such that the matrix multiply corresponding to each attention head is done locally on one GPU.

https://arxiv.org/pdf/1909.08053.pdf

Split the model between GPUs

Distributed tensor computation. Nvidia Megatron



https://arxiv.org/pdf/1909.08053.pdf

Split the optimizer between GPUs

Microsoft ZERO

1. Optimizer State Partitioning (Pos) – 4x memory reduction, same communication volume as data parallelism. Update locally, then all_gather.

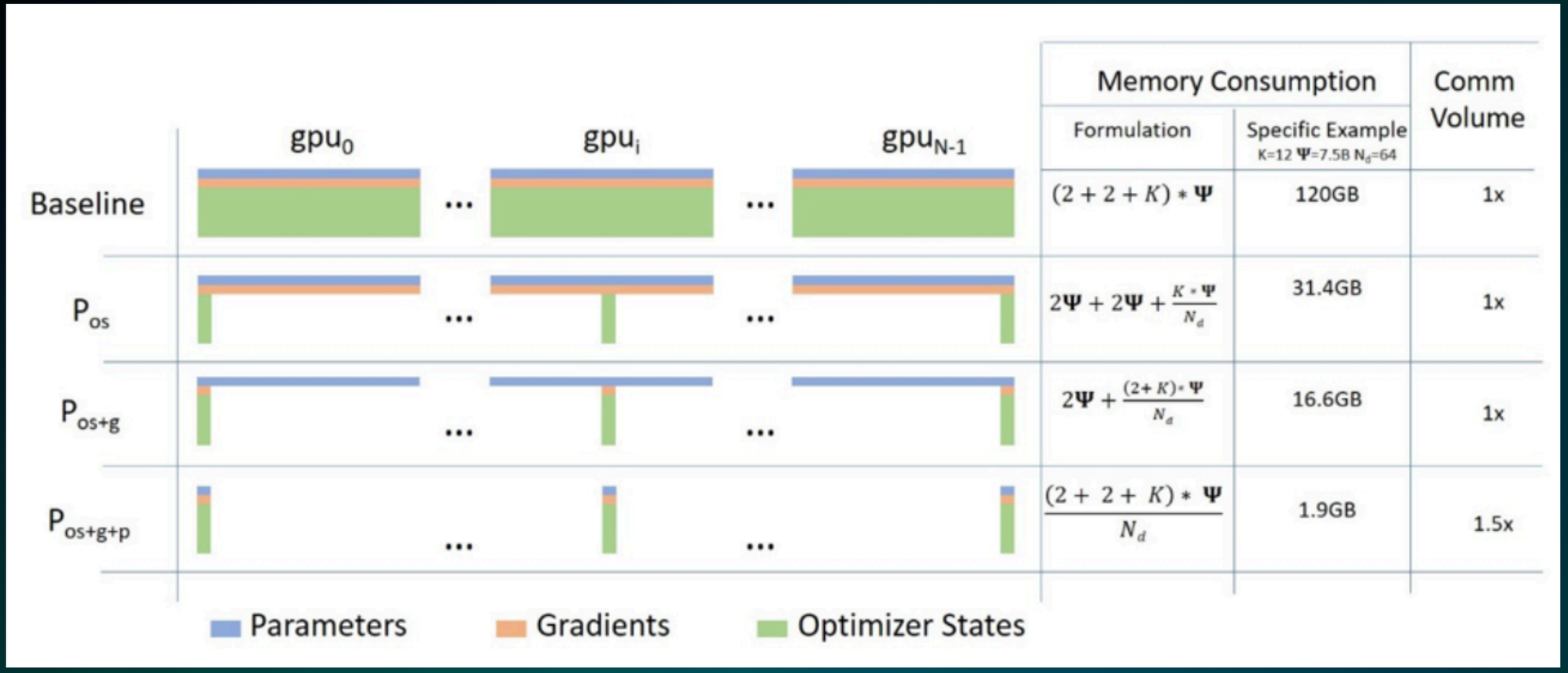
2. Add Gradient Partitioning (Pos+g) – 8x memory reduction, same communication volume as data parallelism

3. Add Parameter Partitioning (Pos+g+p) – Memory reduction is linear with data parallelism degree Nd.

https://arxiv.org/pdf/1910.02054.pdf

Split the optimizer between GPUs

Microsoft ZERO



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

https://arxiv.org/pdf/1910.02054.pdf

Split the optimizer between GPUs

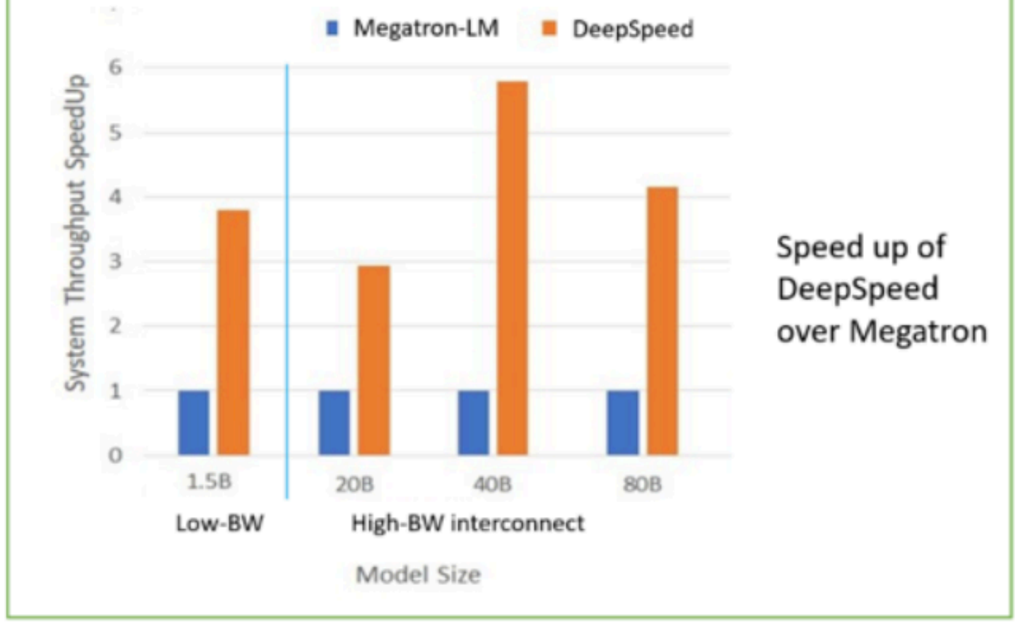
Microsoft ZERO & Deepspeed

Scale
- 100B parameter
- 10X bigger

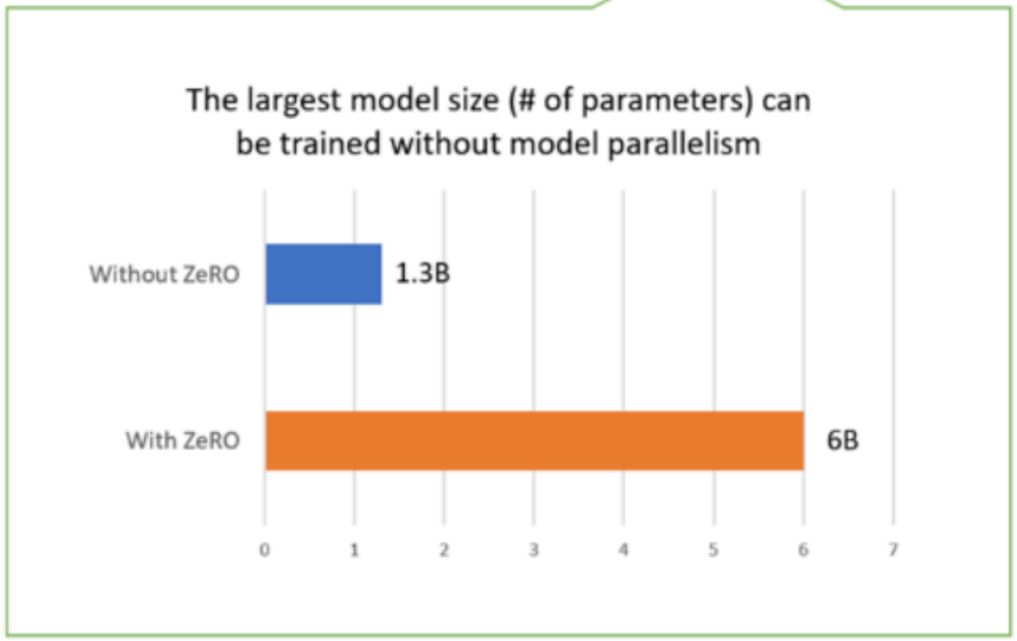
Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

Usability
- Minimal code change



Speed up of DeepSpeed over Megatron



https://arxiv.org/pdf/1910.02054.pdf

Activation Checkpointing

- saves memory
- store only the activations of each *Nth* layer
- recompute others during backprop

https://arxiv.org/pdf/1904.10631.pdf

Time constraints

Self-attention has quadratic (N × N) complexity.

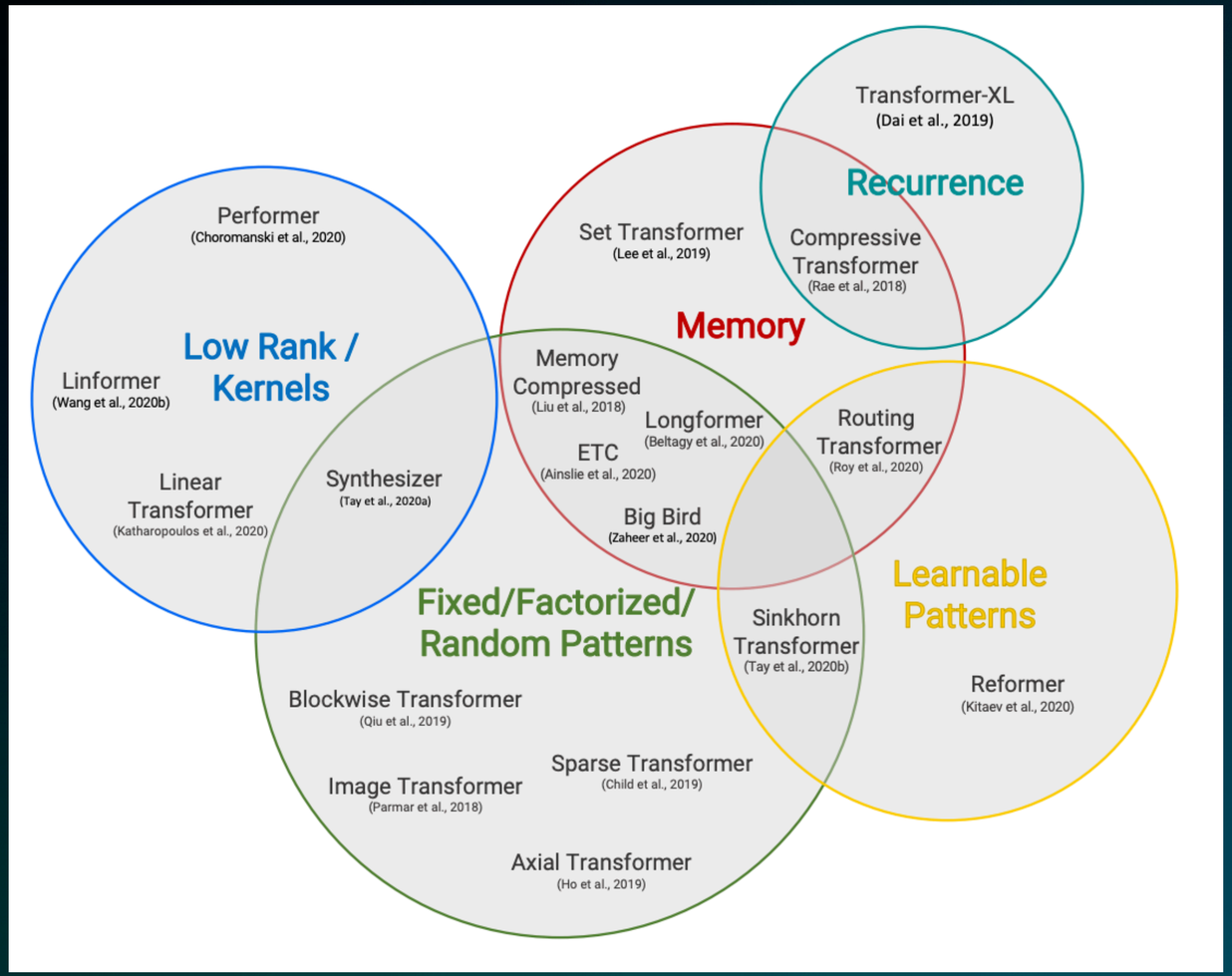
Most of the compute time of transformer model
spent in attention matrix computation.

Attention as weights of components - only few biggest matter most.

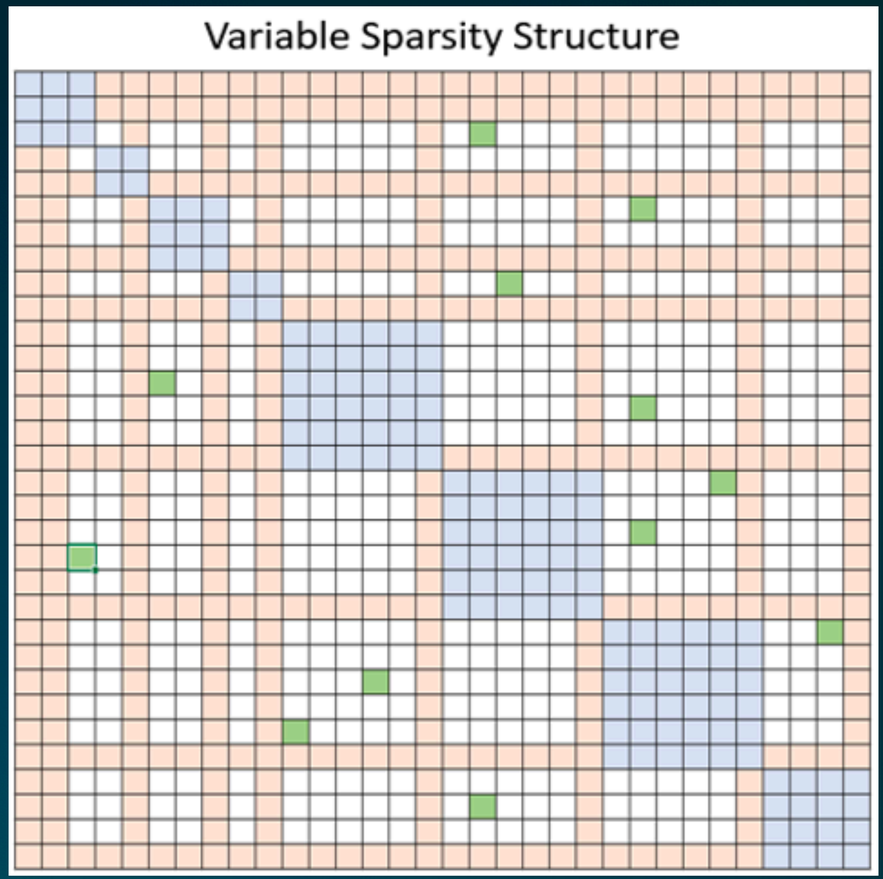
So, lets approximate this matrix with something faster.

https://arxiv.org/pdf/2009.06732.pdf

Time constraints



Exploit inherent attention matrix sparsity



https://arxiv.org/pdf/2009.06732.pdf

Thank you!