Microsoft

deepspeed

Enabling efficient trillion parameter scale training for deep learning models

https://github.com/microsoft/DeepSpeed

Presented by: Olatunji (Tunji) Ruwase

On behalf of the DeepSpeed team

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

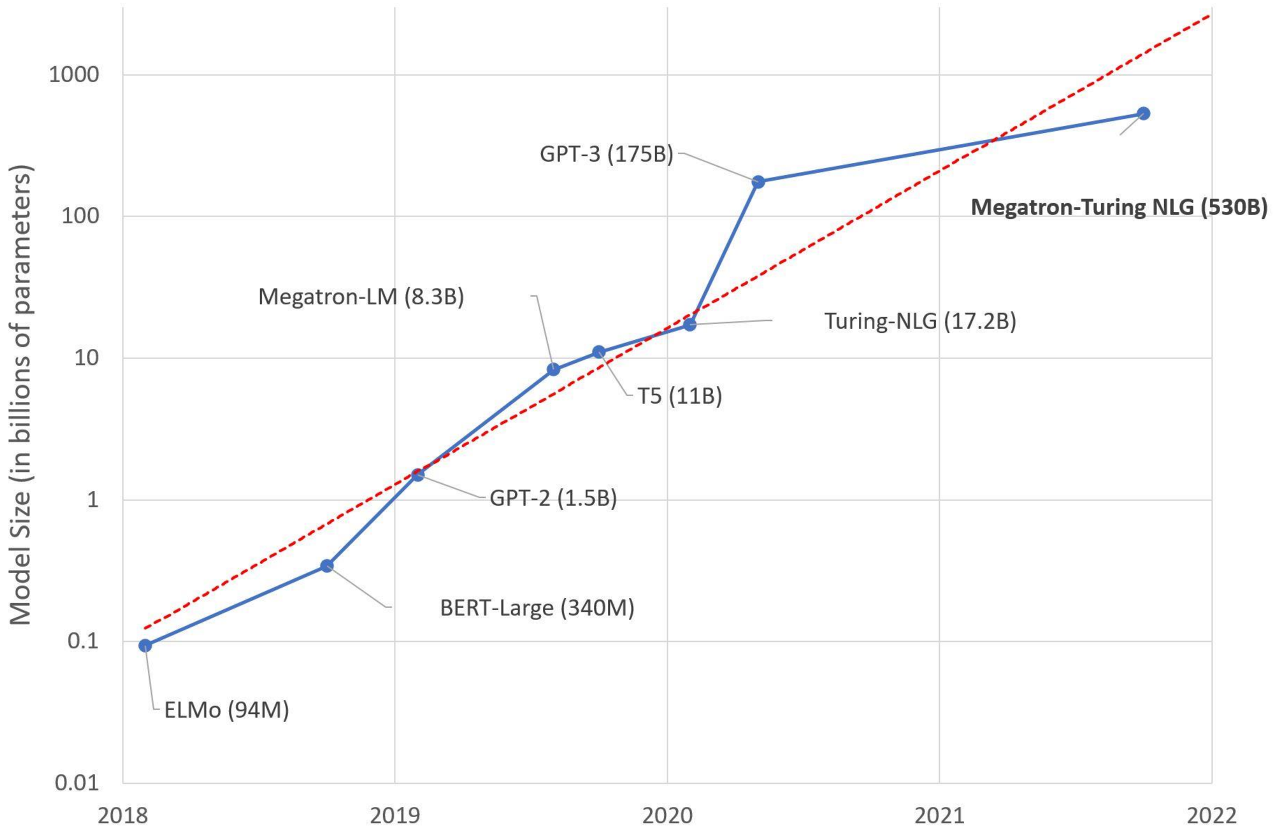
Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Faster & cheaper

Usability
- Few lines of code changes

Motivation: Why large language models?



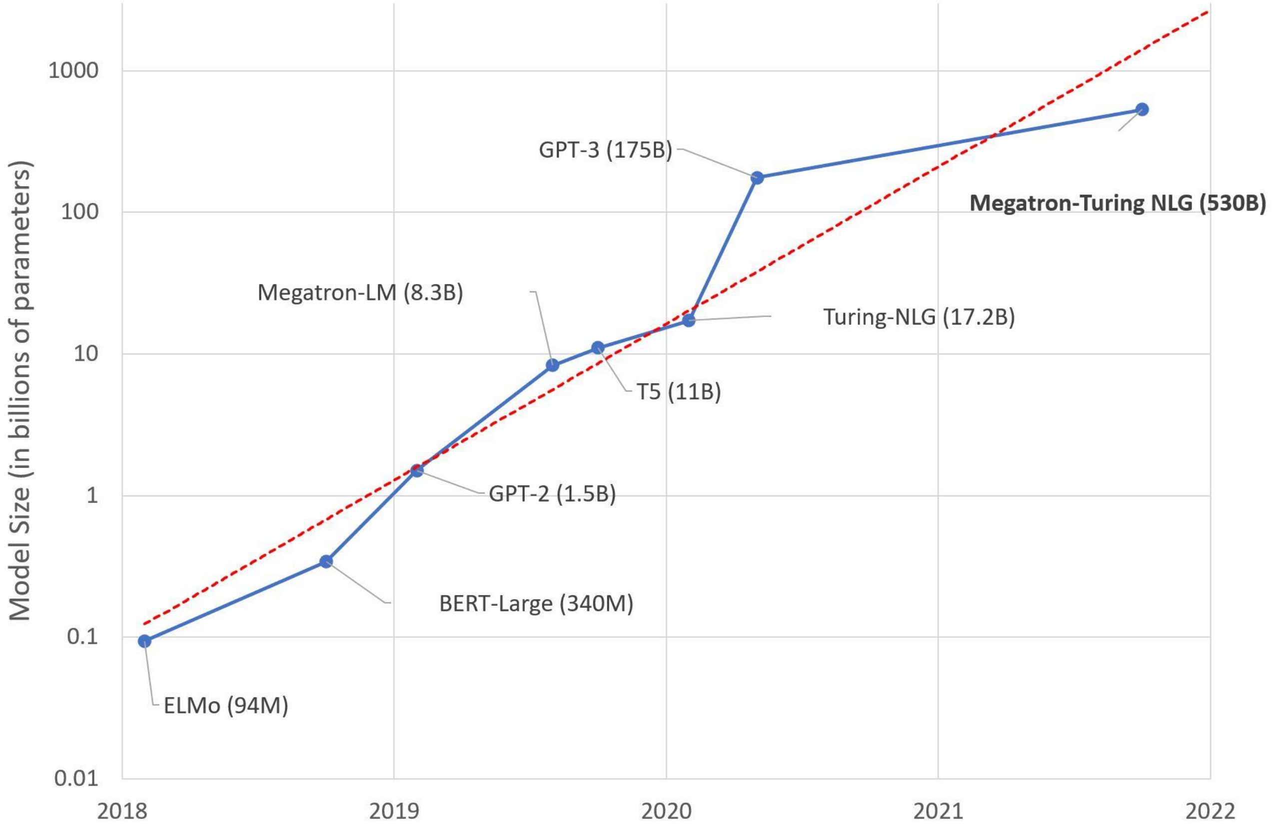
Larger models → better accuracy

Model size is still growing

Not reached the accuracy limit yet

More compute-efficient to train larger models than smaller ones to same accuracy

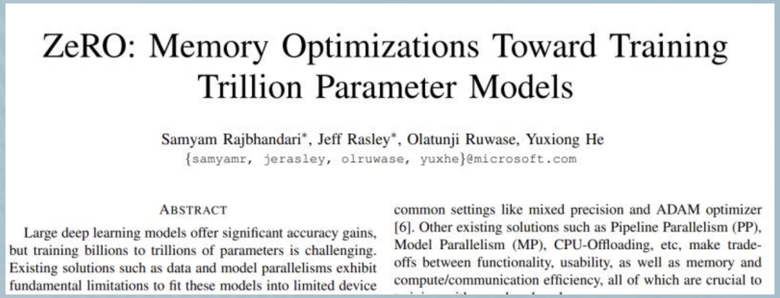
System Challenges/Opportunities of Large language models?



- ➢ Memory
- ➢ Compute
- ➢ Data

ZeRO, ZeRO-Offload, ZeRO-Infinity

Breaking the GPU Memory Wall for DL Training



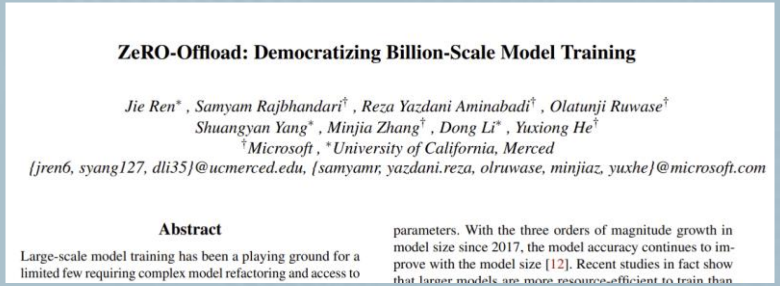
ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Samyam Rajbhandari*, Jeff Rasley*, Olatunji Ruwase, Yuxiong He
{samyamr, jerasley, olruwase, yuxhe}@microsoft.com

ABSTRACT

Large deep learning models offer significant accuracy gains, but training billions to trillions of parameters is challenging. Existing solutions such as data and model parallelisms exhibit fundamental limitations to fit these models into limited device

common settings like mixed precision and ADAM optimizer [6]. Other existing solutions such as Pipeline Parallelism (PP), Model Parallelism (MP), CPU-Offloading, etc, make trade-offs between functionality, usability, as well as memory and compute/communication efficiency, all of which are crucial to



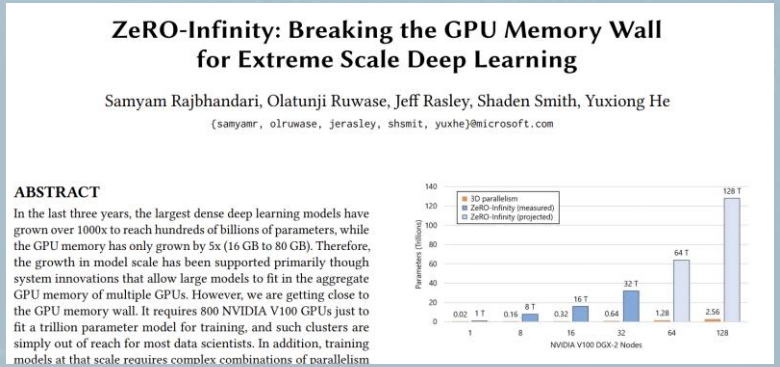
ZeRO-Offload: Democratizing Billion-Scale Model Training

Jie Ren*, Samyam Rajbhandari†, Reza Yazdani Aminabadi†, Olatunji Ruwase†
Shuangyan Yang*, Minjia Zhang†, Dong Li*, Yuxiong He†
†Microsoft, *University of California, Merced
{jren6, syang127, dli35}@ucmerced.edu, {samyamr, yazdani.reza, olruwase, minjiaz, yuxhe}@microsoft.com

Abstract

Large-scale model training has been a playing ground for a limited few requiring complex model refactoring and access to

parameters. With the three orders of magnitude growth in model size since 2017, the model accuracy continues to improve with the model size [12]. Recent studies in fact show



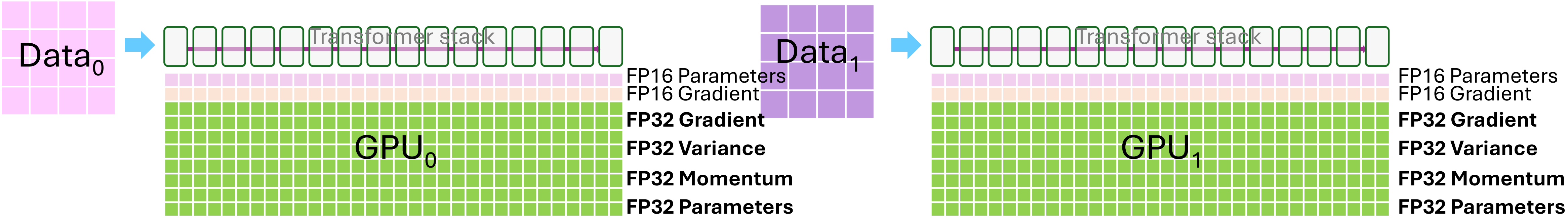
ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, Yuxiong He
{samyamr, olruwase, jerasley, shsmit, yuxhe}@microsoft.com

ABSTRACT

In the last three years, the largest dense deep learning models have grown over 1000x to reach hundreds of billions of parameters, while the GPU memory has only grown by 5x (16 GB to 80 GB). Therefore, the growth in model scale has been supported primarily though system innovations that allow large models to fit in the aggregate GPU memory of multiple GPUs. However, we are getting close to the GPU memory wall. It requires 800 NVIDIA V100 GPUs just to fit a trillion parameter model for training, and such clusters are simply out of reach for most data scientists. In addition, training

Understanding Memory Consumption



- FP(BF)16 parameter : **2M bytes**
- FP(BF)16 Gradients : **2M bytes**
- FP32 Optimizer States : **16M bytes**
  - Gradients, Variance, Momentum, Parameters

M = number of parameters in the model

Example 1B parameter model -> 20GB/GPU

Memory consumption doesn't include:

- Input batch + activations

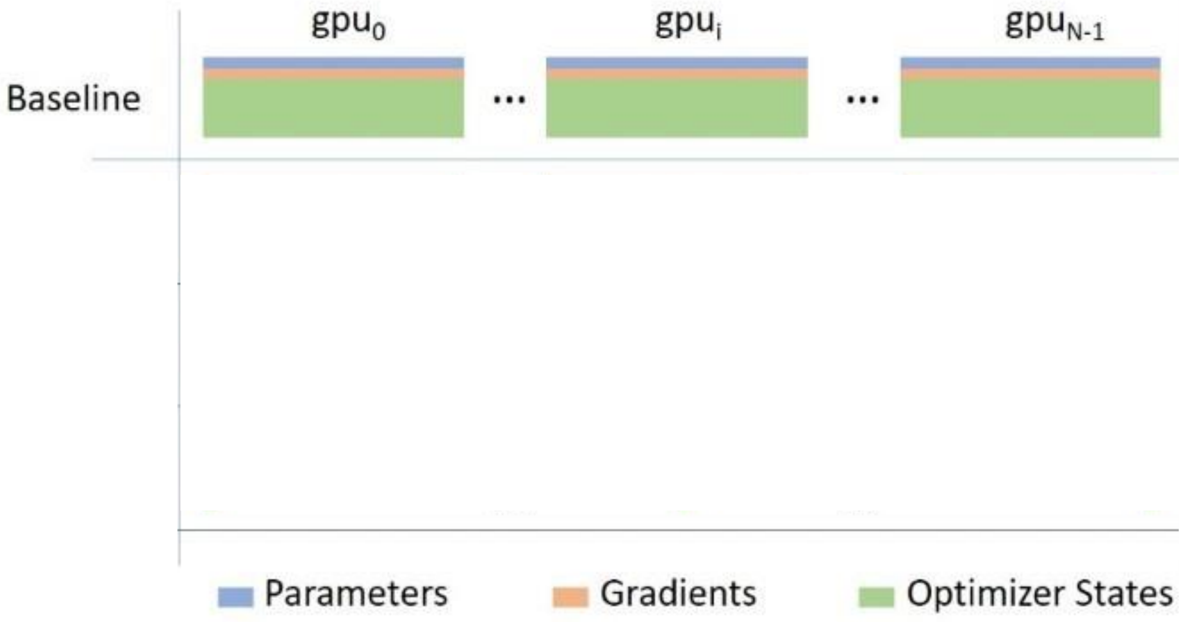
*Mixed Precision Training (ICLR '18) with Adam Optimizer

ZeRO: Overcoming GPU memory wall

- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories

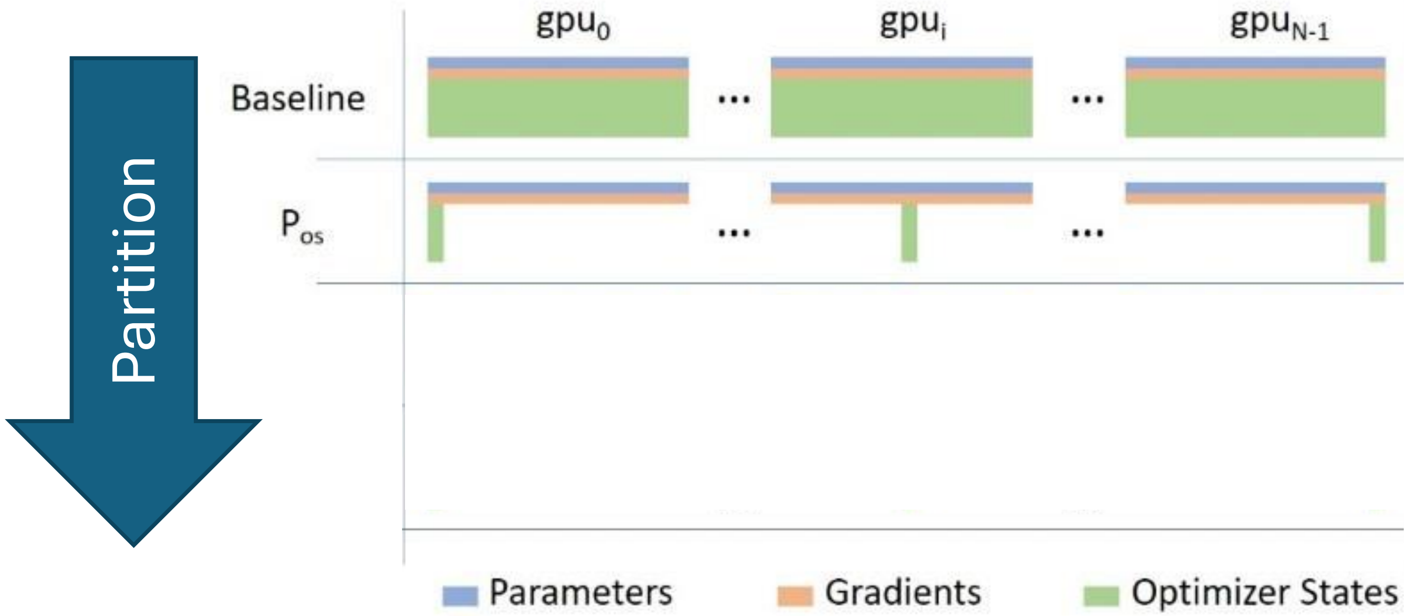
ZeRO: Overcoming GPU memory wall

- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



ZeRO: Overcoming GPU memory wall

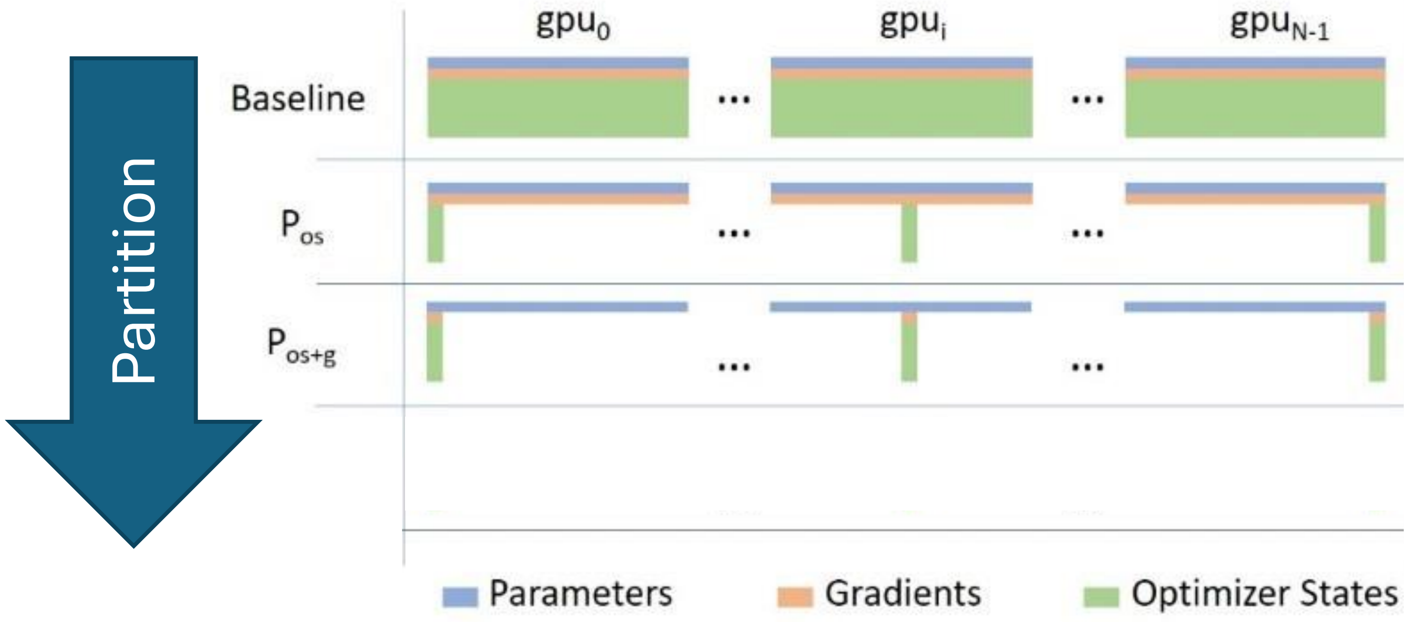
- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



Stage 1 (P_{os})

ZeRO: Overcoming GPU memory wall

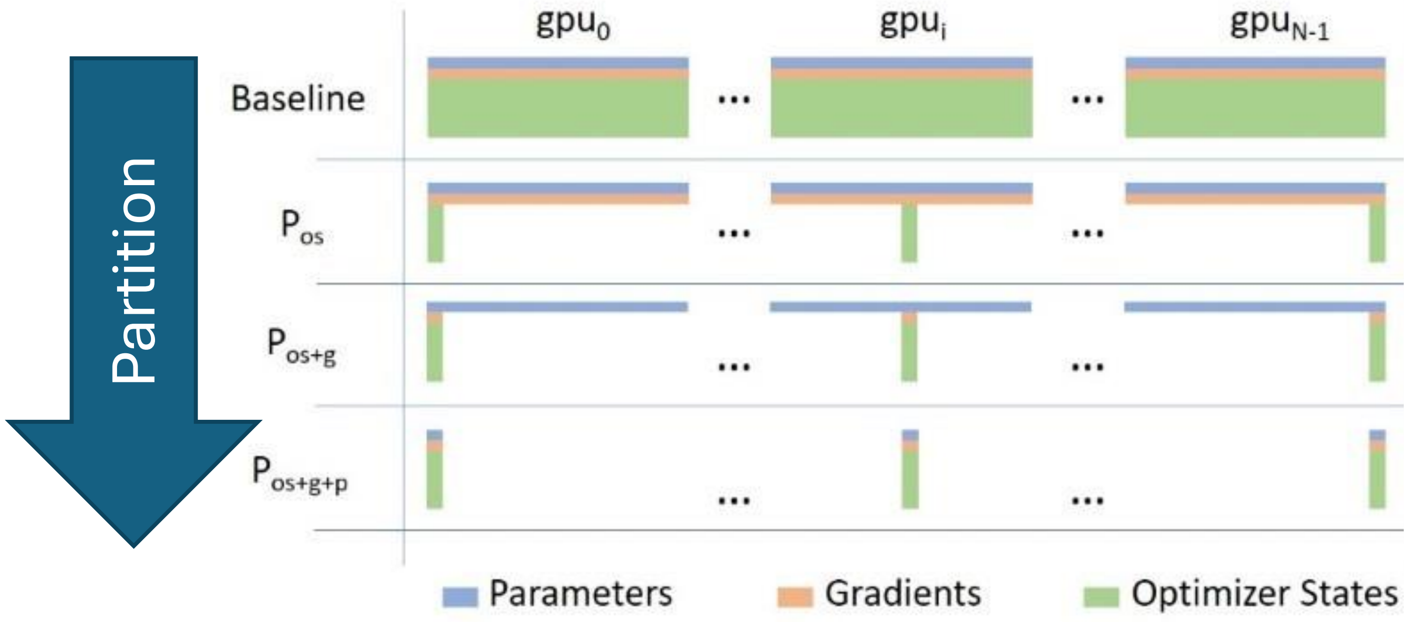
- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



Stage 2 (P_{os+g})

ZeRO: Overcoming GPU memory wall

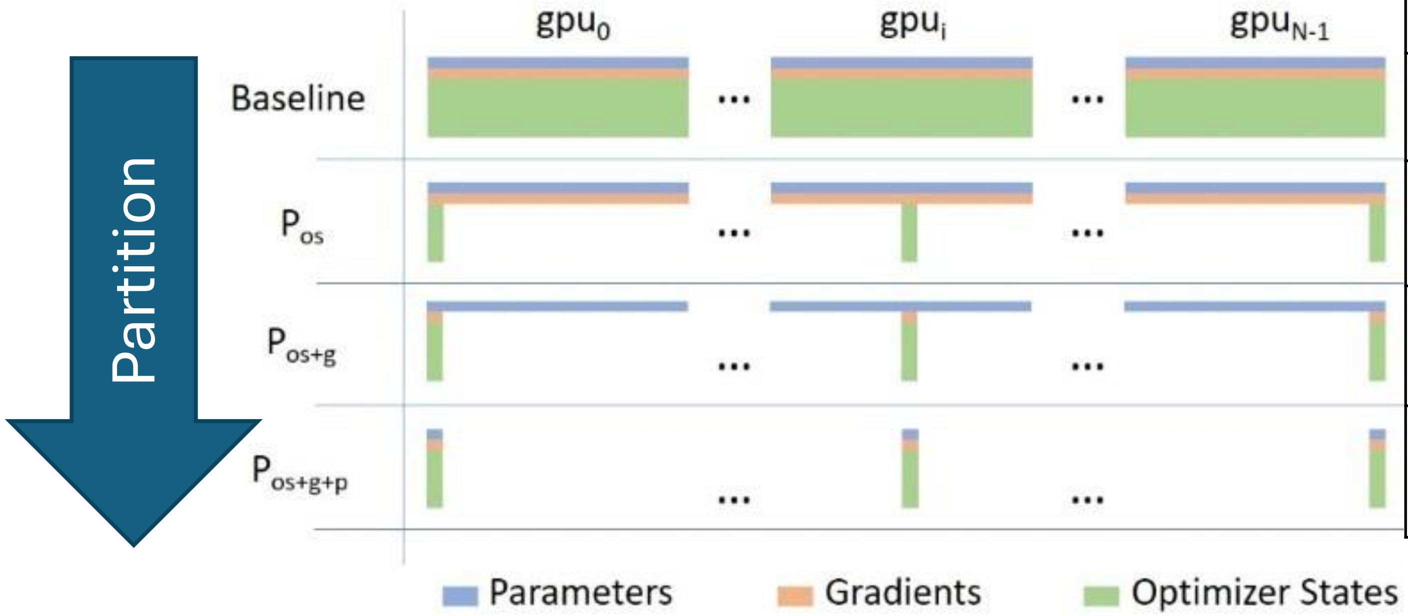
- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



Stage 3 (P_{os+g+p})

ZeRO: Overcoming GPU memory wall

- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



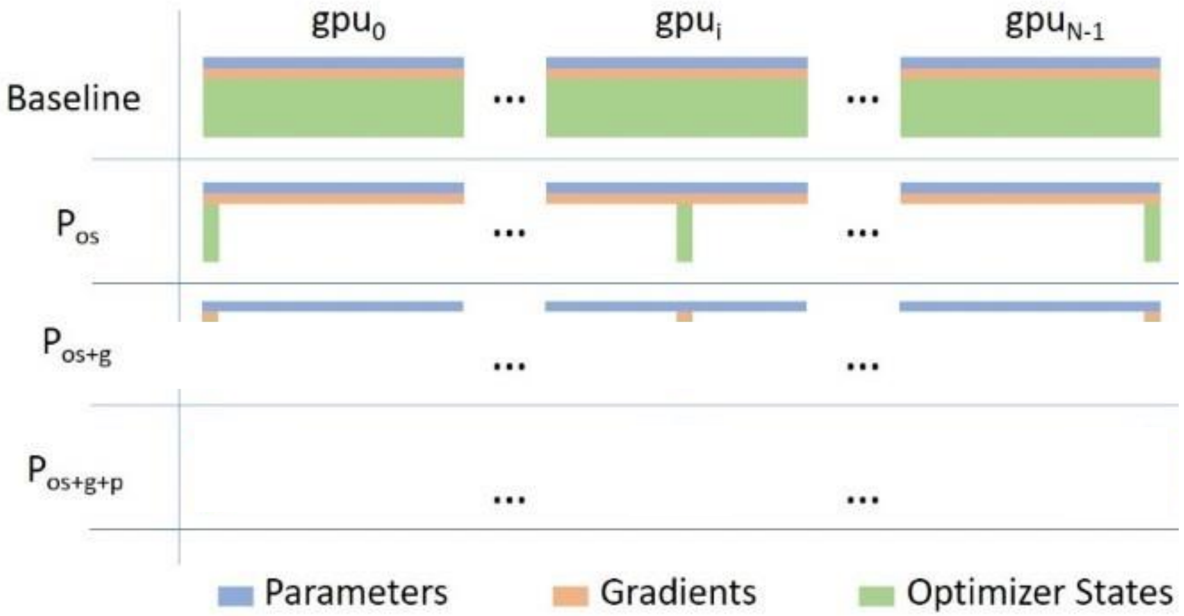
Bytes/param/GPU
2 + 2 + 16 = 20
2 + 2 + (16/N) < 5
2 + ((2+16)/N) < 3
(2 + 2 + 16)/N < 1

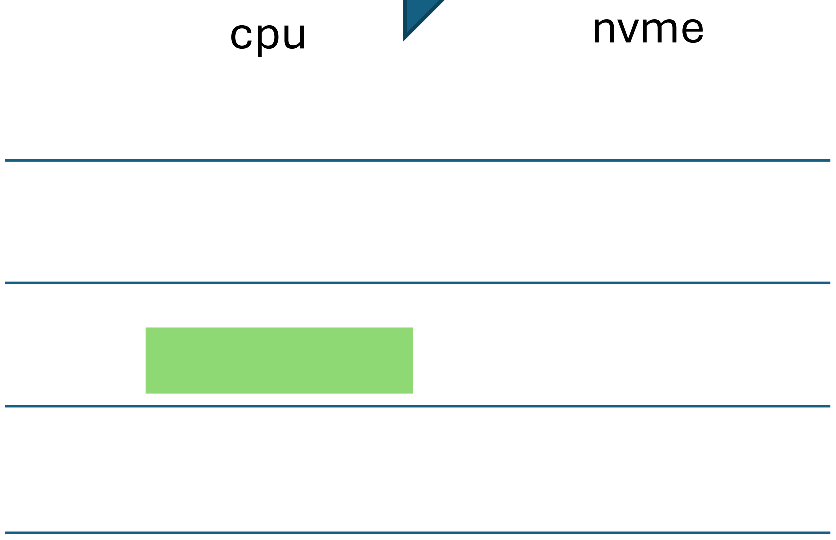
ZeRO: Overcoming GPU memory wall

- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories

Offload

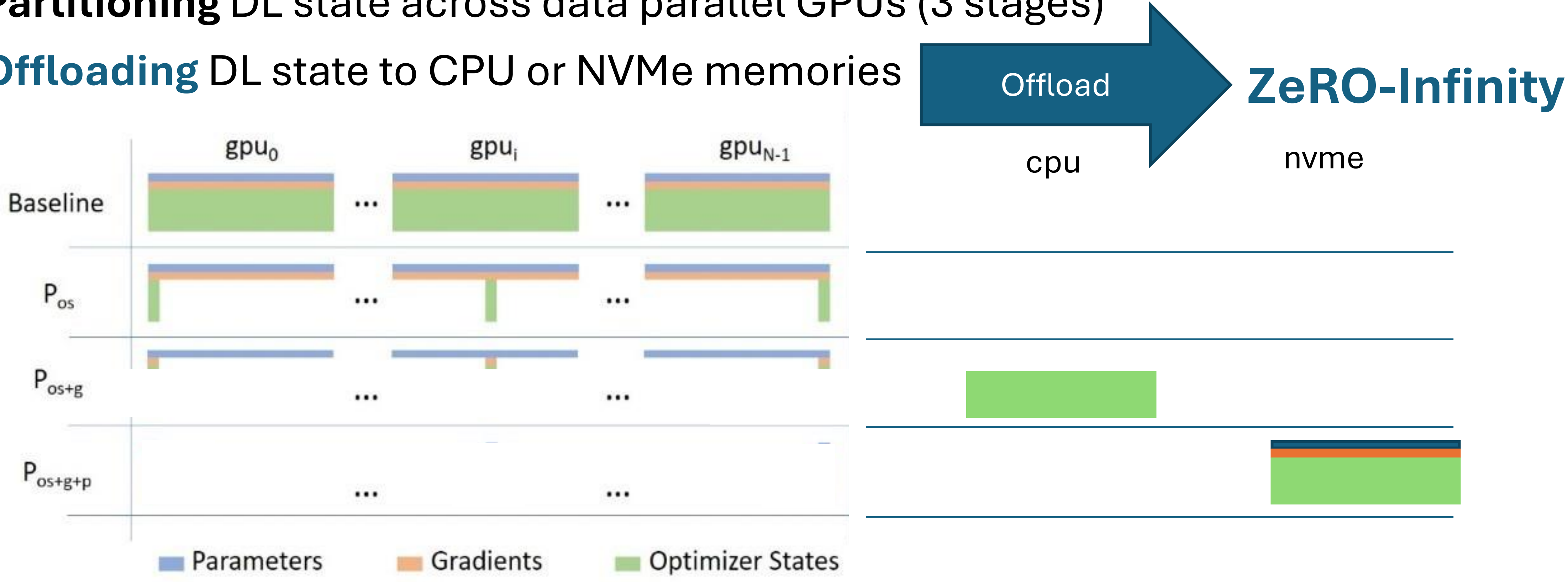
ZeRO-Offload





ZeRO: Overcoming GPU memory wall

- Family of composable optimizations to reduce GPU memory costs of DL state (params, grads, optimizer)
- **Partitioning** DL state across data parallel GPUs (3 stages)
- **Offloading** DL state to CPU or NVMe memories



ZeRO-Infinity

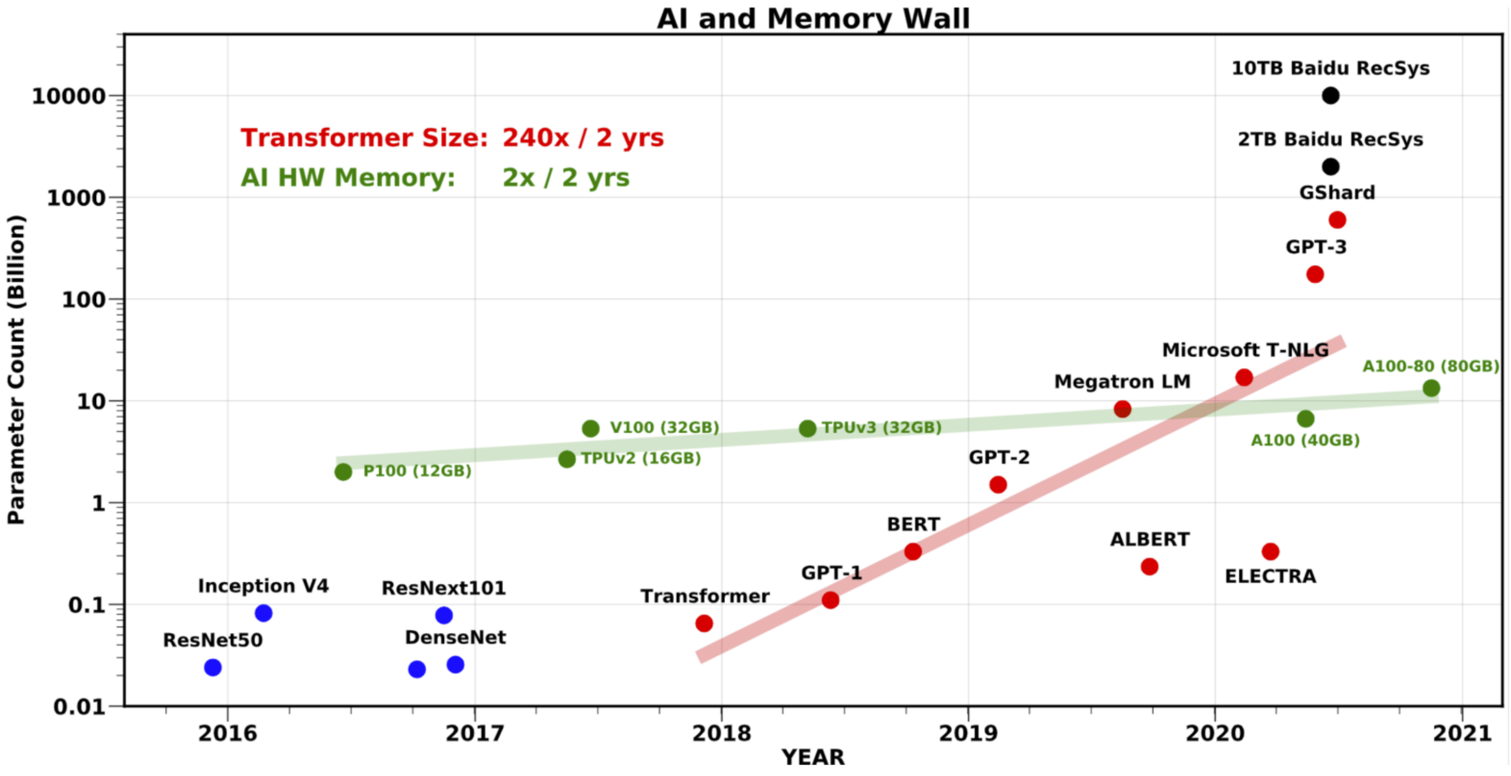
Breaking the GPU Memory Wall for DL Training

Large model training landscape

- GPU Memory Wall
  - 1T (10T) params: 800 (8K) V100 GPUs
  - How do we support the growth in model size?

- Accessibility to large model training
  - 256 GPUs to fine-tune GPT-3
  - Limited access to such resources

- Model code refactoring
  - Re-writing the model using 3D parallelism (tensor-slicing + pipeline parallelism)
  - Painful and error prone

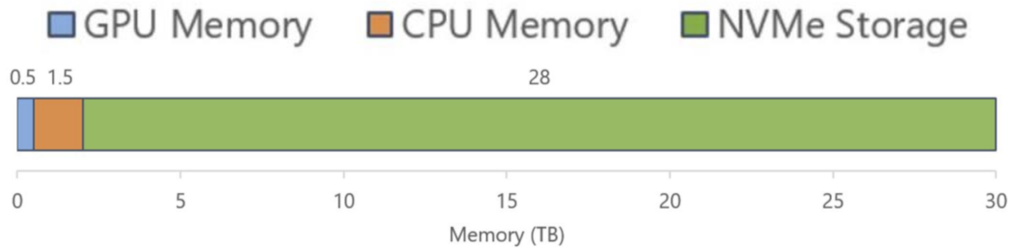


*AI and Memory Wall. (This blogpost has been written in... | by Amir Gholami | riselab | Medium

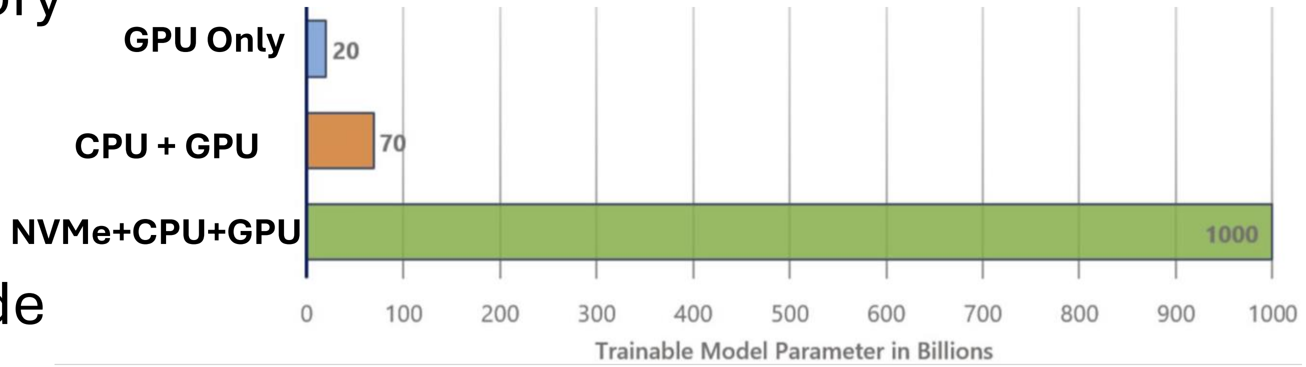
Beyond the GPU Memory

- Modern clusters have heterogeneous memory systems.
- GPU memory is a small fraction
- Leverages GPU/CPU/NVMe memory
  - 32T params on 32 nodes
  - 1T params on a single node
- Fine-tune GPT-3 size on single node

Memory available on a Single DGX-2 Node

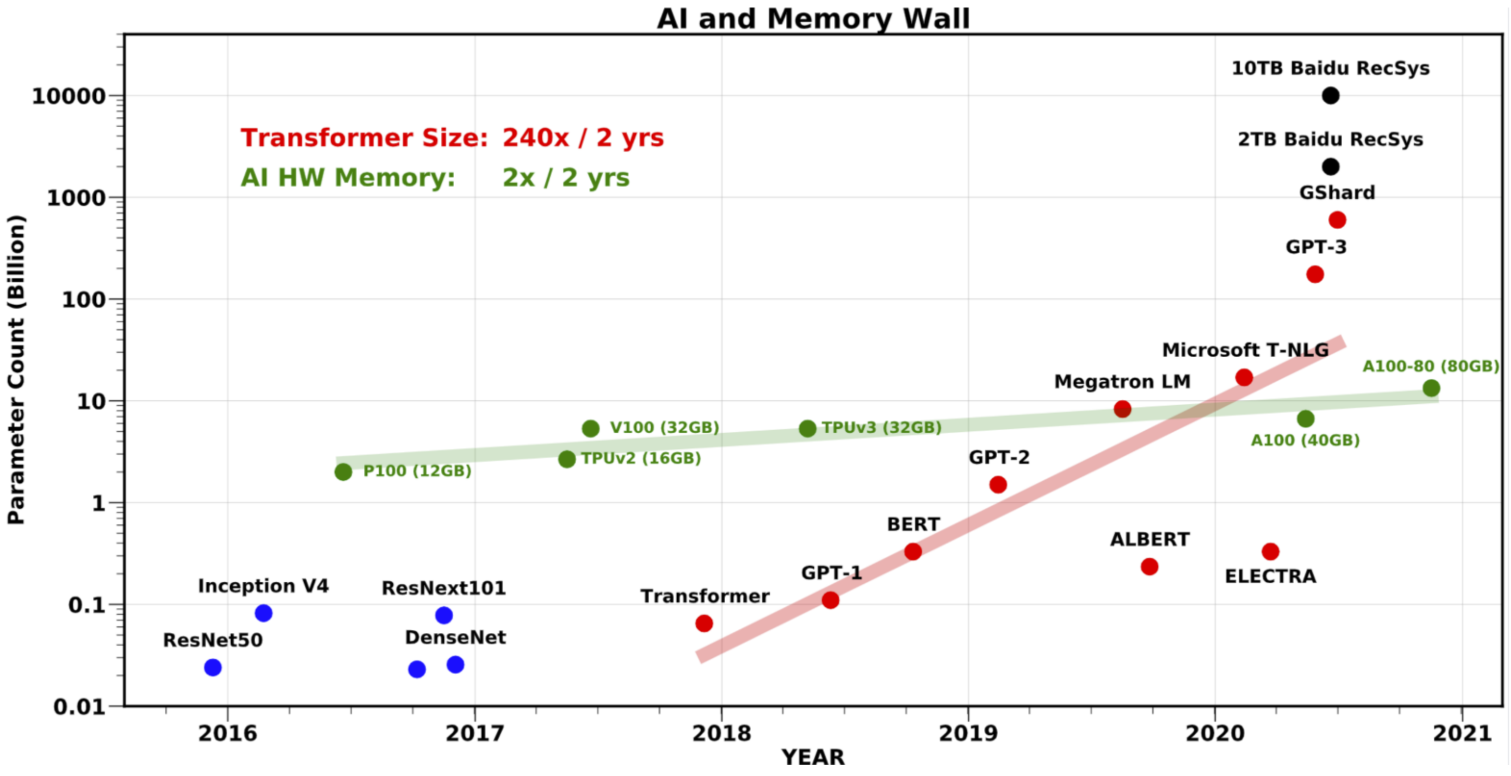


Model Size on a Single DGX-2 Node



Recap: Large model training landscape today

- **GPU Memory Wall**
  - 1T (10T) params: 800 (8K) V100 GPUs
  - How do we support the growth in model size?
- **Accessibility to large model training**
  - 256 GPUs to fine-tune GPT-3
  - Limited access to such resources
- **Model code refactoring**
  - Re-writing the model using 3D parallelism (tensor-slicing + pipeline parallelism)
  - Painful and error prone



*AI and Memory Wall. (This blogpost has been written in... | by Amir Gholami | riselab | Medium

Redefining the landscape with ZeRO-Infinity

- Beyond GPU Memory
  - 50x larger models
  - 32T params on 512 GPUs (instead of 25K)

- Broader access to large model traini
  - GPT-3 sized fine-tuning on a single node/GPU (instead of 16 nodes)

- Excellent Throughput and Scalability
  - Comparable to 3D-parallelism

- Ease of Use
  - No model refactoring necessary

Paper: [ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning (arxiv.org)]

DeepSpeed Mixture of Experts (MoE)

Improving Compute Efficiency for DL scaling

DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, Yuxiong He *Proceedings of the 39th International Conference on Machine Learning*, PMLR 162:18332-18346, 2022.

Abstract

As the training of giant dense models hits the boundary on the availability and capability of the hardware resources today, Mixture-of-Experts (MoE) models have become one of the most promising model architectures due to their significant training cost reduction compared to quality-equivalent dense models. Their training cost saving is demonstrated from encoder-decoder models (prior works) to a 5x saving for auto-aggressive language models (this work). However, due to the much larger model size and unique architecture, how to provide fast MoE model inference remains challenging and unsolved, limiting their practical usage. To tackle this, we present DeepSpeed-MoE, an end-to-end MoE training

A Hybrid Tensor-Expert-Data Parallelism Approach to Optimize Mixture-of-Experts Training

Siddharth Singh
ssingh37@umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

Olatunji Ruwase
olruwase@microsoft.com
Microsoft, Inc.
Redmond, Washington, USA

Ammar Ahmad Awan
ammar.awan@microsoft.com
Microsoft, Inc.
Redmond, Washington, USA

Samyam Rajbhandari
samyamr@microsoft.com
Microsoft, Inc.
Redmond, Washington, USA

Yuxiong He
yuxhe@microsoft.com
Microsoft, Inc.
Redmond, Washington, USA

Abhinav Bhatele
bhatele@cs.umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

ABSTRACT

Mixture-of-Experts (MoE) is a neural network architecture that adds sparsely activated expert blocks to a base model, increasing the number of parameters without impacting computational costs.

1 INTRODUCTION

Contemporary state-of-the-art AI algorithms have come to rely on neural networks such as GPT-3 [4] and MT-NLG [34] with hundreds of billion of parameters. However, training or running inference

Mixture of Experts (MoE): Overview

- MoE models have been around for a while..
- Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer
  - Harder to scale, instability during training, and inefficient training
- GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding
  - 600B models beating 96-layer dense models, 10x training speedup, generic sharding framework (Tensorflow XLA), full precision training
  - Less stability with larger models
- Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity
  - More efficient training
    - Top-1 gating instead of top-2/top-k, Better initialization conditions, Mixed precision training: FP32 gating (instead of FP16), Stable training with larger models
  - SOTA results on language understanding task

MoE: Gshard and Switch Transformer

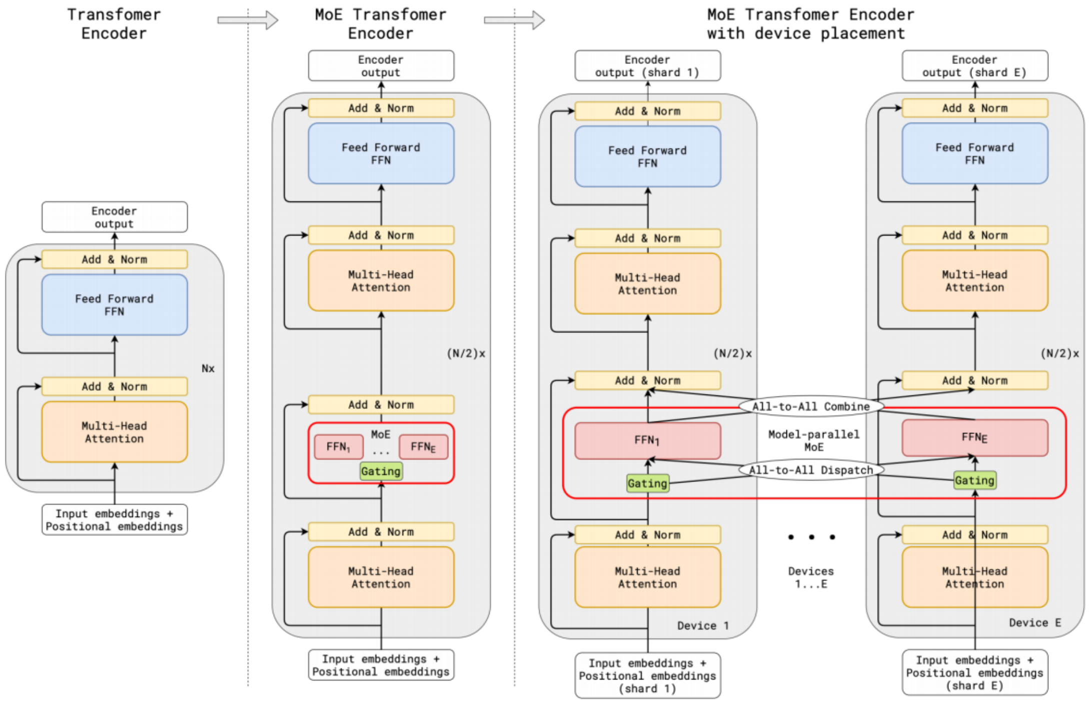


Figure 3: Illustration of scaling of Transformer Encoder with MoE Layers. The MoE layer replaces the every other Transformer feed-forward layer. Decoder modification is similar. (a) The encoder of a standard Transformer model is a stack of self-attention and feed forward layers interleaved with residual connections and layer normalization. (b) By replacing every other feed forward layer with a MoE layer, we get the model structure of the MoE Transformer Encoder. (c) When scaling to multiple devices, the MoE layer is sharded across devices, while all other layers are replicated.

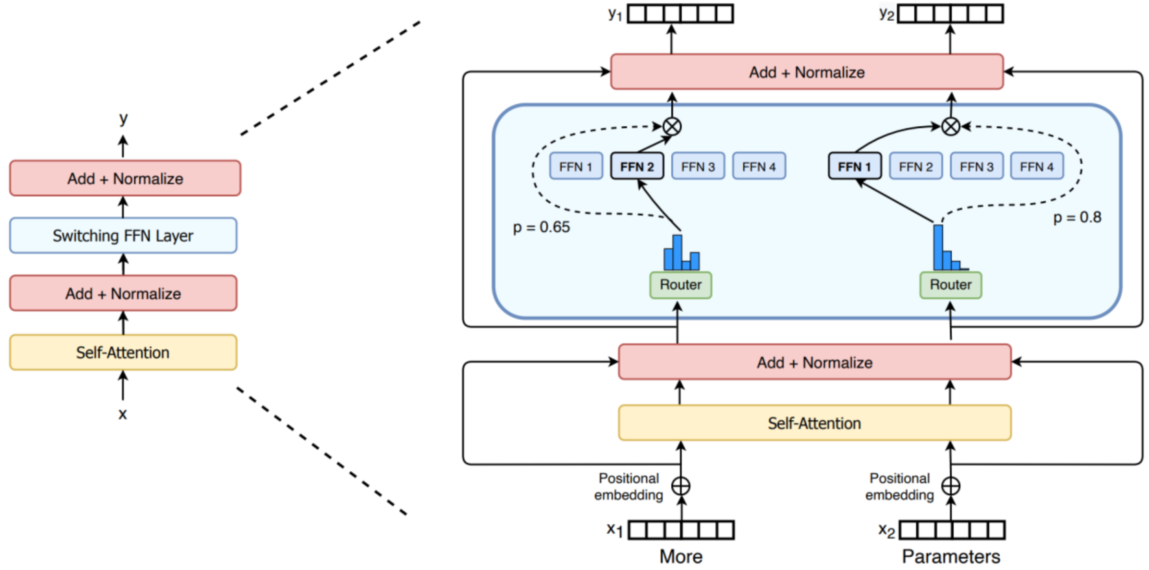


Figure 2: **Illustration of a Switch Transformer encoder block**. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

MoE models are sparse and need less compute

Dense Models:

- All parameters are used in forward and backward paths
- Increasing model capacity needs more computation
- Optimized for dense computation
- **Larger model size → Higher compute requirements (FLOPs)**

Sparse MoE models

- Sparse utilization of subset of parameters based on input
- Same computation is needed regardless of the model size
- Not-optimized for dense computation
- **Larger model size → Similar/Same Compute requirements**

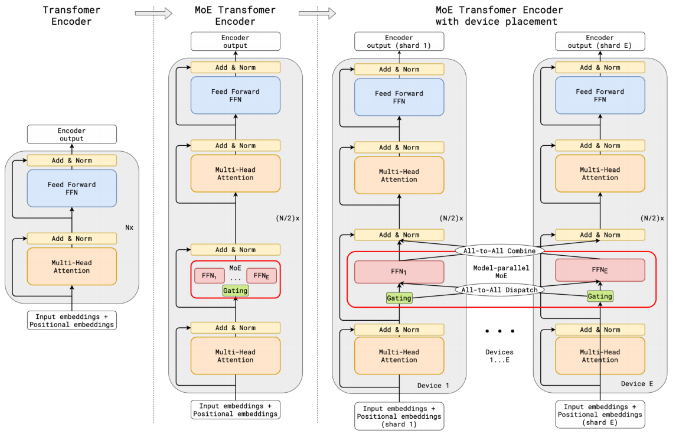


Figure 3: Illustration of scaling of Transformer Encoder with MoE Layers. The MoE layer replaces the every other Transformer feed-forward layer. Decoder modification is similar. (a) The encoder of a standard Transformer model is a stack of self-attention and feed forward layers interleaved with residual connections and layer normalization. (b) By replacing every other feed forward layer with a MoE layer, we get the model structure of the MoE Transformer Encoder. (c) When scaling to multiple devices, the MoE layer is sharded across devices, while all other layers are replicated.

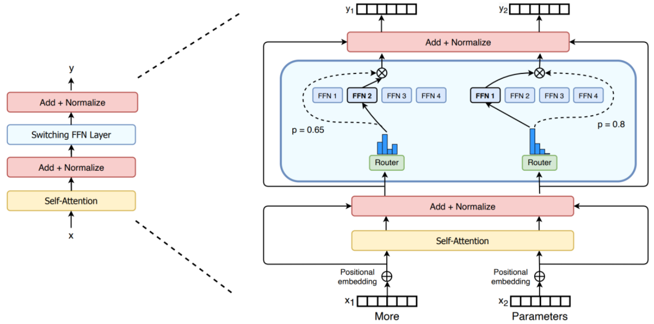
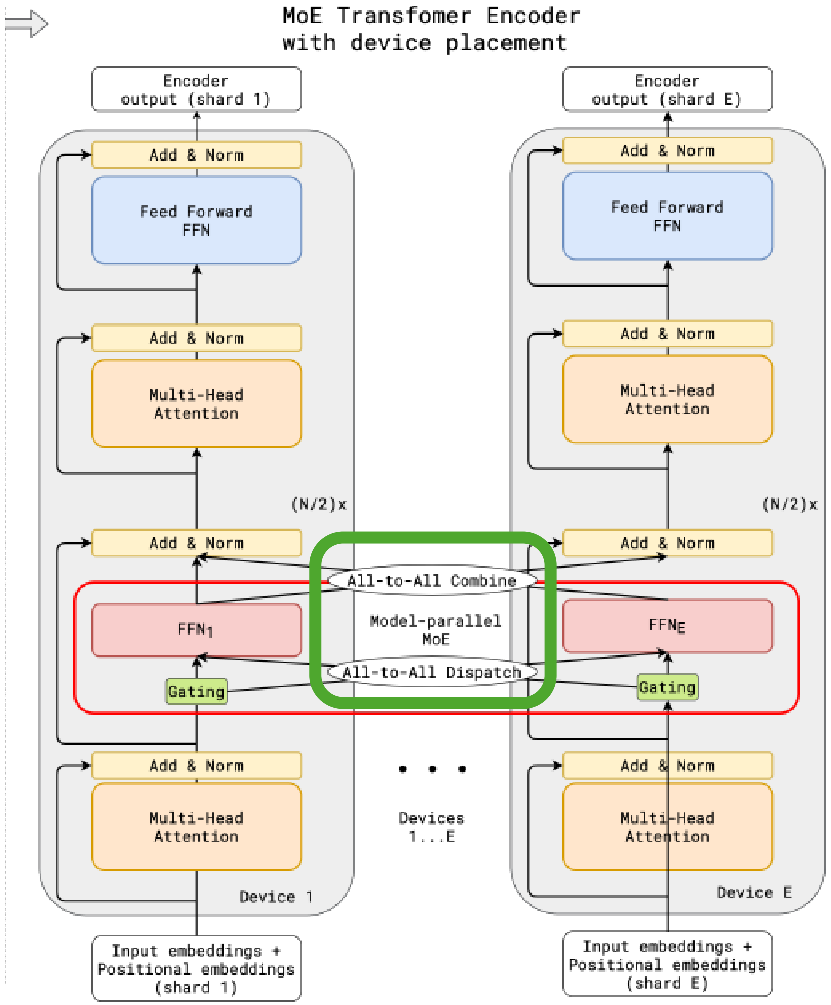


Figure 2: **Illustration of a Switch Transformer encoder block**. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

What is Expert Parallelism?

- Expert Parallelism --> Data and Model parallelism within the model
  - Non-MoE parameters – replicated
    - Like standard data parallelism (DP)
    - ZeRO DP in DeepSpeed can shard these too!
  - MoE parameters – partitioned (sharded)
    - Like model parallelism (MP)
  - **Two All-to-All(s) in Forward and Backward**



Next AI Scale on current hardware

- Can we achieve next generation model quality on current generation of hardware?

- From a training perspective MoE provides a promising path
  - Scale at sub-linear cost
  - Z-Code multi-lingual multi-task model

- MoE is *promising* but is it *practical*?
  - **Limited Scope:** Does it work for NLG or NLR or other models?
  - **Massive Memory Requirements:** 8-10x in size compared to quality equivalent dense
  - **Limited expert scaling:** Diminishing returns at 64-128 experts?

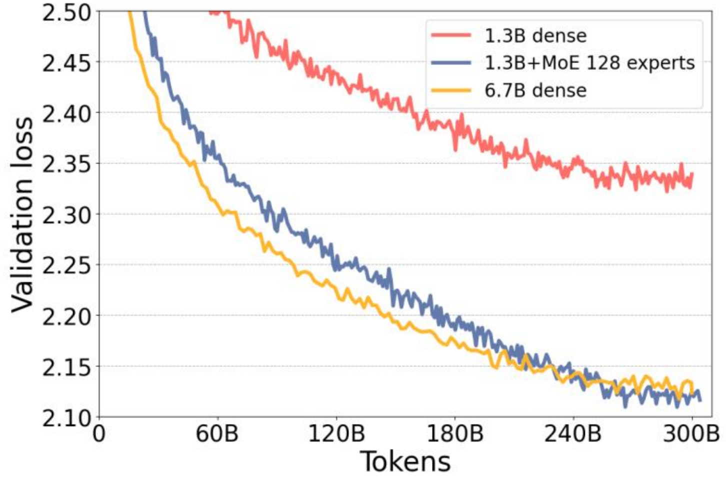
DeepSpeed MoE: Multidimensional Parallelism

Short Name	Flexible Parallelism Combinations	Benefit
E	Expert	Scales the model size by increasing the number of experts
E+D	Expert + Data	Accelerates training throughput by scaling to multiple data parallel groups
E+Z	Expert + ZeRO	Partitions the nonexpert parameters to support larger base models
E+D+M	Expert + Data + Model	Supports massive hidden sizes and even larger base models than E+Z
E+D+Z	Expert + Data + ZeRO	
E+Z-Off+M	Expert + ZeRO-Offload + Model	Leverages both GPU and CPU memory for large MoE models on limited GPU resources

Can scale both: 1) Number of experts and
2) Base model sizes

Cheaper NLG Model Training with MoE

- 1.3B+MoE with 128 experts, compared to 1.3B and 6.7B dense (GPT-3 like)
- **5x** lower training cost to same accuracy using MoE
- **8x** more parameters to same accuracy using MoE



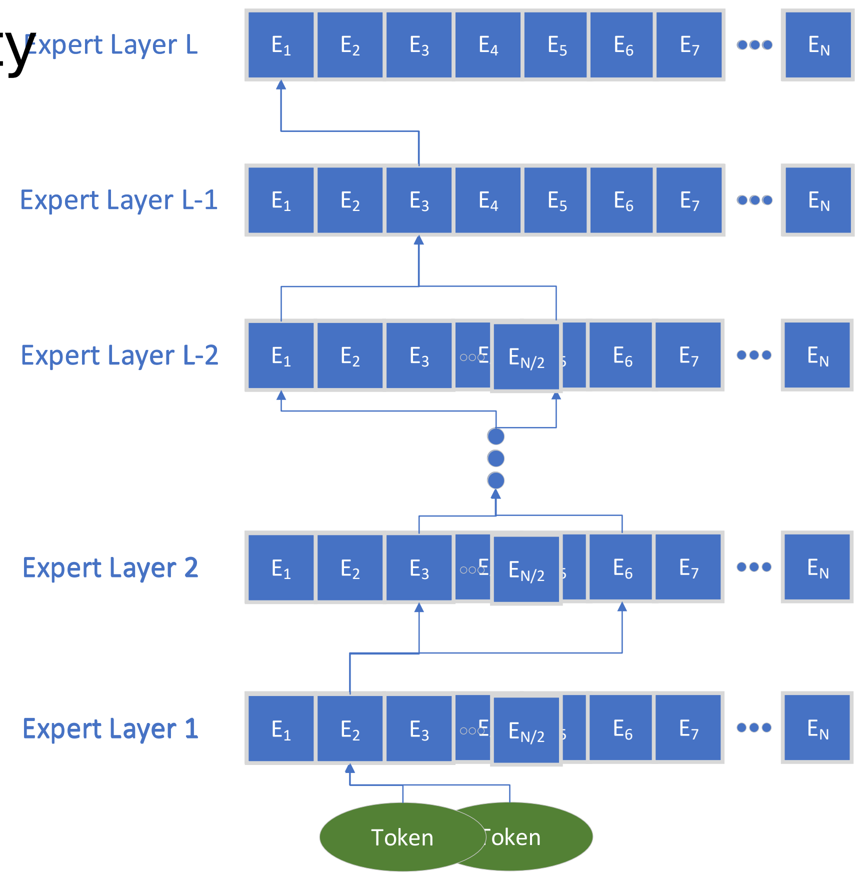
Case	Model size	LAMBADA: completion prediction	PIQA: commonsense reasoning	BoolQ: reading comprehension	RACE-h: reading comprehension	TriviaQA: question answering	WebQs: question answering
Dense NLG:							
(1) 350M	350M	52.03	69.31	53.64	31.77	3.21	1.57
(2) 1.3B	1.3B	63.65	73.39	63.39	35.60	10.05	3.25
(3) 6.7B	$\boxed{6.7B}$	**71.94**	**76.71**	**67.03**	37.42	23.47	5.12
Standard MoE NLG:							
(4) 350M+MoE-128	13B	62.70	74.59	60.46	35.60	16.58	5.17
(5) 1.3B+MoE-128	$\boxed{52B}$	69.84	**76.71**	64.92	**38.09**	**31.29**	**7.19**

	Training samples per sec	Throughput gain/ Cost Reduction
6.7B dense	70	1x
1.3B+MoE-128	372	**5x**

Parameter Efficient MoE via PR-MoE and MoS

- Challenges of MoE: 8x parameters than quality equivalent dense models
  - Training requires large memory footprint
  - Slow inference due to parameter loading

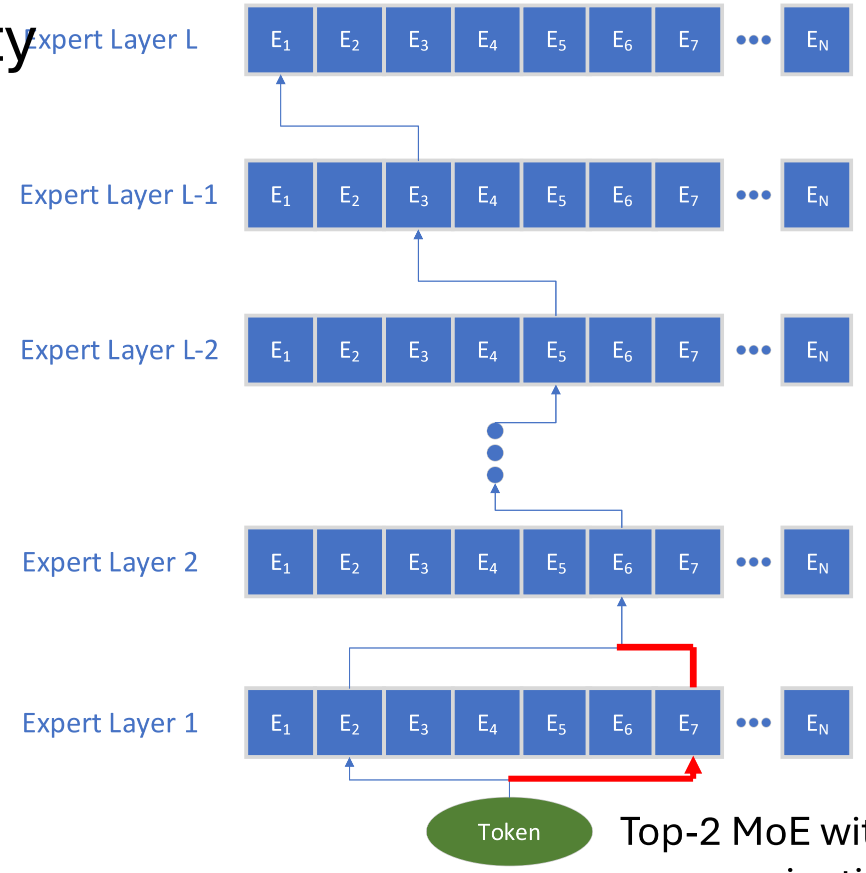
- New opportunities
  - Parameter efficient MoE
    - Homogeneous layer structure → Pyramid MoE



Parameter Efficient MoE via PR-MoE and MoS

- Challenges of MoE: 8x parameters than quality equivalent dense models
  - Training requires large memory footprint
  - Slow inference due to parameter loading

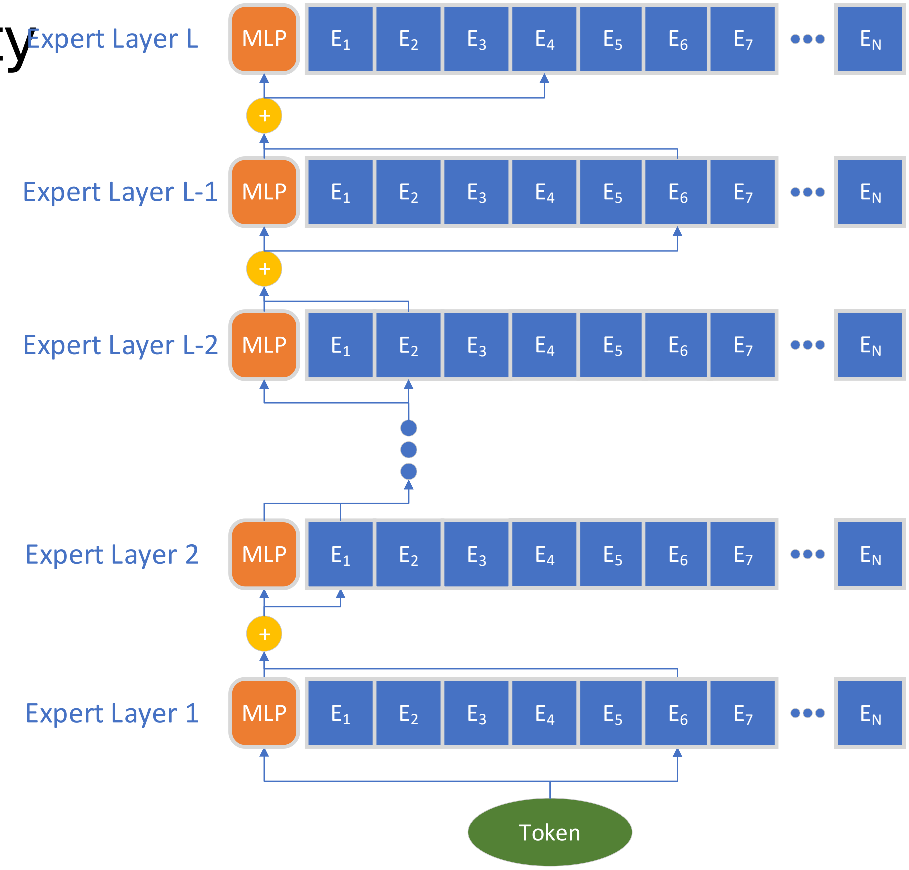
- New opportunities
  - Parameter efficient MoE
    - Homogeneous layer structure → Pyramid MoE
    - Plain structure → Residual MoE



Top-2 MoE with extra communication

Parameter Efficient MoE via PR-MoE and MoS

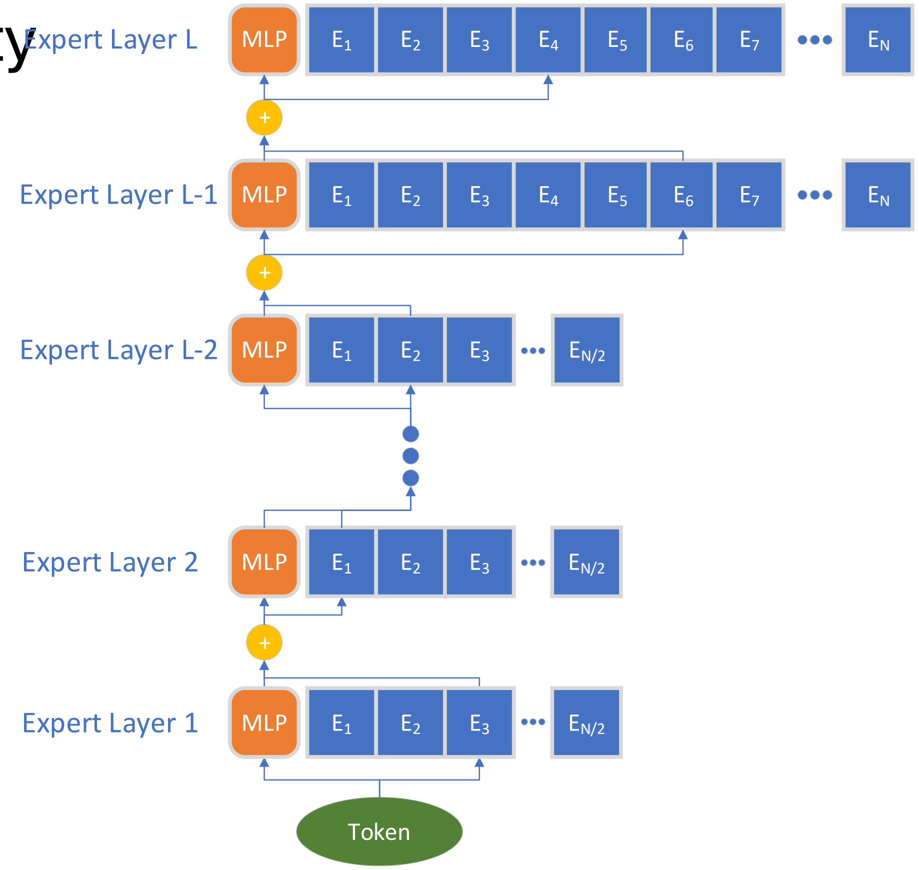
- Challenges of MoE: 8x parameters than quality equivalent dense models
  - Training requires large memory footprint
  - Slow inference due to parameter loading
- New opportunities
  - Parameter efficient MoE
    - Homogeneous layer structure → Pyramid MoE
    - Plain structure → Residual MoE



Parameter Efficient MoE via PR-MoE and MoS

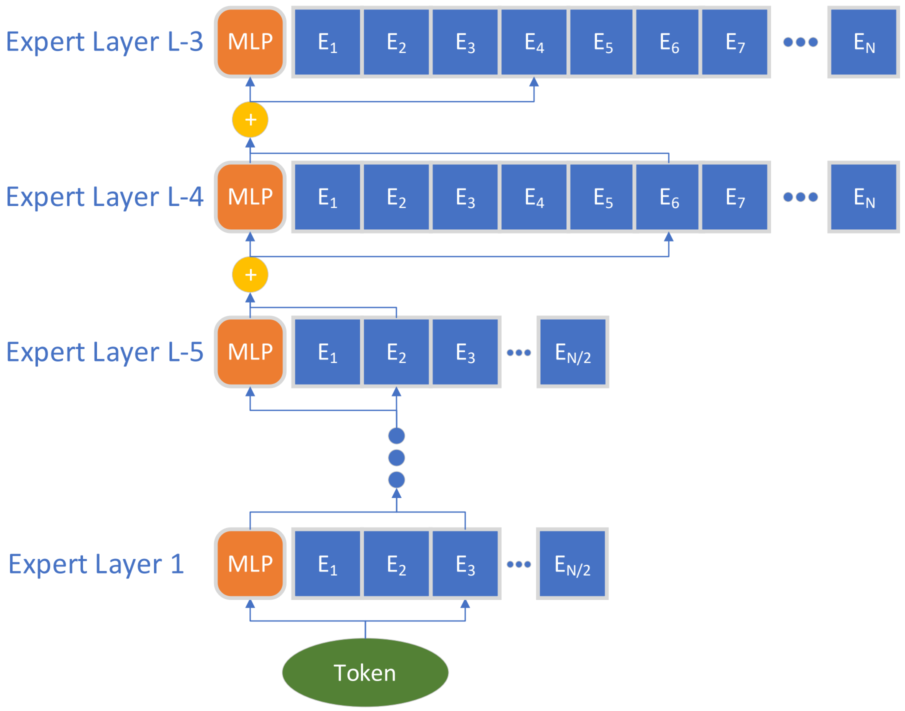
- Challenges of MoE: 8x parameters than quality equivalent dense models
  - Training requires large memory footprint
  - Slow inference due to parameter loading

- New opportunities
  - Parameter efficient MoE
    - Homogeneous layer structure → Pyramid MoE
    - Plain structure → Residual MoE
    - Our design PR-MoE



Parameter Efficient MoE via PR-MoE and MoS

- Challenges of MoE: 8x parameters than quality equivalent dense models
  - Training requires large memory footprint
  - Slow inference due to parameter loading
- New opportunities
  - Parameter efficient MoE
    - Homogeneous layer structure → Pyramid structure
    - Plain structure → Residual structure
    - Our design PR-MoE
  - MoS: MoE-to-MoE Knowledge distillation



Parameter Efficient MoE via PR-MoE and MoS

- PR-MoE: model size reduction from **1.7x** to **3.2x**
- PR-MoE + MoS: model size reduction from **1.9x** to **3.7x**

Case	Model size (Reduction)	LAMBADA	PIQA	BoolQ	RACE-h	TriviaQA	WebQs
MoE NLG with 350M base model:							
(1) MoE	13B (1x)	62.70	**74.59**	**60.46**	35.60	**16.58**	5.17
(2) PR-MoE	4.0B (3.2x)	**63.65**	73.99	59.88	**35.69**	16.30	4.73
(3) PR-MoE + MoS	**3.5B (3.7x)**	63.46	73.34	58.07	34.83	13.69	**5.22**
MoE NLG with 1.3B base model:							
(4) MoE	52B (1x)	69.84	76.71	64.92	**38.09**	**31.29**	7.19
(5) PR-MoE	31B (1.7x)	**70.60**	**77.75**	**67.16**	**38.09**	28.86	7.73
(6) PR-MoE + MoS	**27B (1.9x)**	70.17	77.69	65.66	36.94	29.05	**8.22**

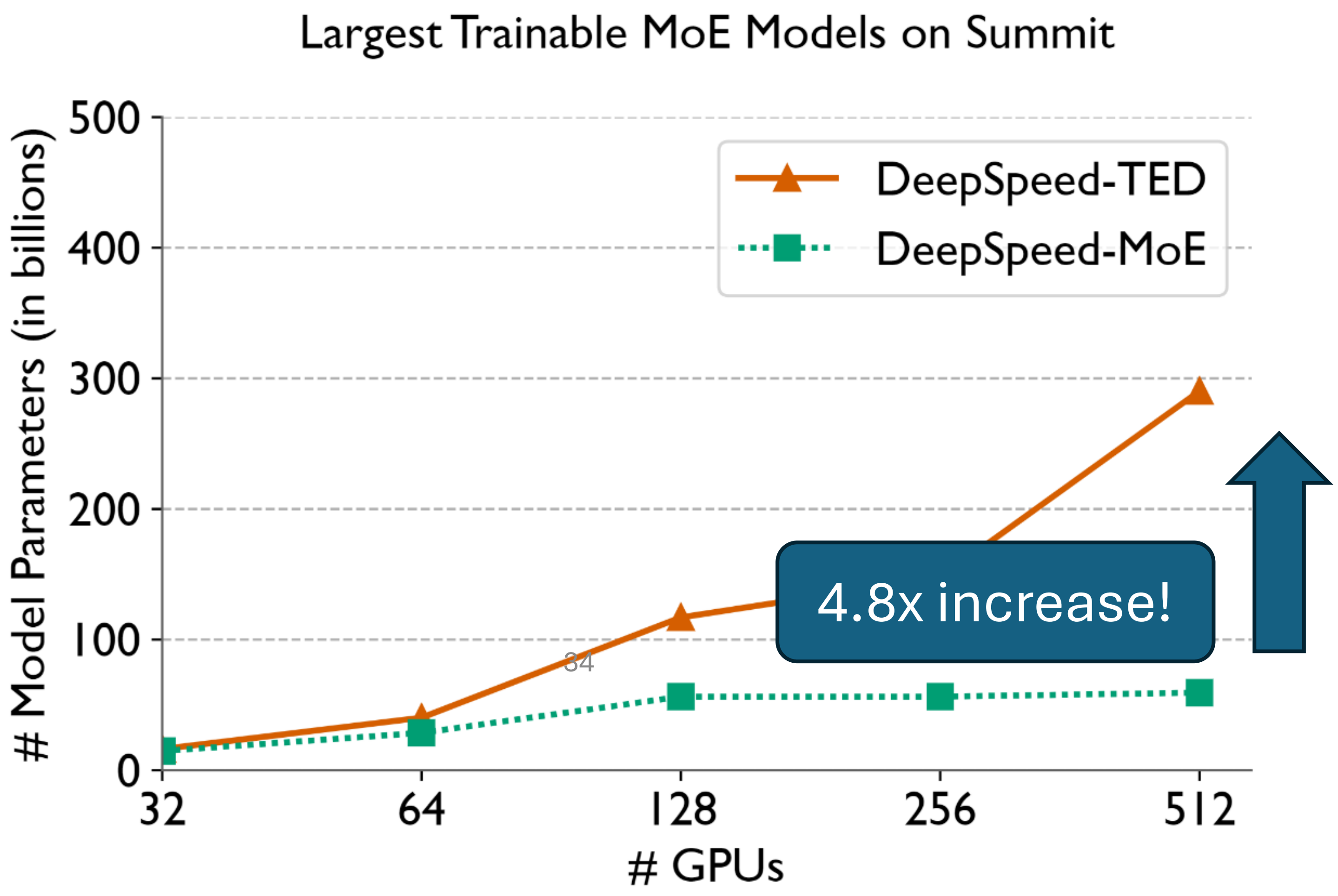
Paper: DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale (ICML’22)

Deepspeed-TED: Scaling MoE base model

- Enable MoEs with large base models
- Minimize communication times to maintain efficiency.

- A three-dimensional hybrid of state-of-the-art parallel training algorithms
  - T – Tensor Parallelism (Megatron-LM [3])
  - E – Expert Parallelism (DeepSpeed-MoE [4])
  - D – Sharded Data Parallelism (ZeRO [5])

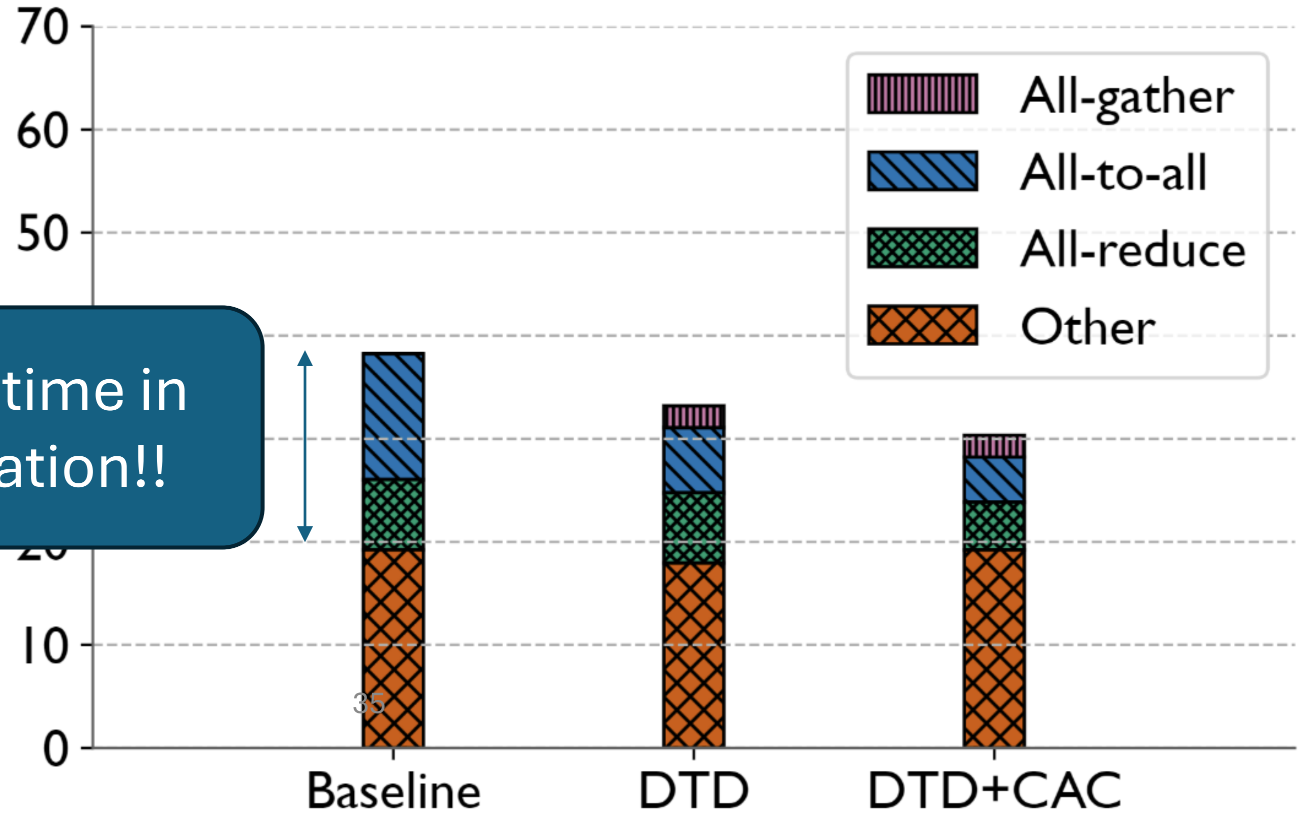
3D parallelism helps up train larger models



- Limit Number of experts to 128
- Limit tensor parallelism to a node.

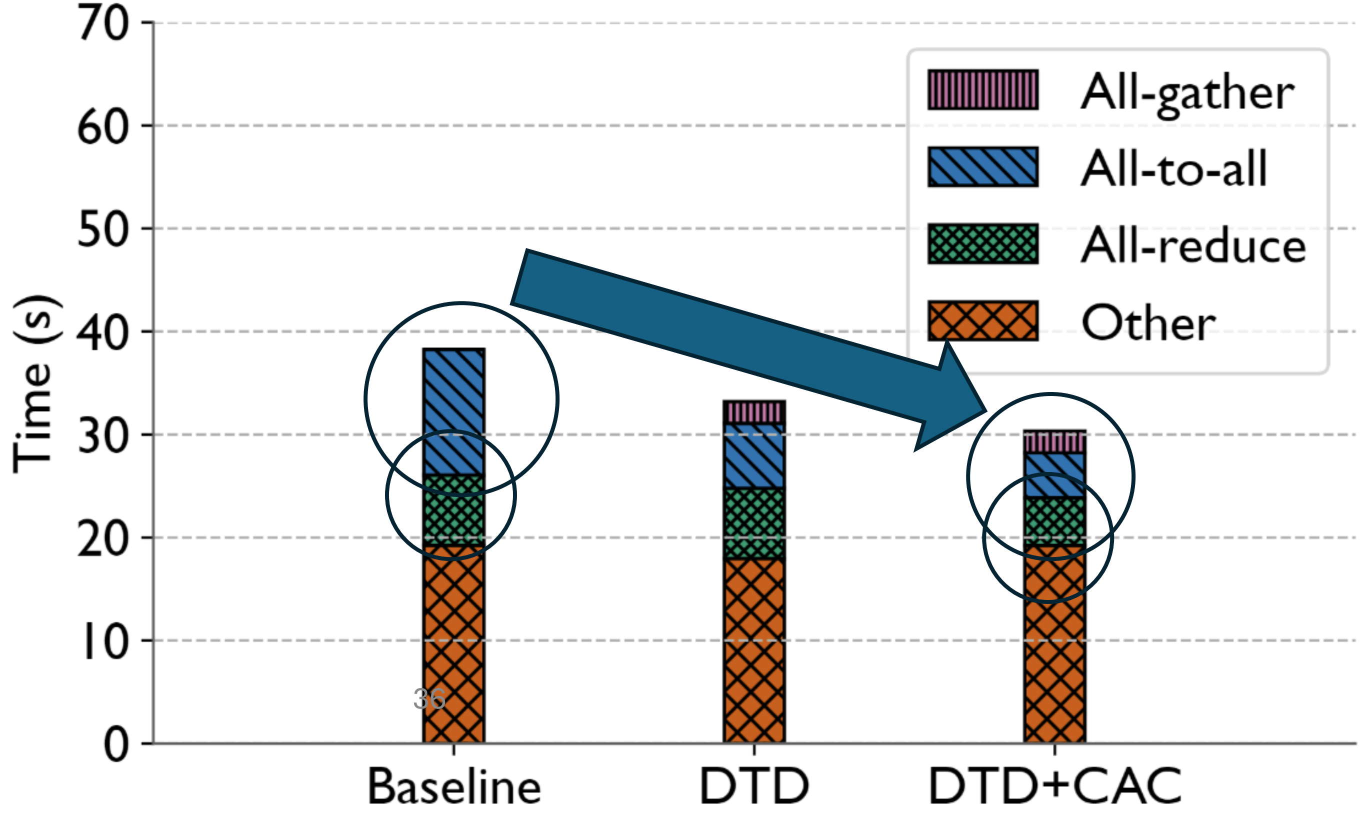
Results

Nearly 50% time in communication!!



Batch Time Profile of a 6.7B base model + 16 experts on 128 GPUs of Summit

Results: Communication optimizations

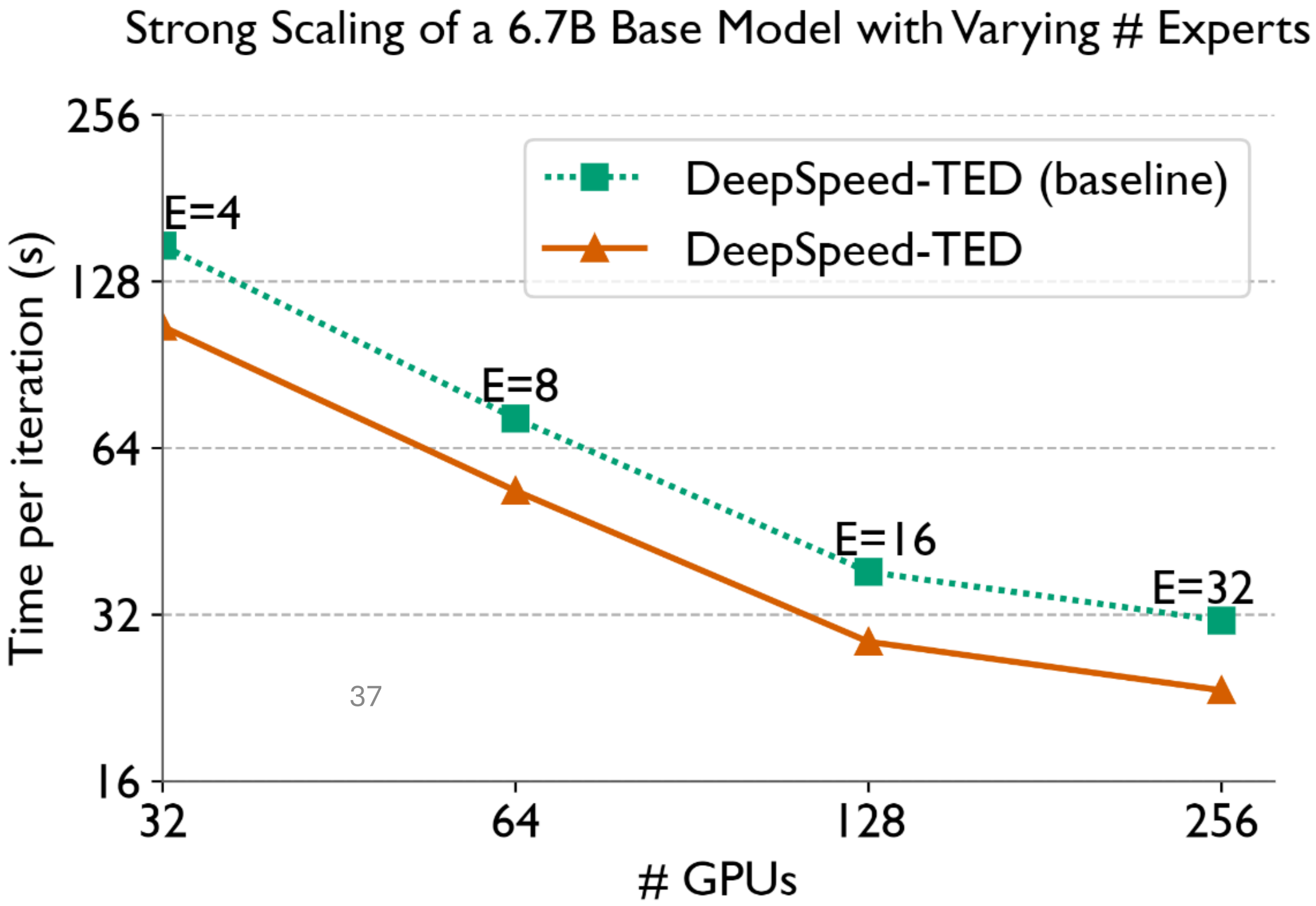


Overall 21% Speedup

Batch Time Profile of a 6.7B base model + 16 experts on 128 GPUs of Summit

Results (Strong Scaling)

Machine - Summit



22-29% speedups!

Data Efficiency

Improving Deep Learning Model Quality and Training Efficiency via Efficient Data Sampling and Routing

The Stability-Efficiency Dilemma: Investigating Sequence Length Warmup for Training GPT Models

Conglong Li Microsoft conglong.li@microsoft.com
Minjia Zhang Microsoft minjiaz@microsoft.com
Yuxiong He Microsoft yuxhe@microsoft.com

Abstract

Recent works have demonstrated great success in pre-training large-scale autoregressive language models (e.g., GPT-3) on massive GPUs. To reduce the wall-clock training time, a common practice is to increase the batch size and learning rate. However, such practice is often brittle and leads to a so-called stability-efficiency

DeepSpeed Data Efficiency: Improving Deep Learning Model Quality and Training Efficiency via Efficient Data Sampling and Routing

Conglong Li*, Zhewei Yao*, Xiaoxia Wu*, Minjia Zhang, Connor Holmes, Cheng Li, Yuxiong He
Microsoft

Abstract

Recent advances on deep learning models come at the price of formidable training cost. The increasing model size is one of the root causes, but another less-emphasized fact is that data scale is actually increasing at a similar speed as model scale, and the training cost is proportional to both of them. Compared to the rapidly evolving model architecture, how to efficiently use the training data (especially for the expensive foundation model pretraining) is both less explored and

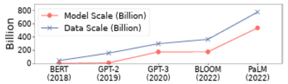


Figure 1: Model scale (number of parameters) and data scale (number of consumed training tokens) of representative language models in the last 5 years (Devlin et al. 2019; Shoeybi et al. 2019; Brown et al. 2020; Scao et al. 2022; Chowdhery

Random-LTD: Random and Layerwise Token Dropping Brings Efficient Training for Large-scale Transformers

Zhewei Yao*, Xiaoxia Wu*, Conglong Li, Connor Holmes
Minjia Zhang, Cheng Li, Yuxiong He
Microsoft
{zheweiyao, xiaoxiawu, conglong.li, connorholmes, minjiaz, chengli1, yuxhe}@microsoft.com

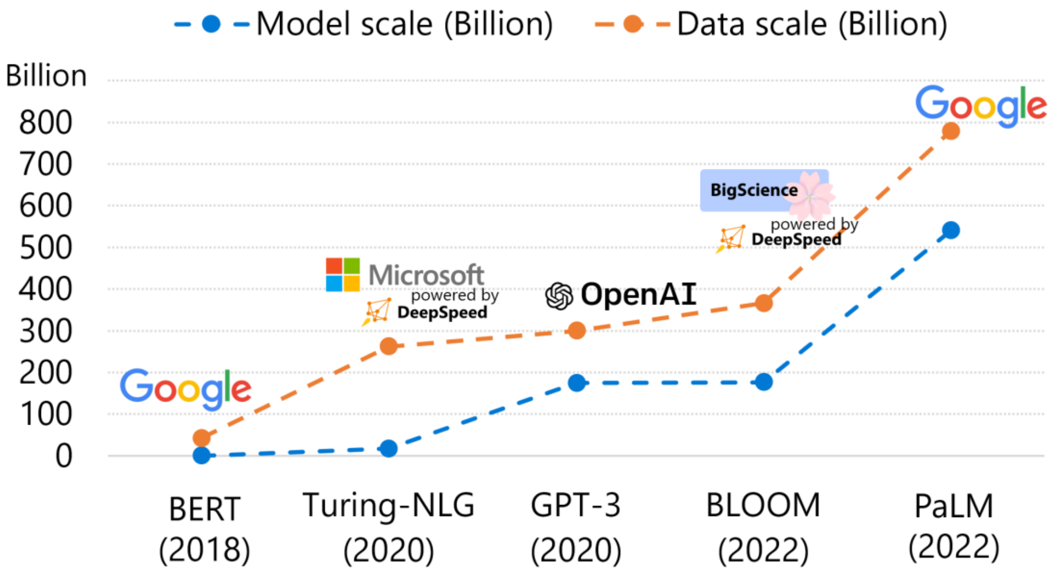
November 22, 2022

Abstract

Large-scale transformer models have become the de-facto architectures for various machine learning applications, e.g., CV and NLP. However, those large models also introduce prohibitive training costs. To

DeepSpeed Data Efficiency

- Why we care about data efficiency
  - Training cost = O(model scale * data scale)
  - Data scale is increasing as fast as model scale
- Our goal
  - Achieve same model quality with less data
  - Achieve better model quality with same data
  - No/minimal model architecture change



The Stability-Efficiency Dilemma

- When pretraining large-scale language models
  - We want large batch sizes & learning rates to increase **training efficiency**
  - But they affect **training stability**, causing poor convergence or divergence

The Stability-Efficiency Dilemma

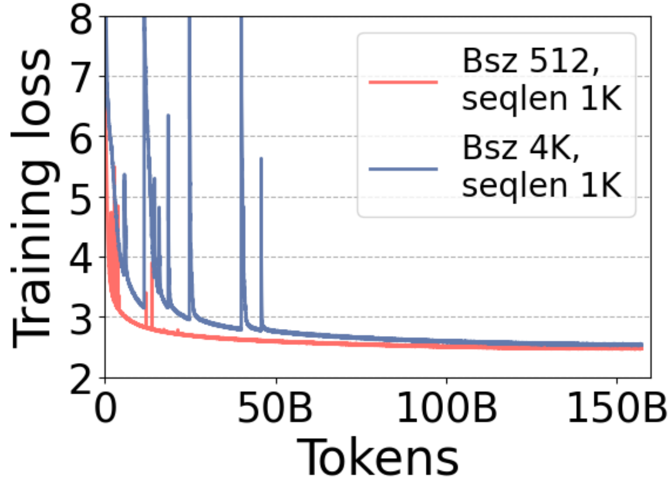
- When pretraining large-scale language models
  - We want large batch sizes & learning rates to increase **training efficiency**
  - But they affect **training stability**, causing poor convergence or divergence
- We study this dilemma by in-depth analysis on GPT-2 pretraining
  - Larger batch sizes & LR reduces training time, but affect model quality
  - (In paper) A correlation between training instability and gradient variance

GPT-2 1.5B model	Training time	WikiText PPL ↓	LAMBADA Acc ↑
Batch size 512	341Hr	13.89	57.29%
Batch size 4K, 4x LR	151Hr	14.76	55.06%

The Stability-Efficiency Dilemma

- When pretraining large-scale language models
  - We want large batch sizes & learning rates to increase **training efficiency**
  - But they affect **training stability**, causing poor convergence or divergence
- We study this dilemma by in-depth analysis on GPT-2 pretraining
  - Larger batch sizes & LR reduces training time, but affect model quality
  - (In paper) A correlation between training instability and gradient variance

GPT-2 1.5B model	Training time	WikiText PPL ↓	LAMBADA Acc ↑
Batch size 512	341Hr	13.89	57.29%
Batch size 4K, 4x LR	151Hr	14.76	55.06%

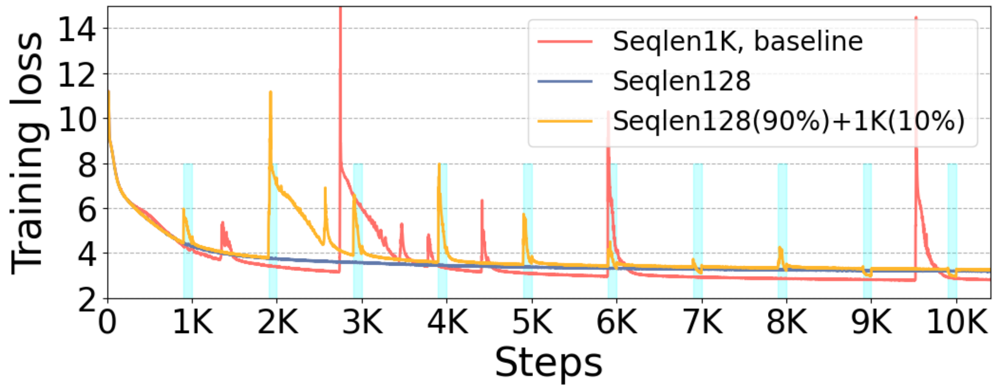


Proposed Sequence Length Warmup Method

- Instability mostly at early stage → some sort of “warmup” is needed
  - LR and batch size warmup didn’t help

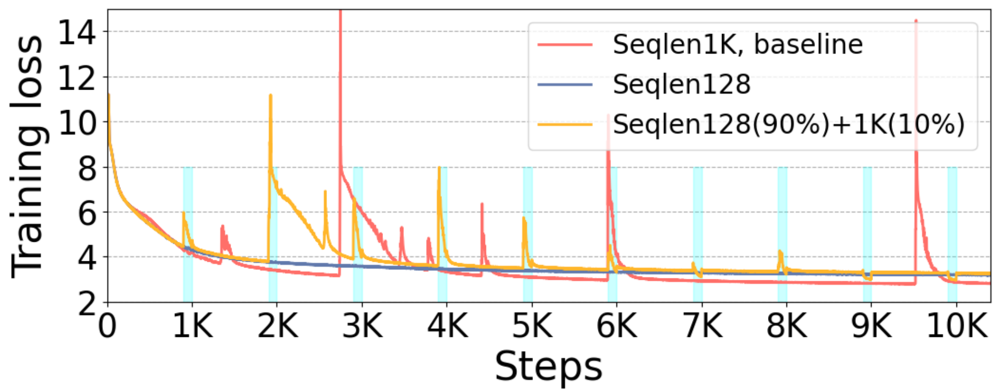
Proposed Sequence Length Warmup Method

- Instability mostly at early stage → some sort of "warmup" is needed
  - LR and batch size warmup didn't help
  - Our study shows warming up sequence length is promising



Proposed Sequence Length Warmup Method

- Instability mostly at early stage → some sort of "warmup" is needed
  - LR and batch size warmup didn't help
  - Our study shows warming up sequence length is promising



- The Sequence Length Warmup (SLW) method
  - Two configs: starting sequence length, number of warmup steps
  - (In paper) Simple truncation-based implementation, low-cost tuning strategy

GPT-2 Evaluation

- Stable training under large batch size & LR
- Same model quality under less token/time

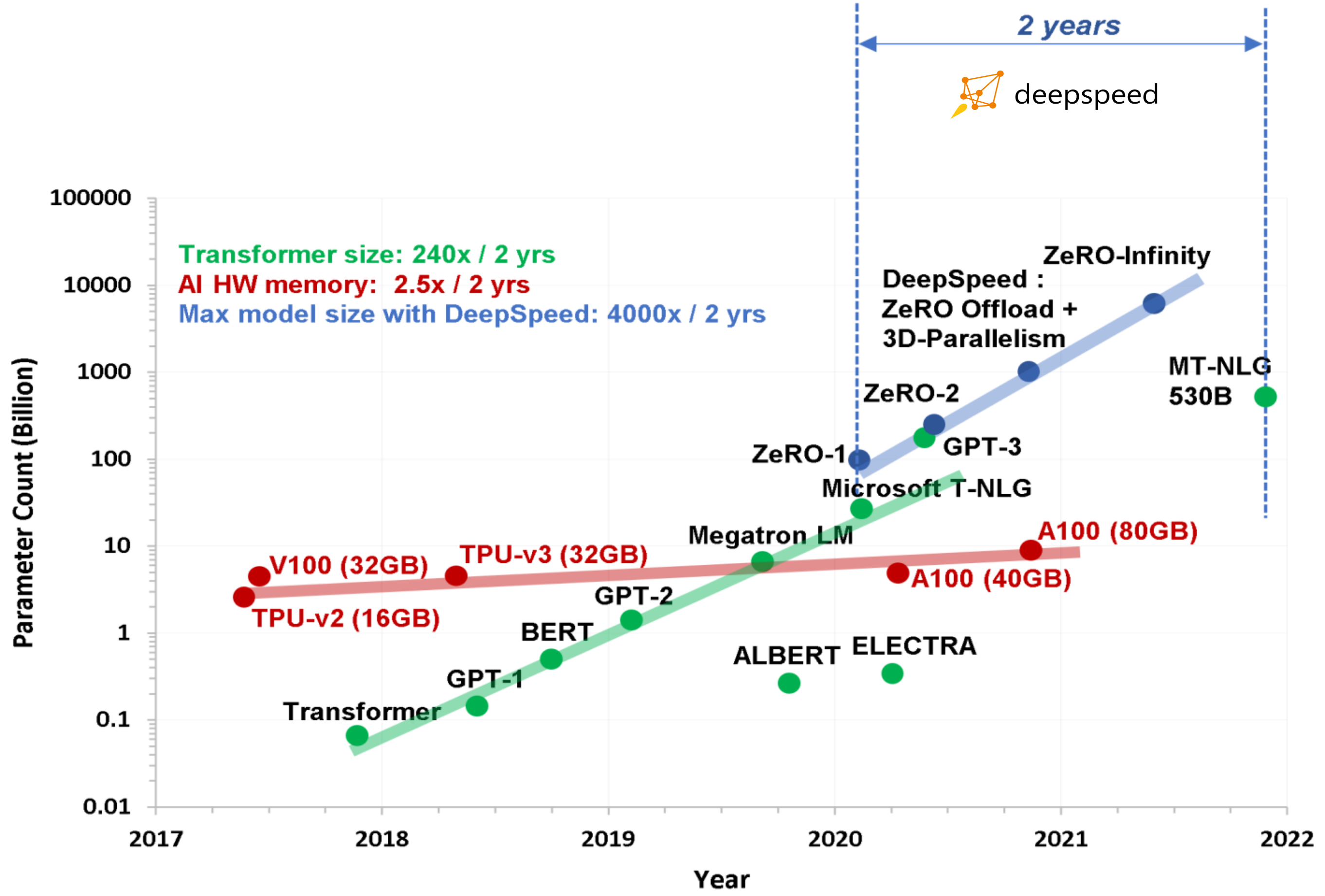
GPT-2 1.5B model	Training time	Training tokens	WikiText PPL ↓	LAMBADA Acc ↑
Batch size 512, baseline	341Hr	157B	13.89	57.29%
Batch size 4K, 4x LR, baseline	151Hr	157B	14.76	55.06%
Batch size 4K, 4x LR, ours	121Hr	121B	13.88	58.20%

GPT-2 Evaluation

- Stable training under large batch size & LR
- Same model quality under less token/time
- Even better model quality under same token/time

GPT-2 1.5B model	Training time	Training tokens	WikiText PPL ↓	LAMBADA Acc ↑
Batch size 512, baseline	341Hr	157B	13.89	57.29%
Batch size 4K, 4x LR, baseline	151Hr	157B	14.76	55.06%
Batch size 4K, 4x LR, ours	121Hr	121B	13.88	58.20%
Batch size 4K, 4x LR, ours	155Hr	157B	13.72	58.47%

DeepSpeed: Reshaping the Large Model Training Landscape



*System capability to efficiently train models with **trillions of parameters***

*[AI and Memory Wall. (This blogpost has been written in... | by Amir Gholami | riselab | Medium]

Powered Massive Models

- METRO-LM **(5.4B)**
- Microsoft-Turing NLG (**17B**)
- GPT Neo-X (**20B**)
- AlexaTM **(20B)**
- IDEFICS **(80B)**
- YaLM (**100B**)
- GLM (**130B**)
- BLOOM: Big Science (**176B**)
- Jurrasic-1 (**178B**) AI21 labs
- Megatron-Turing NLG (**530B**)
- ...

Powered Frameworks

Transformers

Accelerate

Lightning

mosaicML

Determined AI

MMEngine

...

Accelerator support

NVIDIA CUDA

AMD ROCm

intel GAUDI

intel XEON

...

DeepSpeed Transformation

deepspeed

Past

Training optimization library

Training
- Speed
- Scale
- Cost
- Democratization

Today and Future

Multi-purpose DL optimization suite

Training
- Speed Scale Cost
- Democratization
- MoE models
- Long sequence
- RLHF

Inference
- Large models
- Latency
- Serving cost
- Agility

Compression
- Model size
- Latency
- Composability
- Runnable on client devices

Science
- Speed
- Scale
- Capability
- Diversity

Microsoft

deepspeed

We welcome contributions! Make your first pull request ☺

https://github.com/microsoft/DeepSpeed

www.deepspeed.ai

Follow us on X: @MSFTDeepSpeed

Thank You!

ZeRO-Infinity in Action