Colossal-AI

Training large AI models

Code & Tutorial

Slack Q&A

https://github.com/hpcaitech/ColossalAI

James Demmel

demmel@berkeley.edu

- ✔ UC Berkeley Distinguished Professor
- ✔ National Academy of Sciences (USA)
- ✔ National Academy of Engineering (USA)
- ✔ IEEE/ACM/SIAM/AMS/AAAS Fellow

Berkeley UNIVERSITY OF CALIFORNIA · NATIONAL ACADEMY OF SCIENCES · IEEE

Yang You

youy@comp.nus.edu.sg

- ✔ PhD from UC Berkeley
- ✔ IEEE-CS Early Career Excellence Award
- ✔ Presidential Young Professor at NUS
- ✔ Most cited fresh PhD in HPC (2020)

Berkeley UNIVERSITY OF CALIFORNIA · NUS National University of Singapore · IEEE

We'll start soon

https://github.com/hpcaitech/ColossalAI

Join Colossal-AI Slack!
Get this slides and Q&A

Colossal-AI

Colossal-AI: Scaling AI Models in Big Model Era

Code & Tutorial

Slack Q&A

https://github.com/hpcaitech/ColossalAI

James Demmel

demmel@berkeley.edu

- ✔ UC Berkeley Distinguished Professor
- ✔ National Academy of Sciences (USA)
- ✔ National Academy of Engineering (USA)
- ✔ IEEE/ACM/SIAM/AMS/AAAS Fellow

Berkeley UNIVERSITY OF CALIFORNIA · NATIONAL ACADEMY OF SCIENCES · IEEE

Yang You

youy@comp.nus.edu.sg

- ✔ PhD from UC Berkeley
- ✔ IEEE-CS Early Career Excellence Award
- ✔ Presidential Young Professor at NUS
- ✔ Most cited fresh PhD in HPC (2020)

Berkeley UNIVERSITY OF CALIFORNIA · NUS National University of Singapore · IEEE

Outline

1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Outline

1. Introduction of Big Model Era

2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

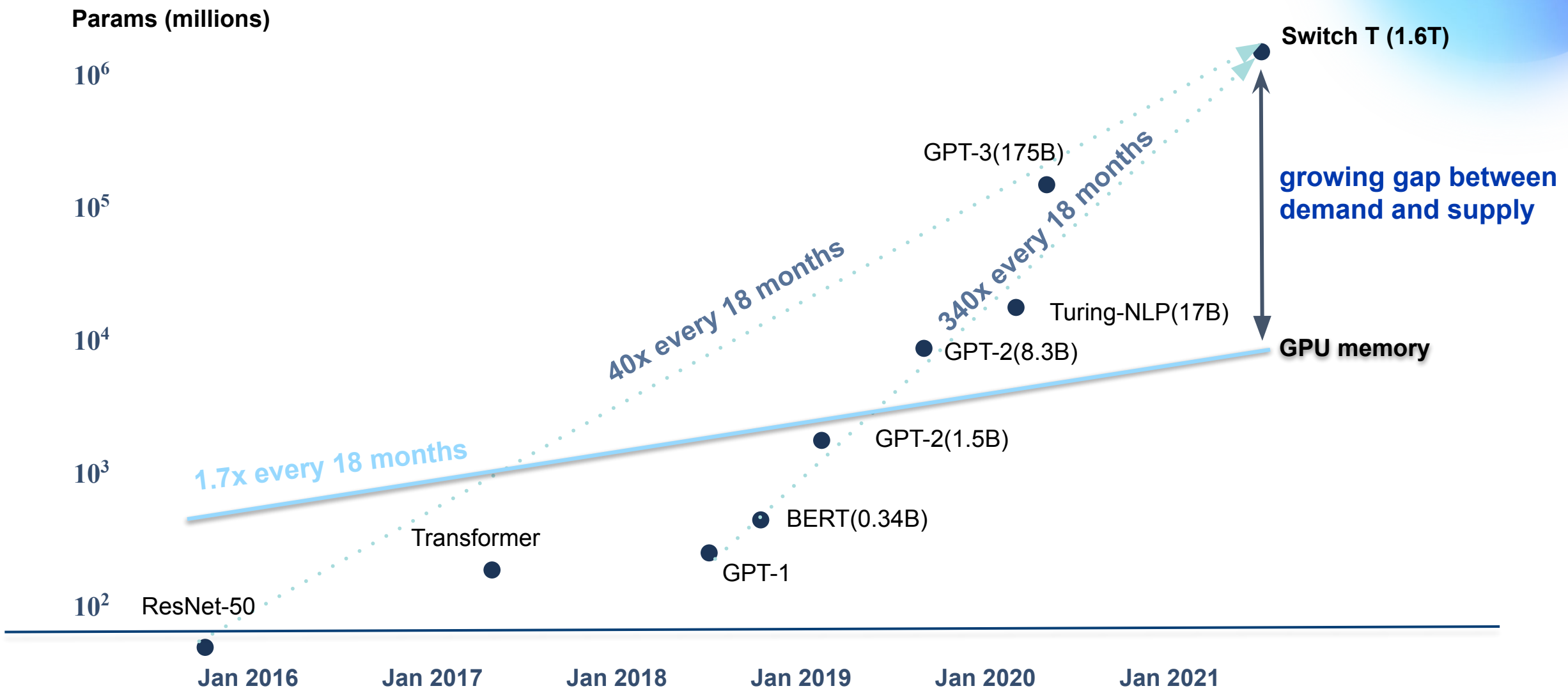
3. Automatic Parallelism System

4. Efficient Memory System

5. Outstanding Performance & Use Cases

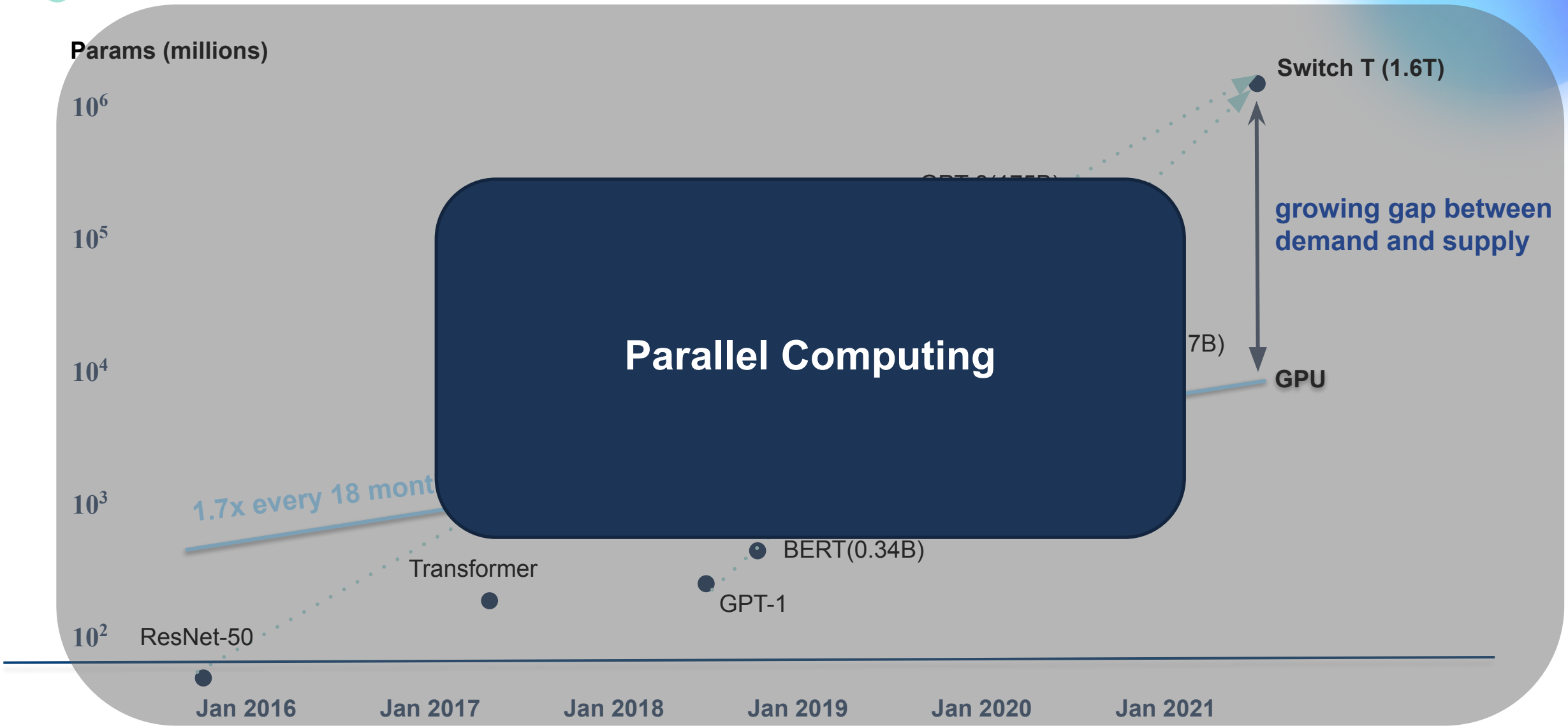
Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Problem



https://www.youtube.com/watch?v=tgB671SFS4w

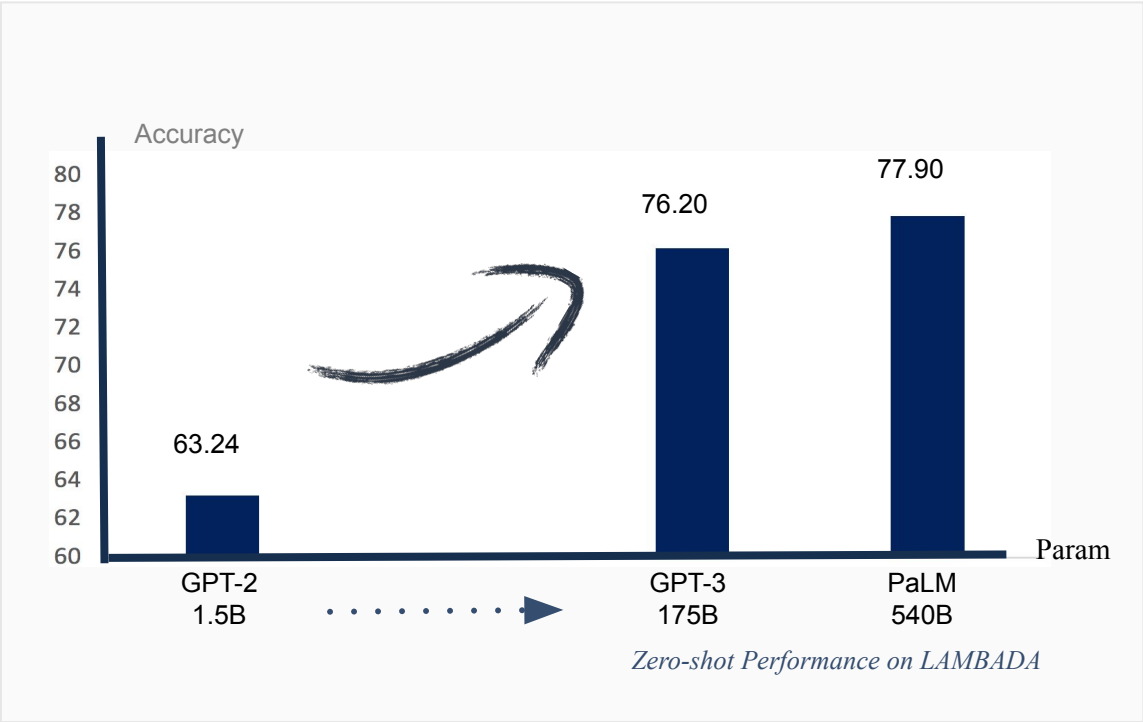
The Practical Solution in Next 10 Years



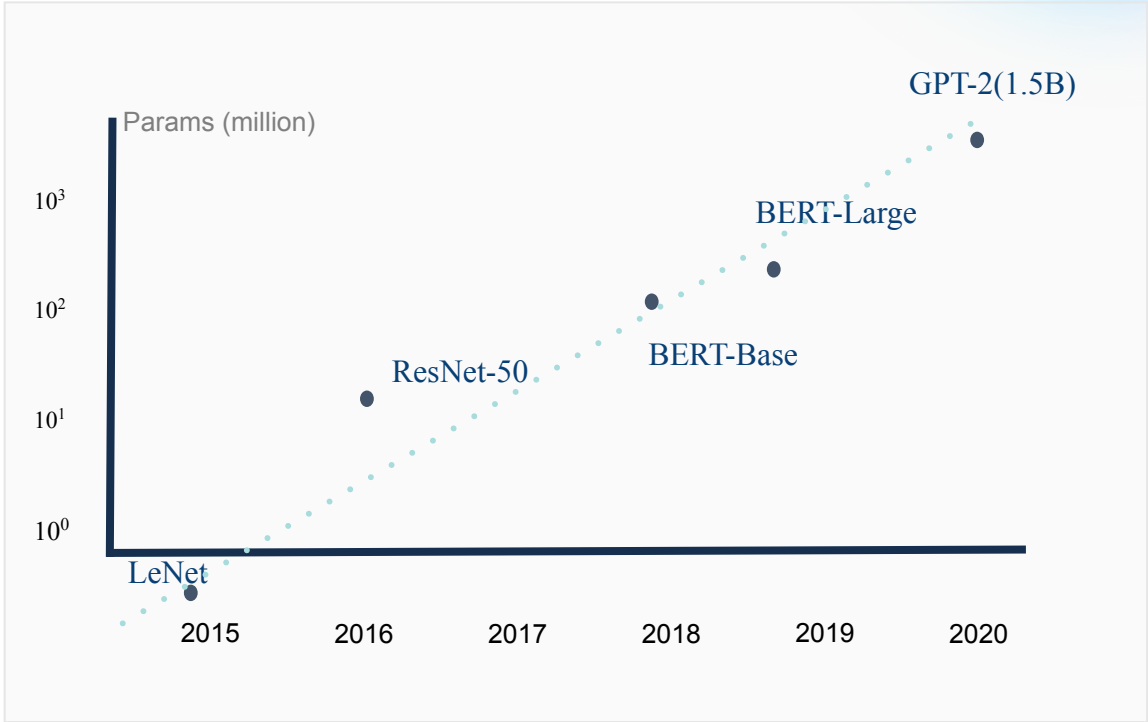
https://www.youtube.com/watch?v=tgB671SFS4w

Why Do We Believe in Large Models?

Larger Model: Better Performance

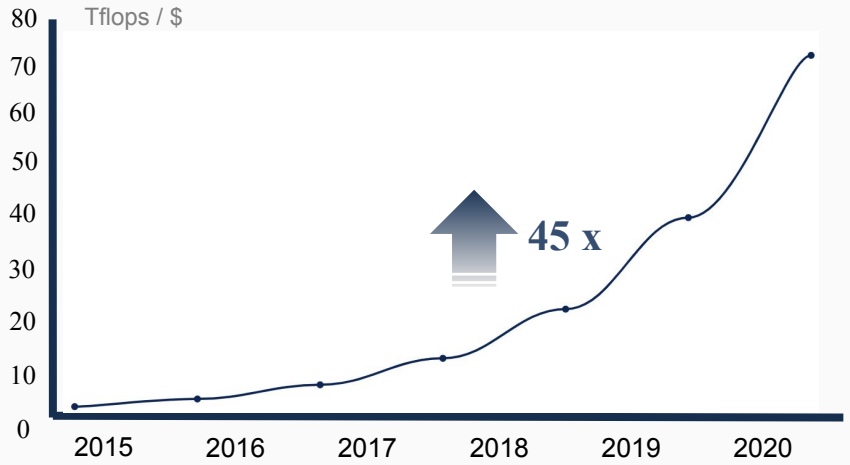


Models Used in SMEs are Growing Exponentially



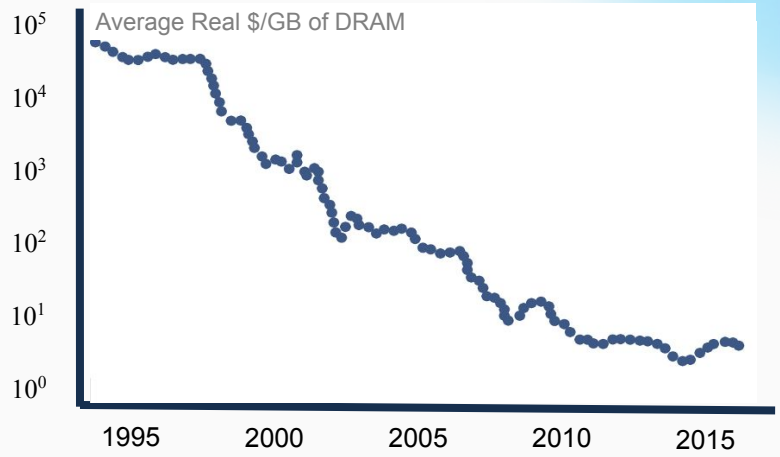
"Small Models" are Growing Exponentially

Why Do We Believe in Large Models?

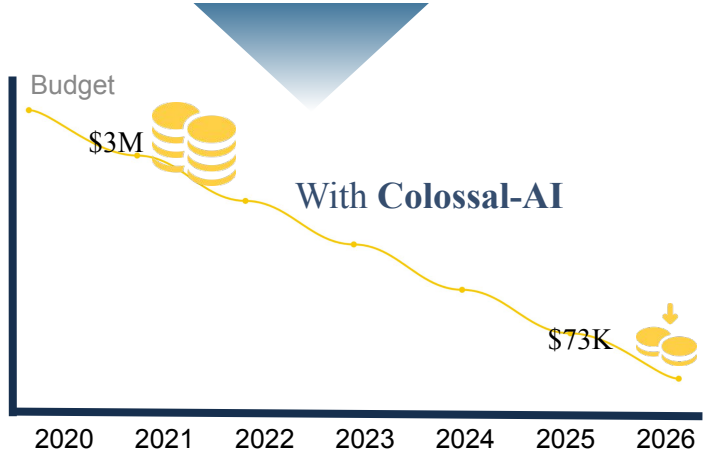


#Tflops per dollar is rising around **1.86x** per year

Prices of GPU and Memory
Are Falling **Exponentially**



The price / GB of DRAM -33% per year



Training Cost of GPT-3 in 2026 can be reduced to $73K

Challenges of Using Large AI Models

PaLM：300 years by 1 NV A100 GPUs, $9.2M+

Training

- **GPT-2 (2019):** “COVID-19 is a high capacity LED-emitter.”
- **GPT-J (2021):** “COVID-19 is a novel coronavirus.”

needs to **re-train** on new data repeatedly

Inference Fine-tuning

- A cluster of GPUs is required simply to load & make predictions
- GPT-3: 2400+ GB; NV A100 GPU: 80 GB

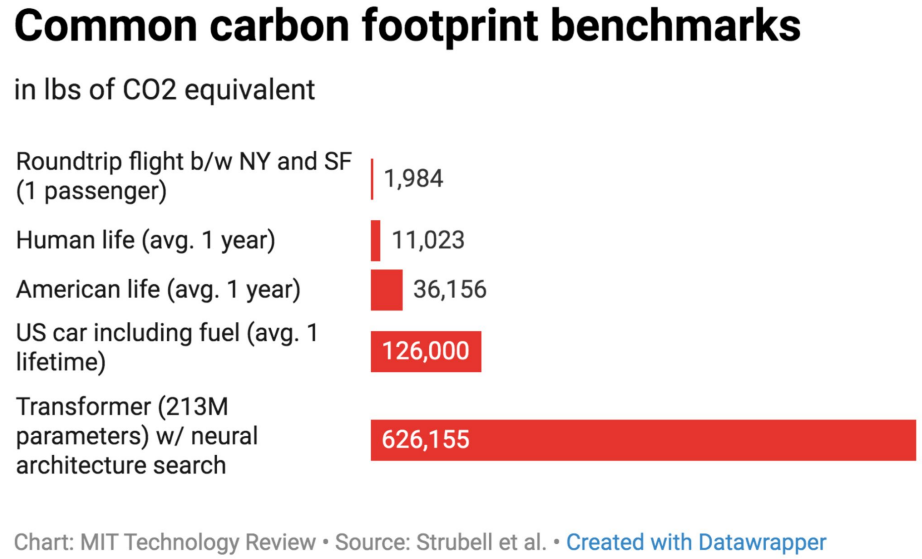
single GPU server is **out-of-memory**

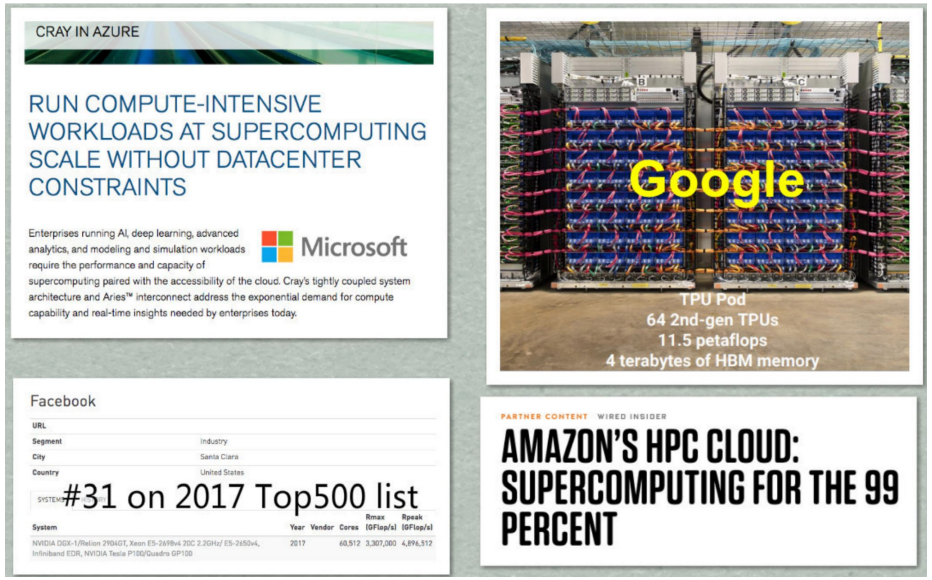
Deployment

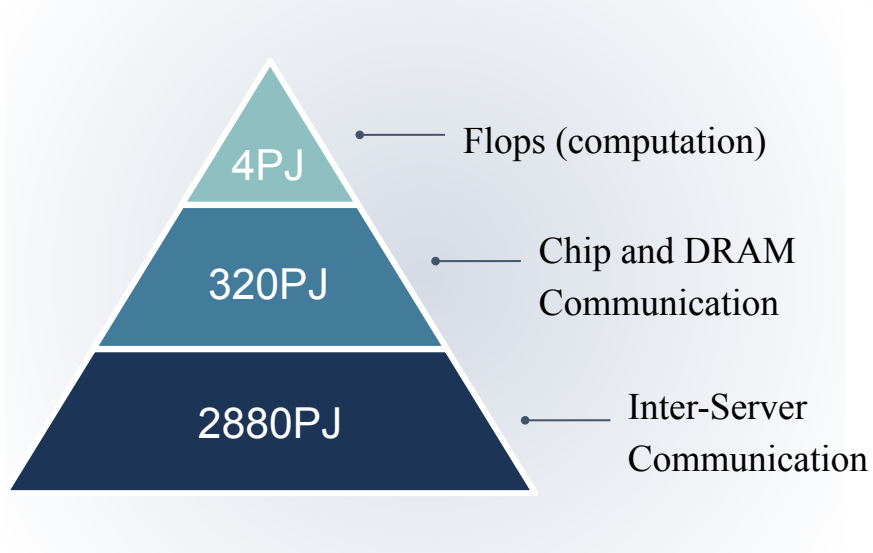
A company needs 70 people building their internal tools for AI: $20M per year (impossible for startups)

Expensive Infrastructure and Systems

Large Model Consumes a Lot of Energy







Training GPT-3

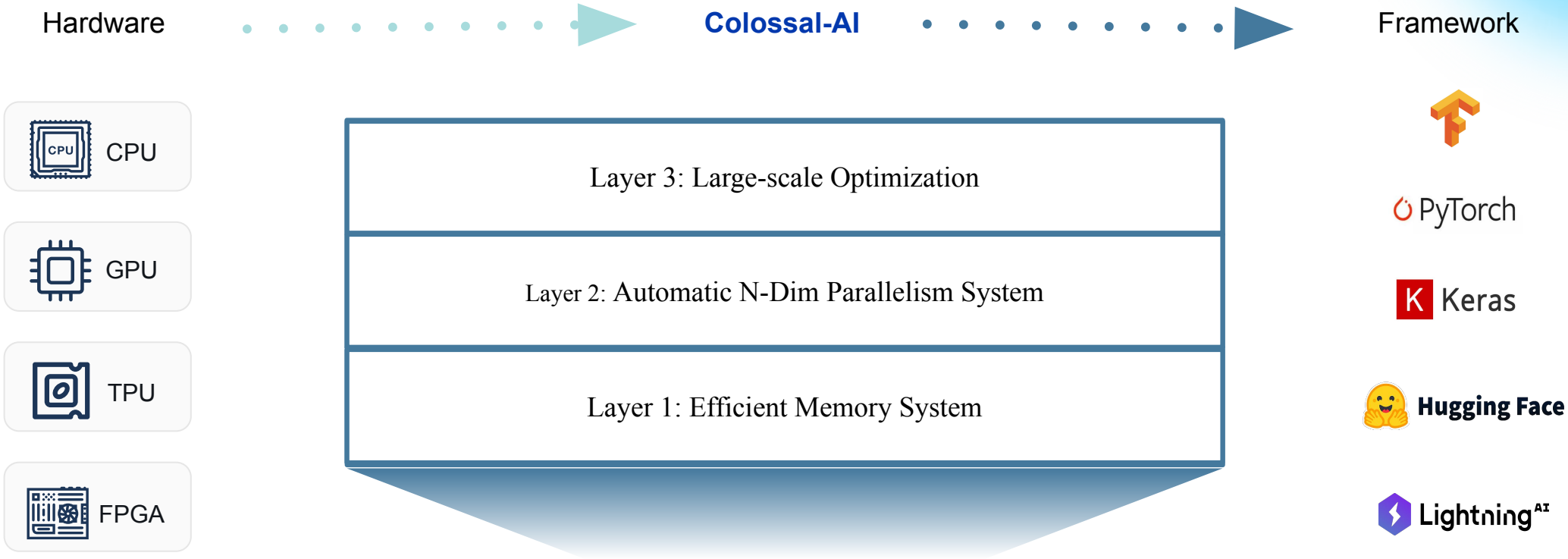
Cost: 12 million USD

Communication: 1.32e+24 pJ

Computation: 1.26e+24 pJ

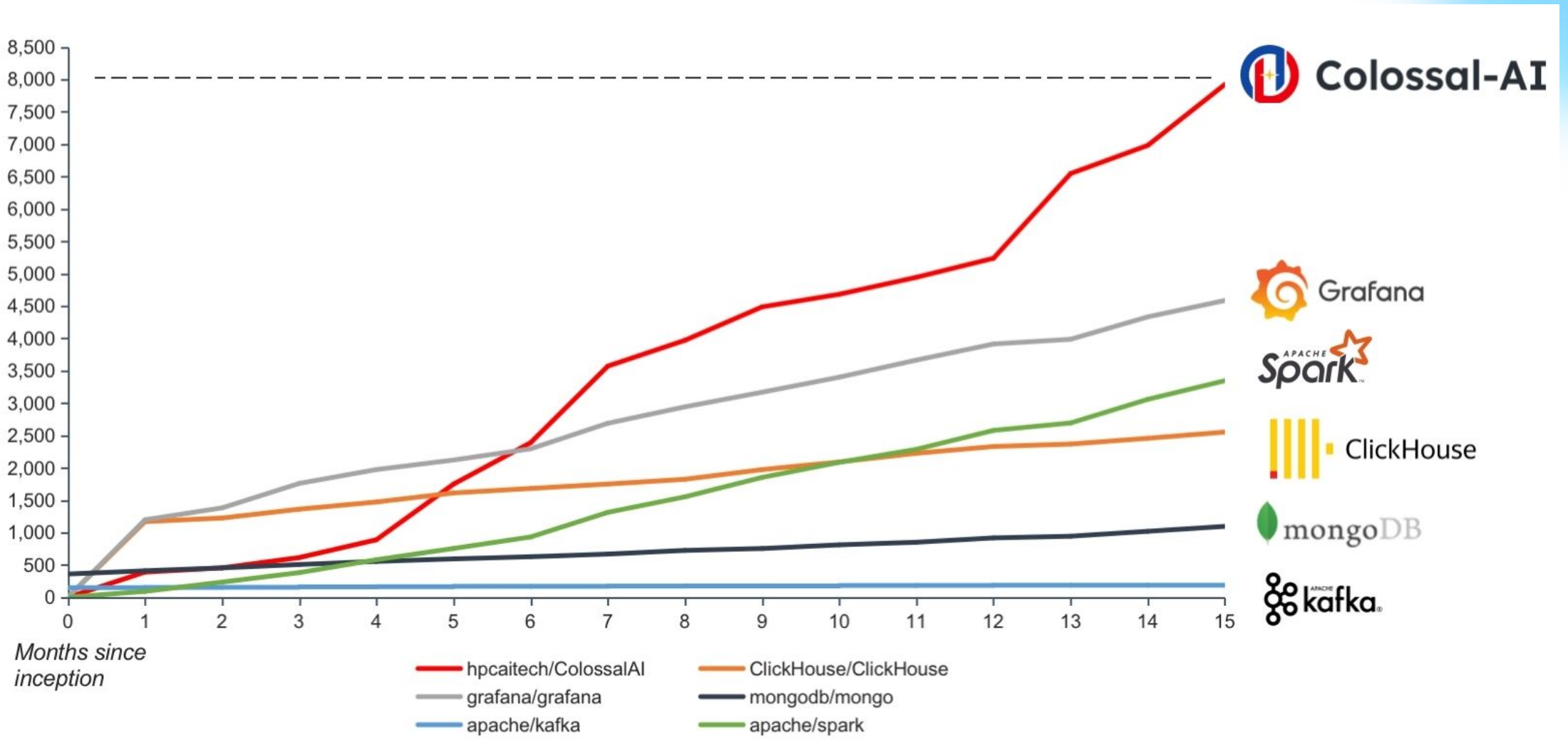
source: NUS CS5260

Colossal-AI = Performance + Efficiency + Cheapness



- Maximize computational efficiency
- Minimize system running time
- Minimize communication
- Minimize code refactoring
- Dynamic adaptive scaling
- Reduce memory footprint

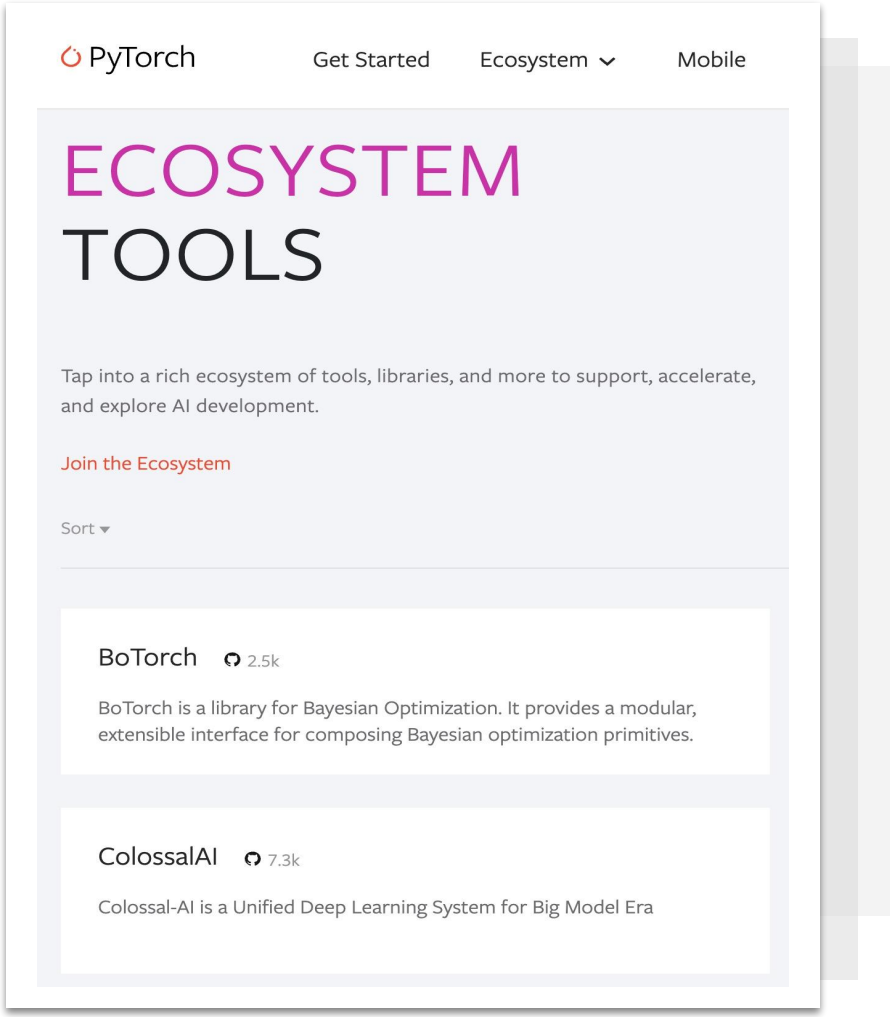
Open Source GitHub Stars



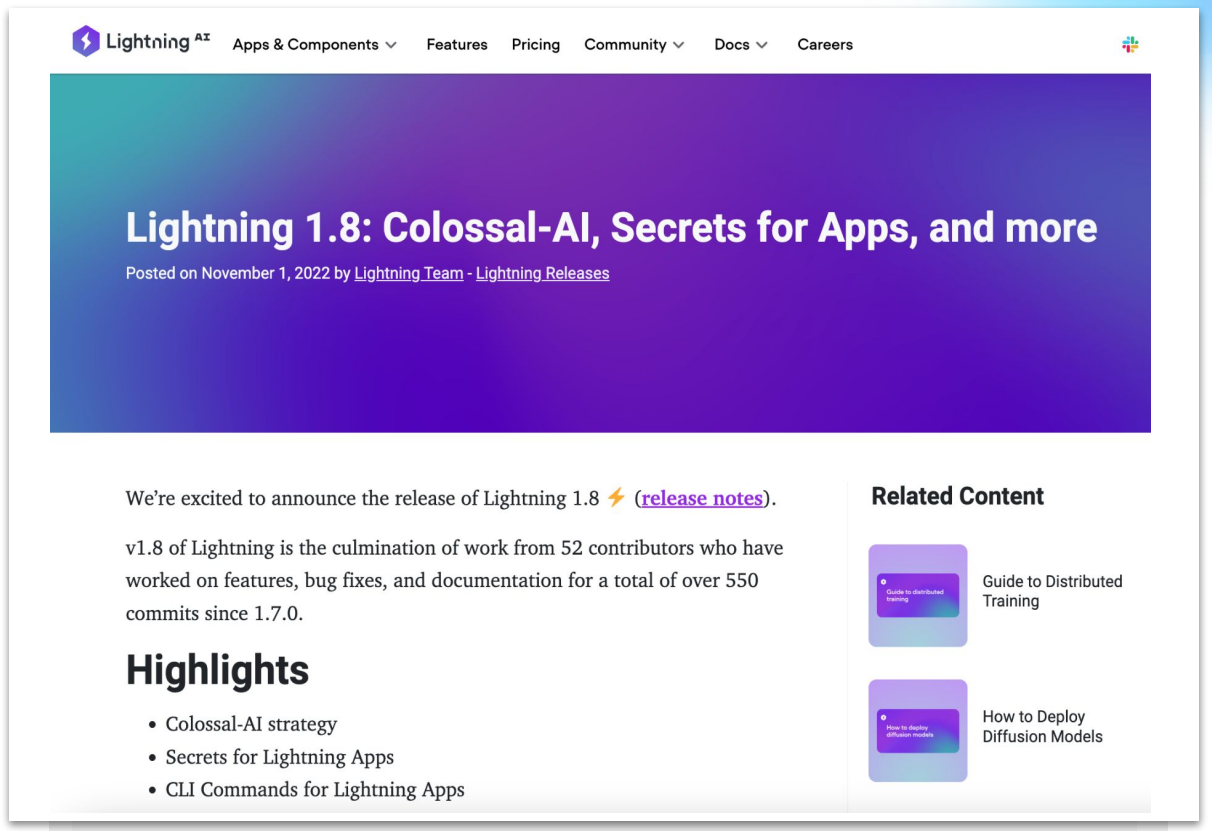
Colossal-AI is public and available at https://github.com/hpcaitech/ColossalAI

Stargazers of Colossal-AI are All Over the World

Users from Global AI Ecosystems

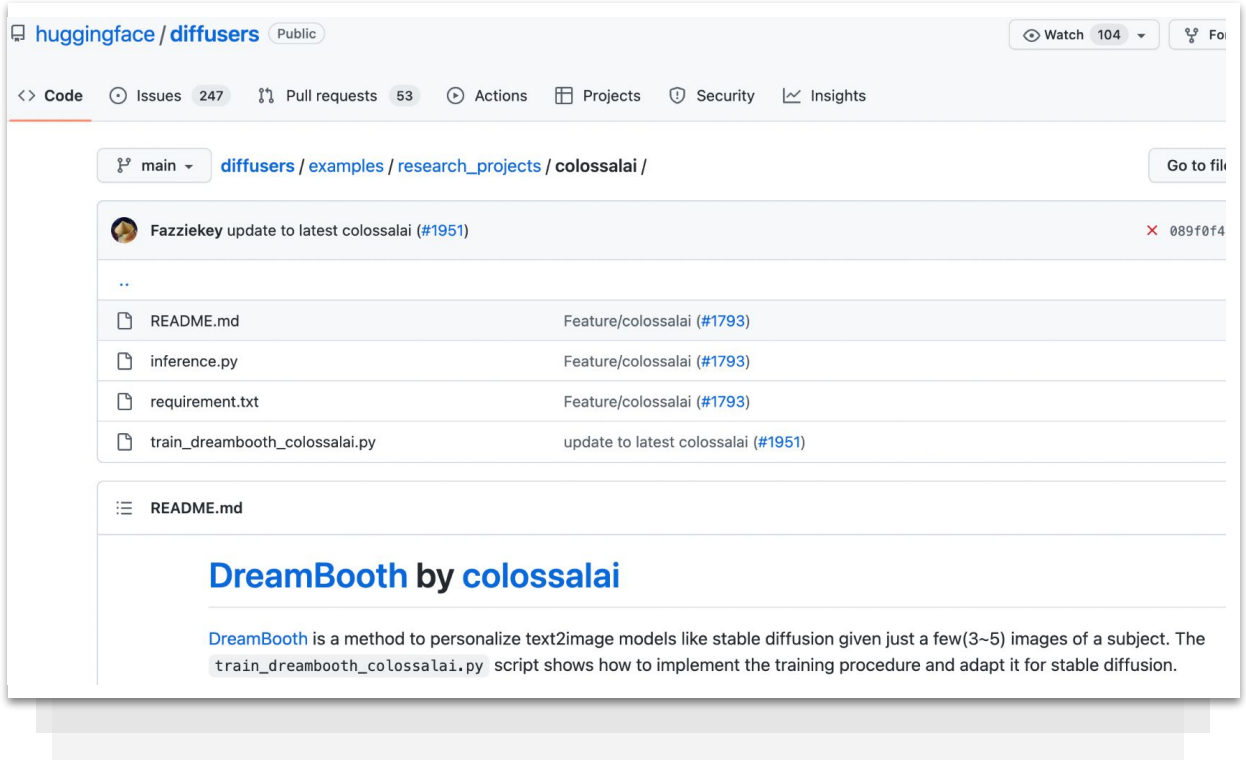


PyTorch Users

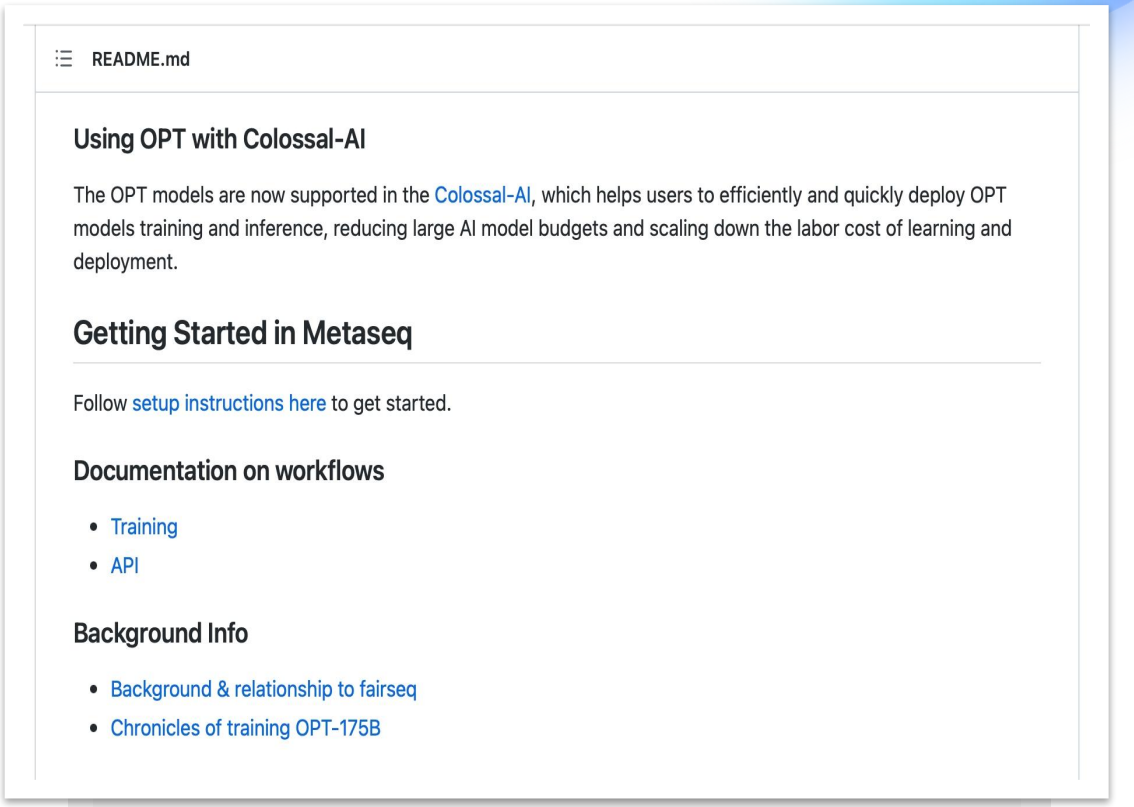


Lightning AI Users

Users from Global AI Ecosystems



Hugging Face Users



Facebook OPT Users

Users of Colossal-AI: All Over the World

Outline

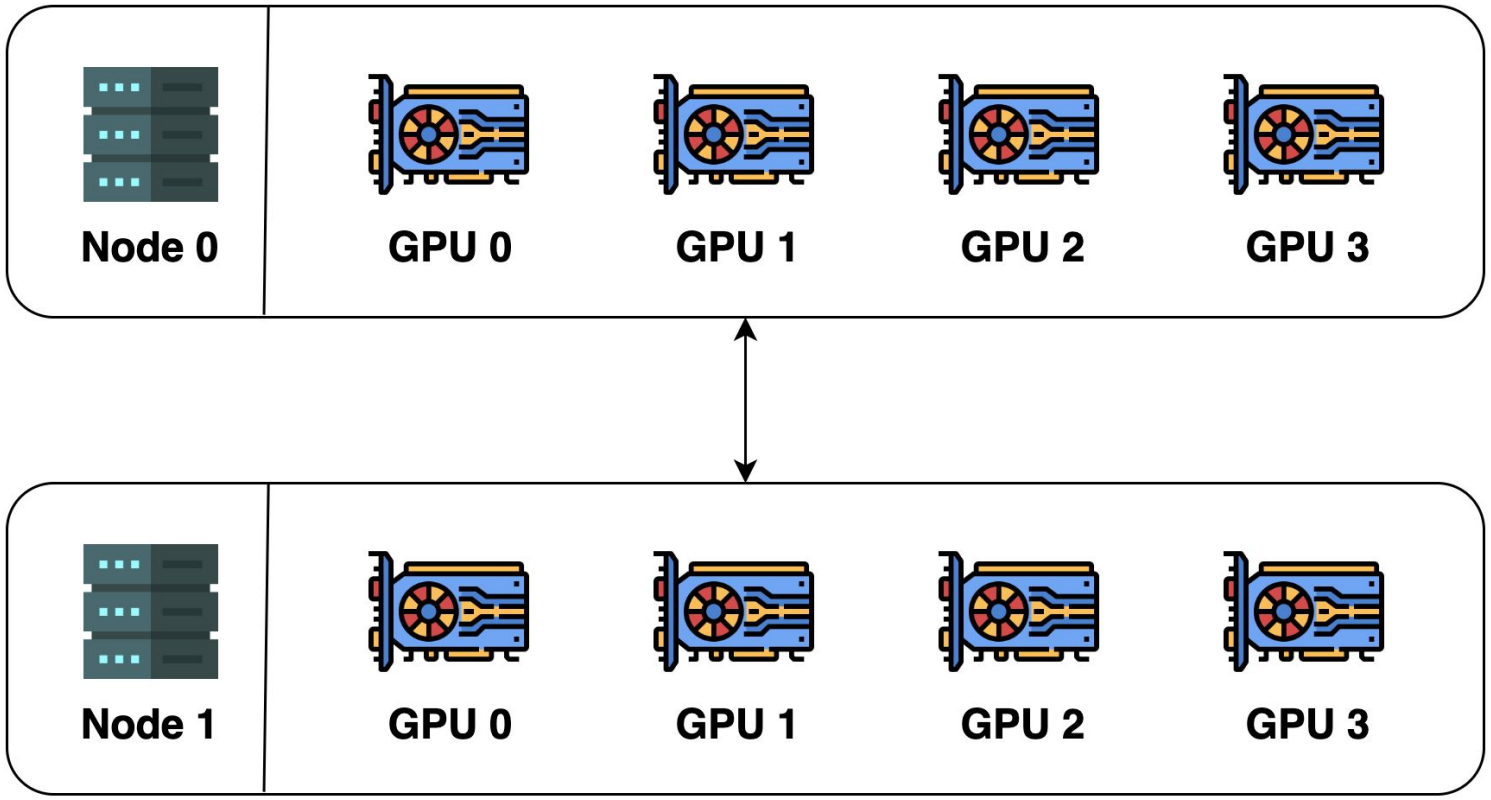
1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

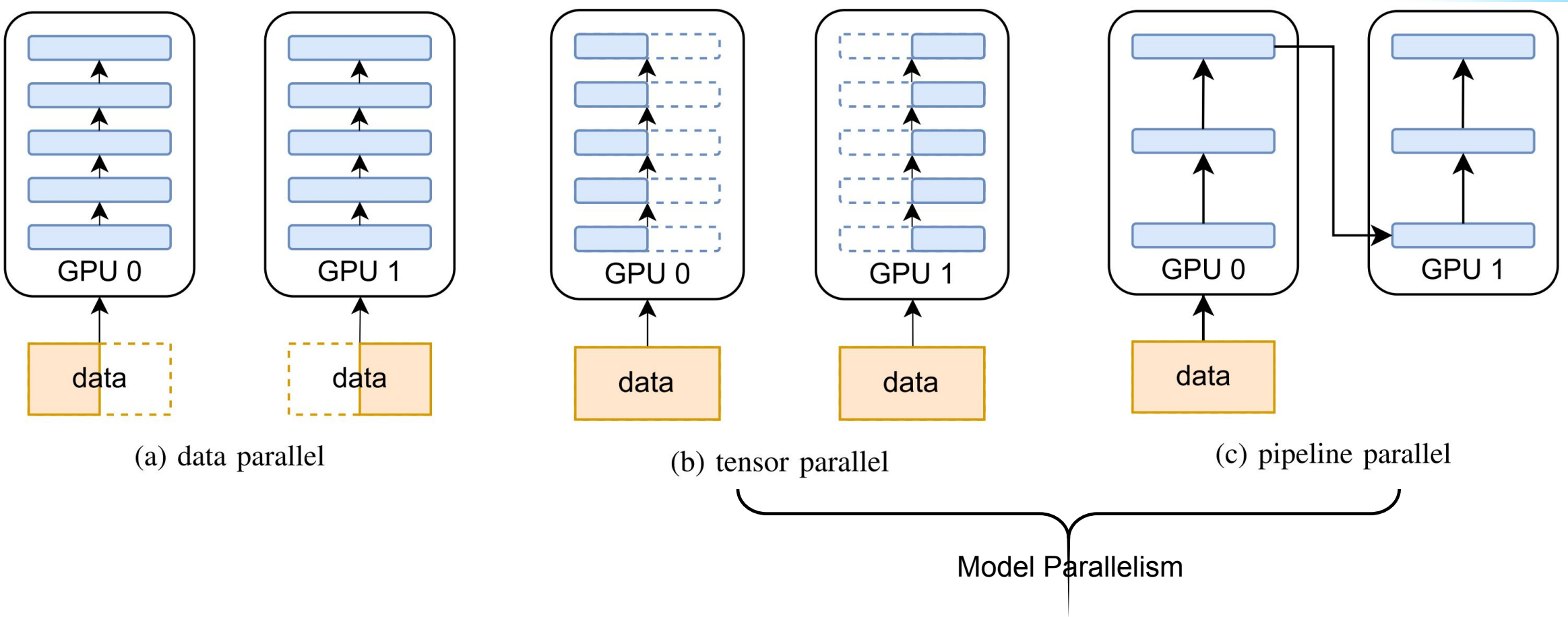
Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Distributed system



- A distributed system example in DL
- https://www.comp.nus.edu.sg/~youy/lecture.pdf

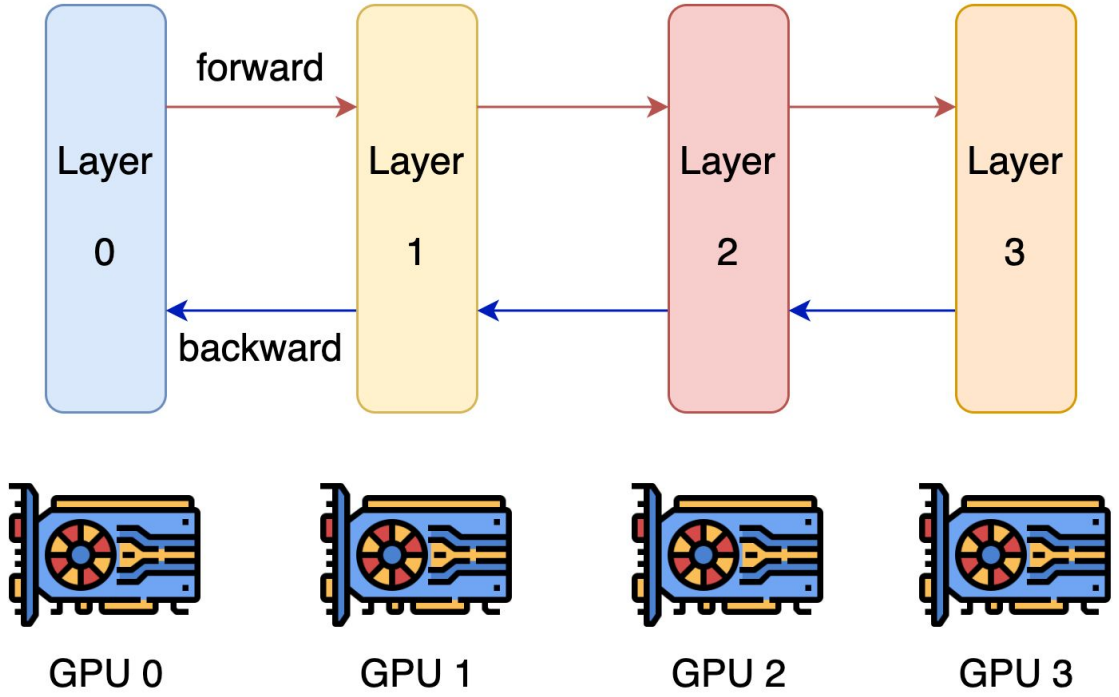
Parallelism



- Existing parallelism for distributed training

Pipeline Parallelism

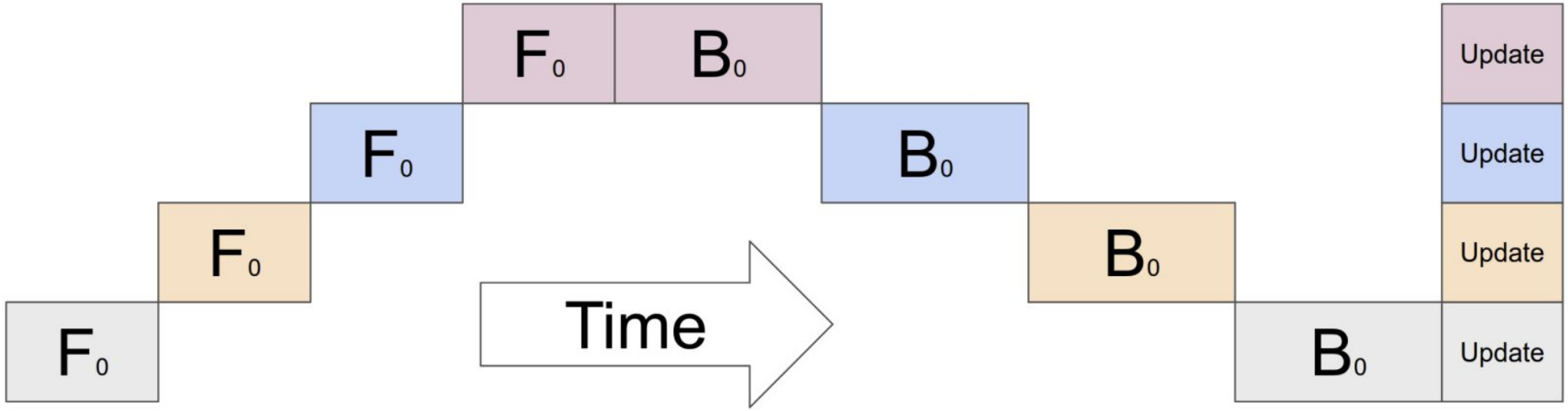
- Model is split by layer into several chunks, each chunk is given to a device.
- Forward: pass the intermediate activation to the next stage.
- Backward: pass the gradient of the input tensor back to the previous pipeline stage.



Pipeline Parallelism

- **Bubble Time**

  One drawback of pipeline parallel training is that there will be some bubble time where some devices are not engaged in computation, leading to waste of computational resources.

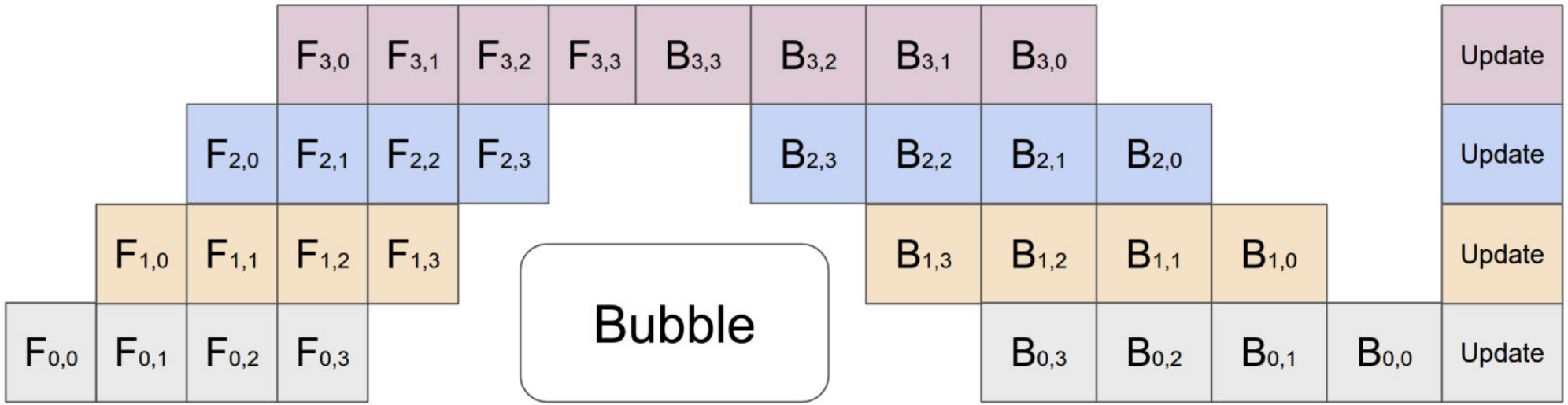


Src: Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems 32 (2019): 103-112.

Pipeline Parallelism

- **Pipelined Execution**

  To alleviate this problem, pipeline parallelism splits the input minibatch into multiple microbatches and pipelines the execution of these microbatches across multiple GPUs.

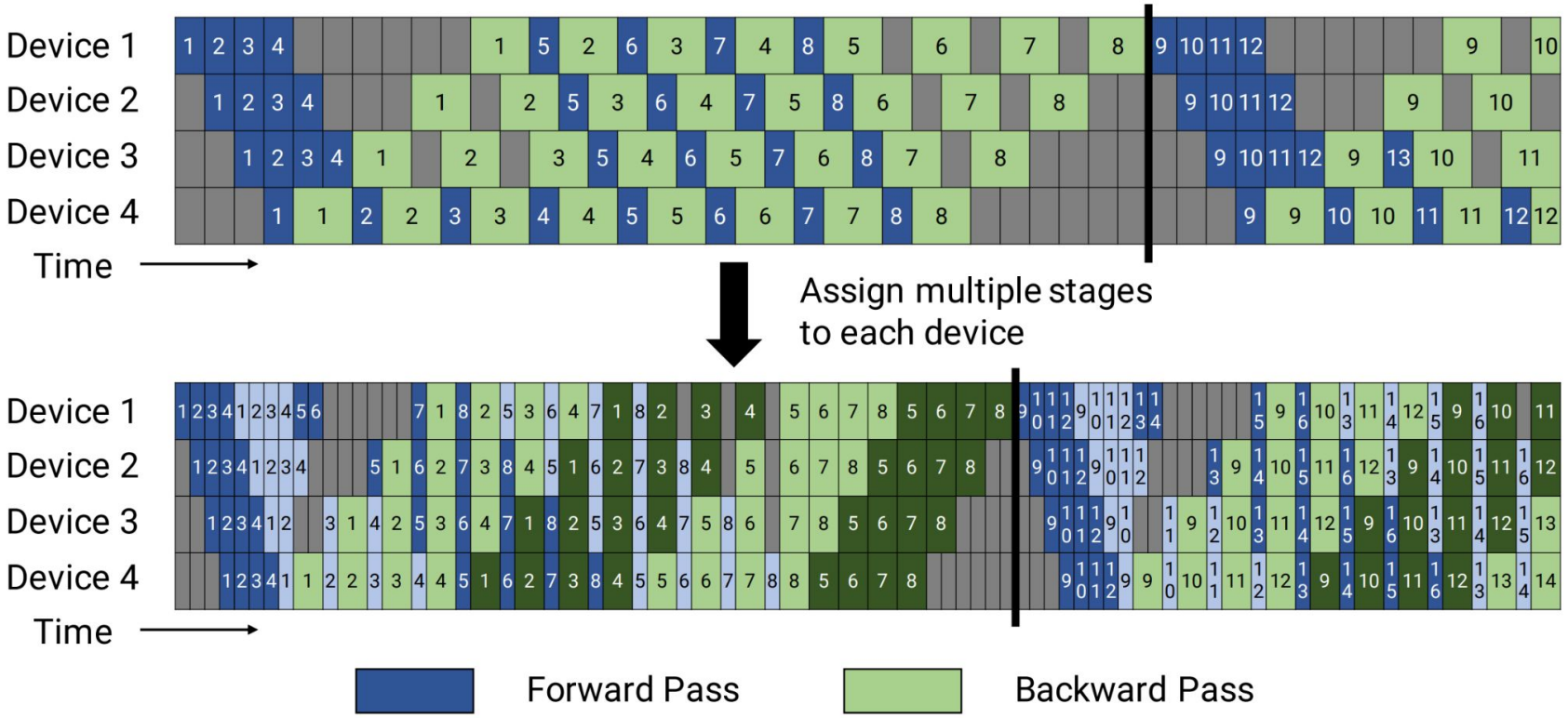


Src: Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems 32 (2019): 103-112.

Pipeline Parallelism

- **Interleaved 1F1B schedule**

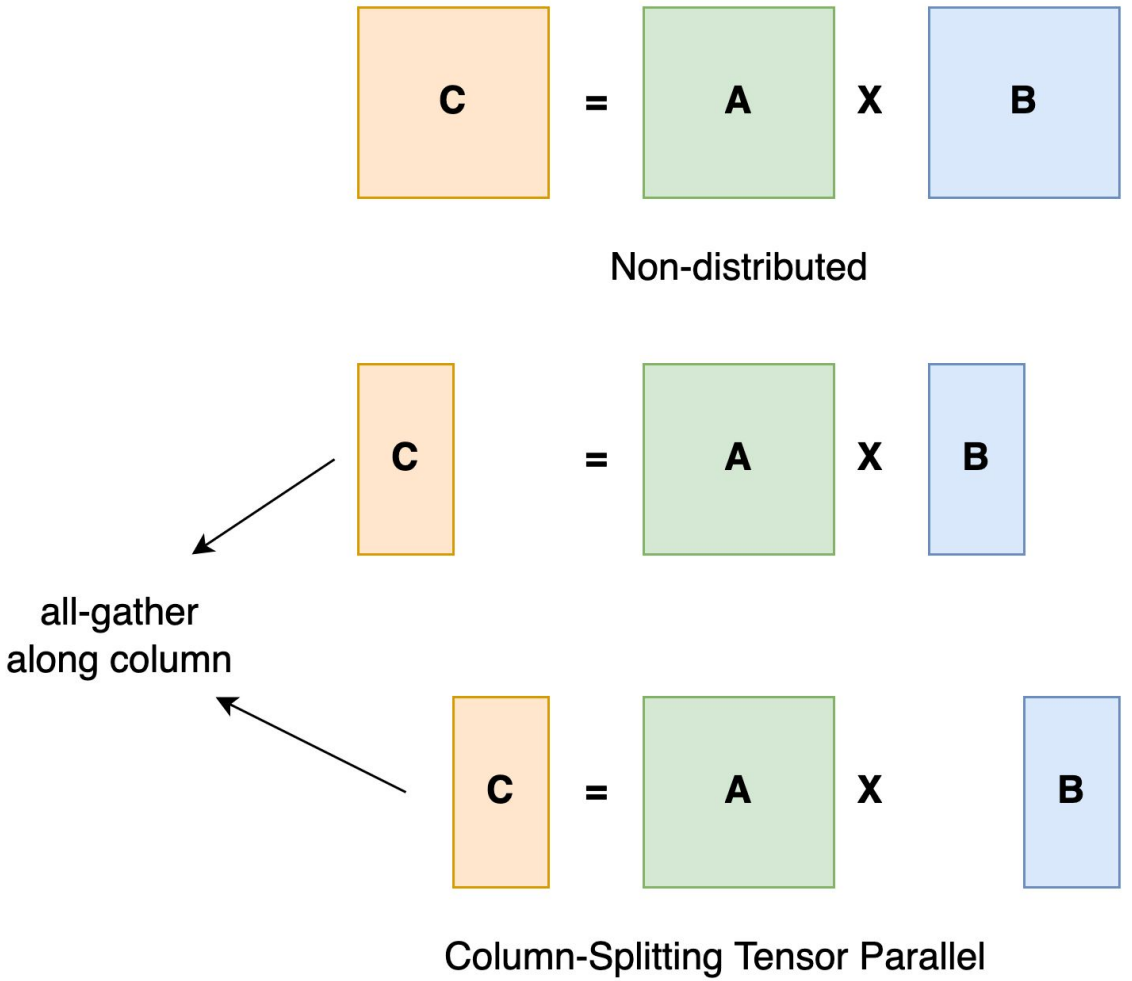
  To reduce the size of the pipeline bubble, each device can perform computation for multiple subsets of layers (called a model chunk), instead of a single contiguous set of layers.



Narayanan, Deepak, et al. "Efficient large-scale language model training on GPU clusters using megatron-LM." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021.

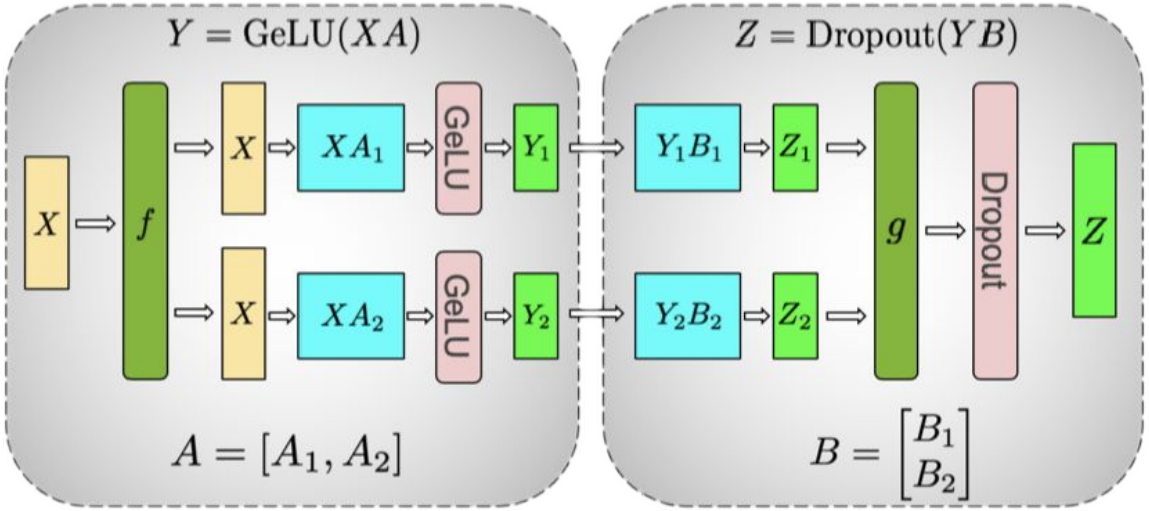
Tensor Parallelism

- Split a tensor into N chunks along a specific dimension
- Each device only holds 1/N of the whole tensor
- Retain correctness of the computation graph

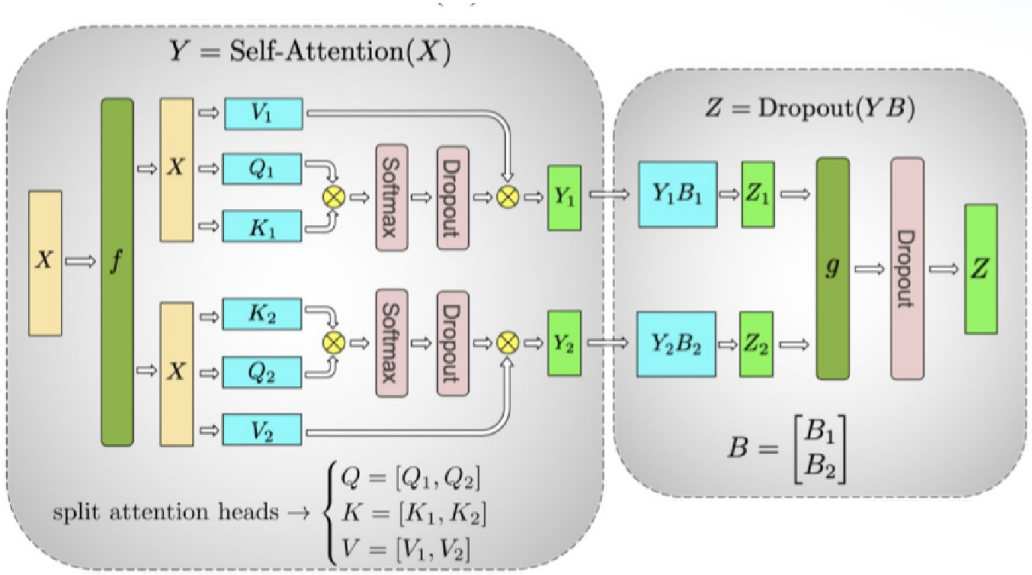


Existing Solutions

- Megatron — NVIDIA
  - Featuring by 1-D tensor splitting



(a) MLP

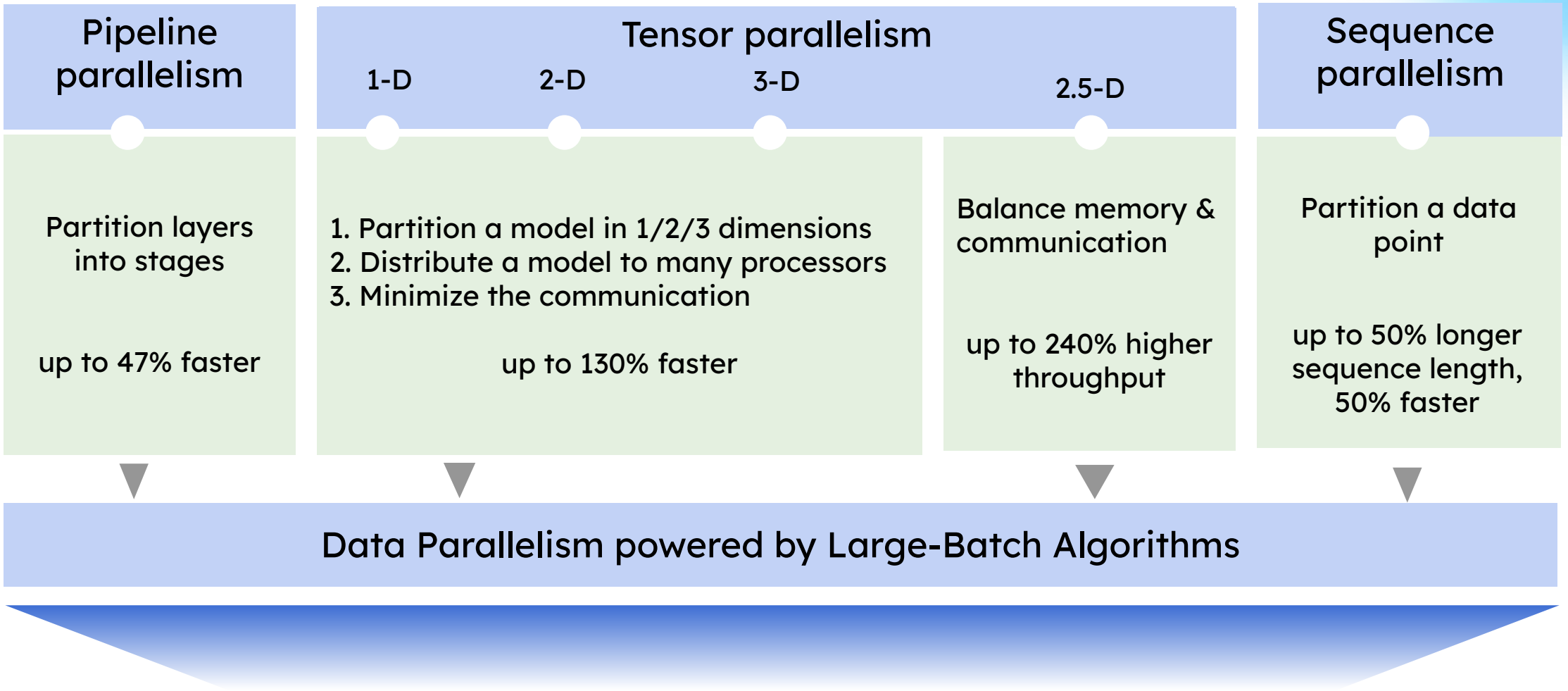


(b) self attention

- DeepSpeed — Microsoft
  - Compatible with Megatron
  - Support Zero Redundancy Optimizer (Eliminate memory redundancies)

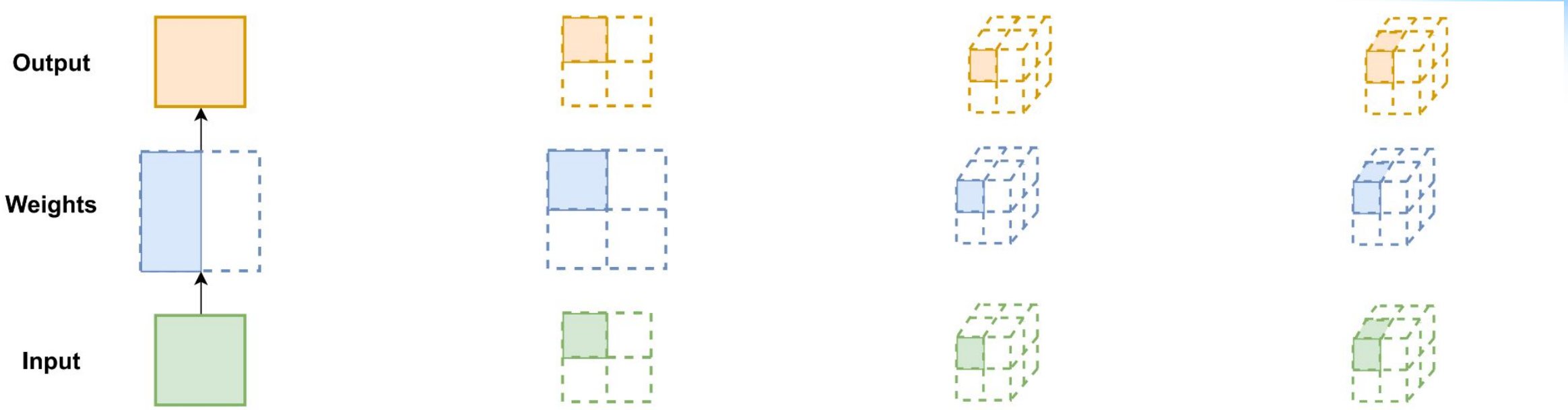
Shoeybi, Mohammad, et al. "Megatron-lm: Training multi-billion parameter language models using model parallelism." arXiv preprint arXiv:1909.08053 (2019).

Overview of N-Dim Parallelism System



All in N-Dim Parallelism System
from Colossal-AI

Tensor Parallelism



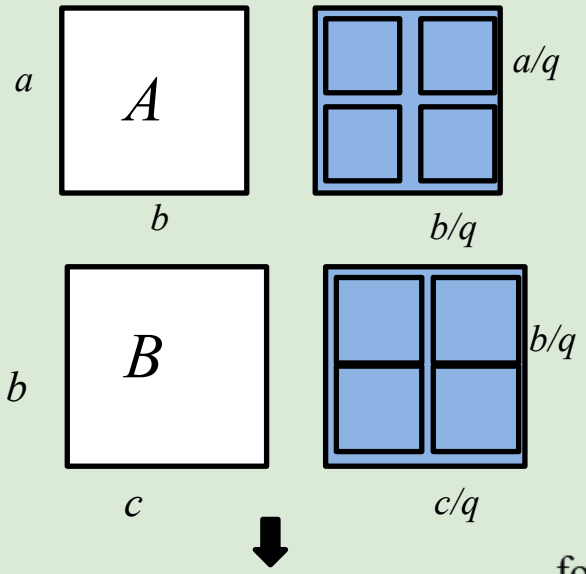
(a) 1D tensor parallelism (b) 2D tensor parallelism (c) 2.5D tensor parallelism (d) 3D tensor parallelism

- Tensor parallel illustration

2-D/2.5-D Tensor Parallelism

2-D Tensor Parallelism

A:[a,b] B:[b,c] Processors:[q,q]

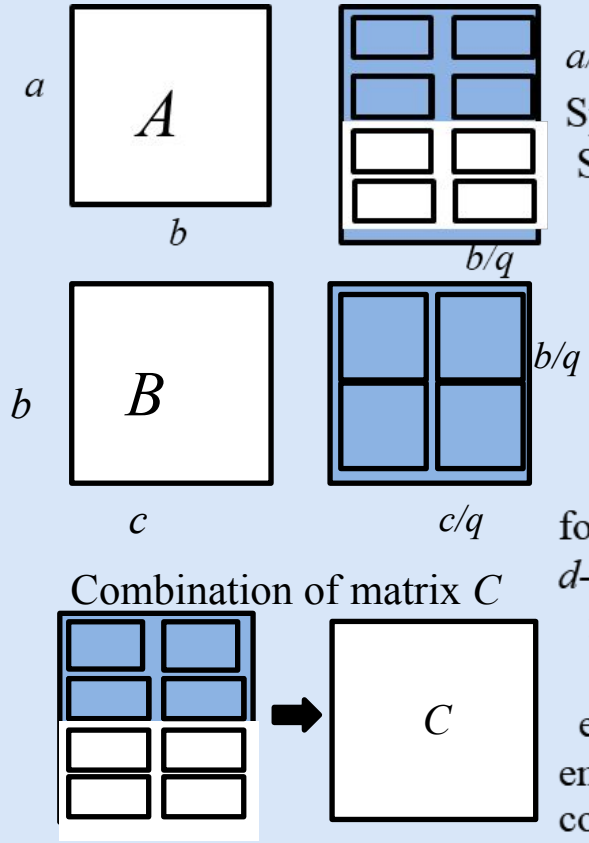


Split matrix A into $q \times q$ parts
Split matrix B into $q \times q$ parts
Broadcast:
A_{il} along the i direction
B_{lj} along the j direction

for i,j in {0,..., q-1} do
  for t in {0,..., q-1} do
    $C_{ij} = C_{ij} + A_{it} * B_{tj}$
  end for
end for
combine all C_{ij} accordingly to C

2.5-D Tensor Parallelism

A:[a,b] B:[b,c] Processors:[q,q,d]

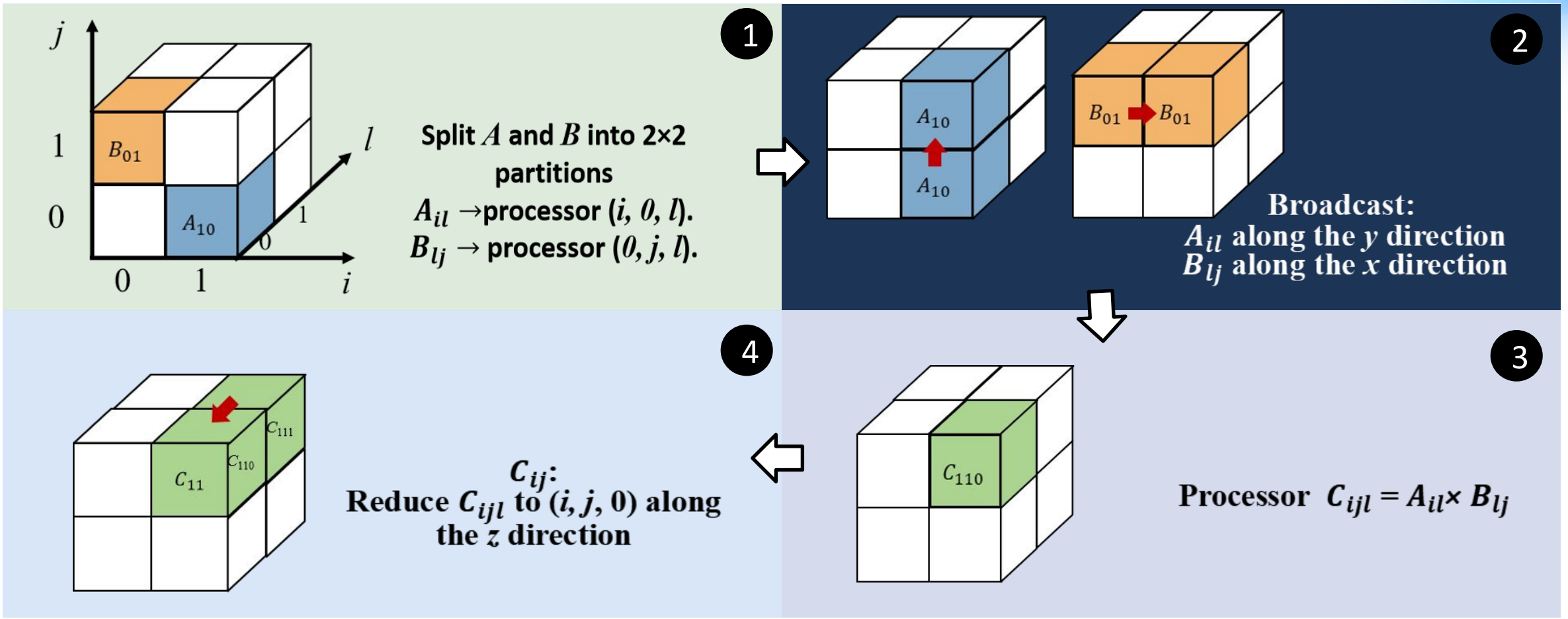


Split matrix A into $qd \times q$ parts
Split matrix B into $q \times q$ parts
Broadcast:
A_{itl} along the t direction
B_{tlj} along the t direction

for i,j in {0,..., q-1} do,k in {0,..., d-1} do
  for t in {0,..., q-1} do
    $C_{ijk} = C_{ijk} + A_{itk} * B_{tjk}$
  end for
end for
combine all C_{ij} accordingly to C

3-D Tensor Parallelism

- 3-D matrix multiplication example: $C = AB$ on a $2 \times 2 \times 2$ processors



- Advantage: Smaller communication cost. In this example, only 3 communications are required, and each communication is only carried out on $P^{1/3}$ processes.

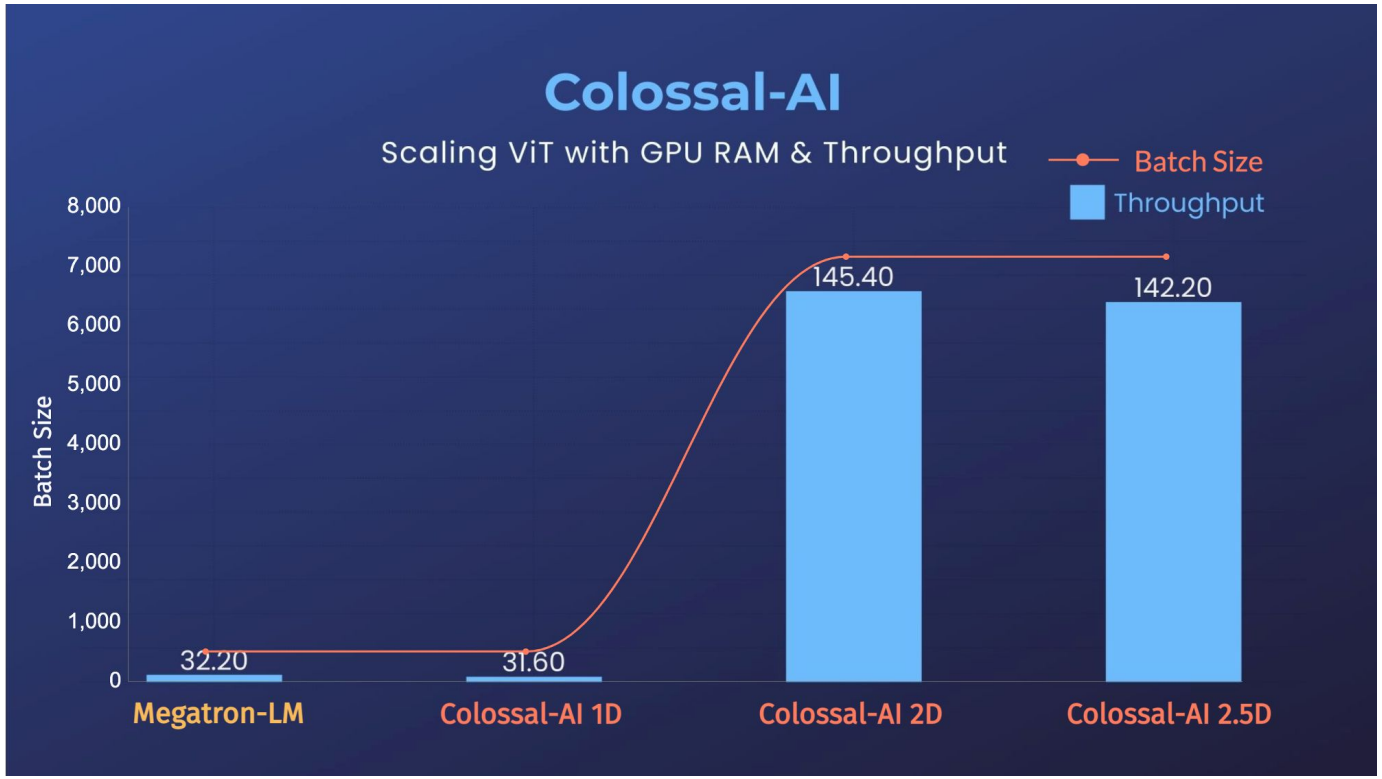
Tensor Parallelism

Efficiency:

	Computation	Memory (parameters)	Memory (activations)	Communication (bandwidth)	Communication (latency)
1 D	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O(1)$	$O(1)$	$O(P)$
2 D	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{\sqrt{P}}\right)$	$O(\sqrt{P})$
2.5 D	$O\left(\frac{1}{P}\right)$	$O\left(\frac{d}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{\sqrt{dP}}\right)$	$O\left(\sqrt{\frac{P}{d^3}}\right)$
3 D	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P^{\frac{2}{3}}}\right)$	$O\left(P^{\frac{1}{3}}\right)$

- P: number of processors

ViT

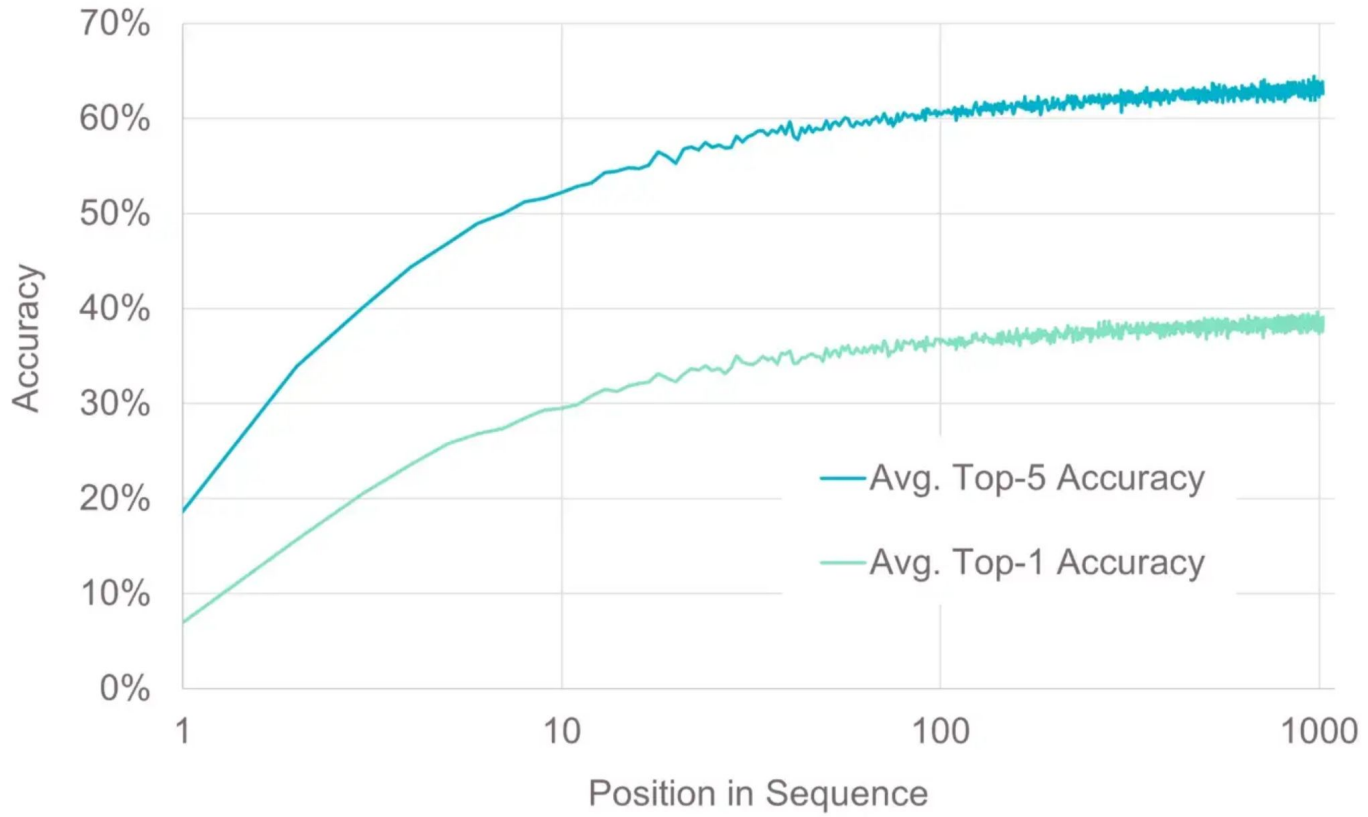


- the full advanced sample is available on the project homepage https://github.com/hpcaitech/ColossalAI

- Up to 14x larger batch size, and 5x faster training for Tensor Parallelism = 64.

Long Sequences Matter

- Long sequence is common: document, image, amino acids in protein, etc.
- Pre-trained GPT-2 on the next token prediction task



- A larger context helps to better predict which token is about to come next.

https://medium.com/@cerebras/context-is-everything-why-maximum-sequence-length-matters-for-ai-fa1f4c81009f

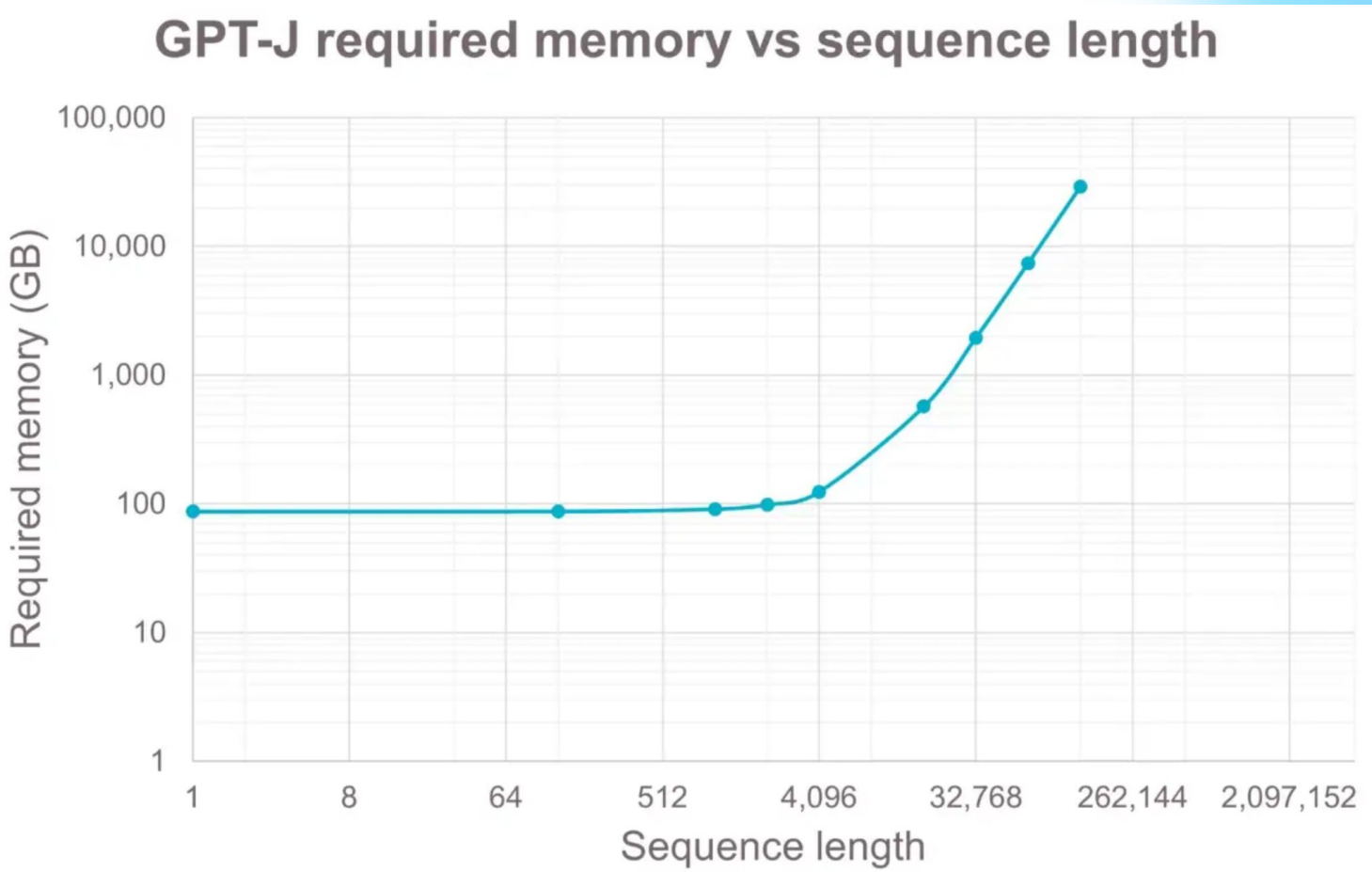
Memory Bottleneck of Sequence Length

- Model
  - Weights
  - Gradients
- Optimizer
  - States

fixed

- Input Data
  - Activation

vary



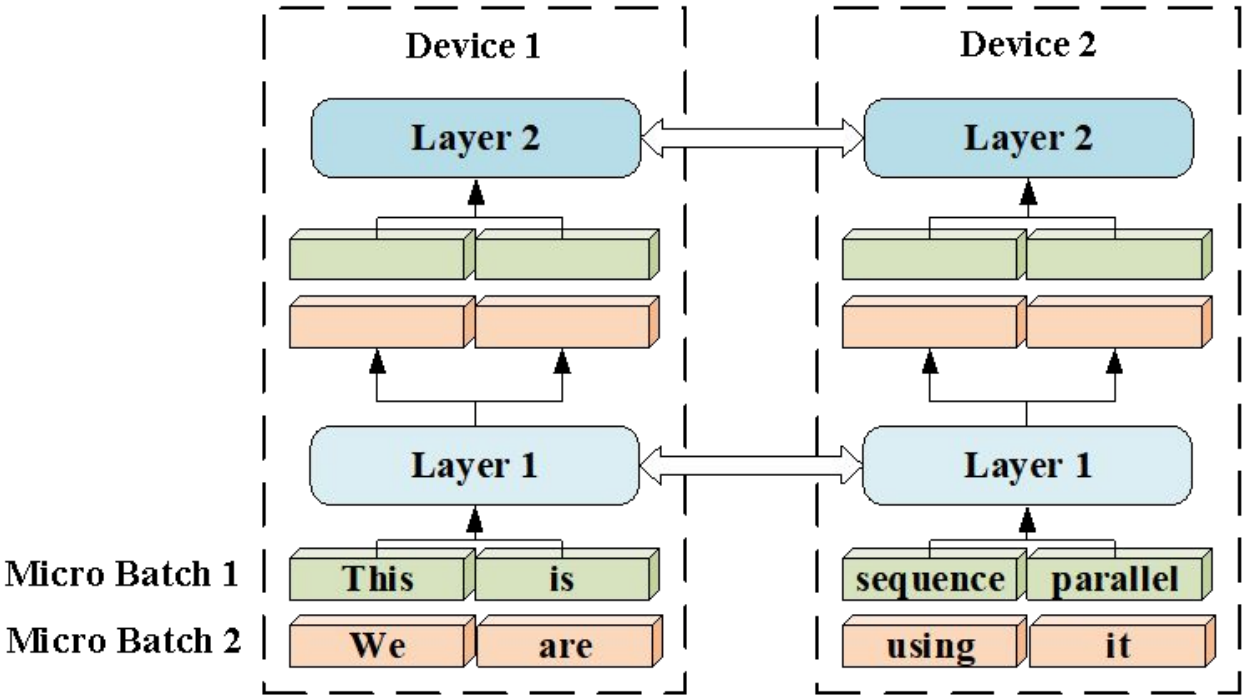
- Transformer (Attention) has quadratic complexity at memory.
- When data dimension is large, it can become the memory bottleneck.

https://medium.com/@cerebras/context-is-everything-why-maximum-sequence-length-matters-for-ai-fa1f4c81009f

Sequence Parallelism

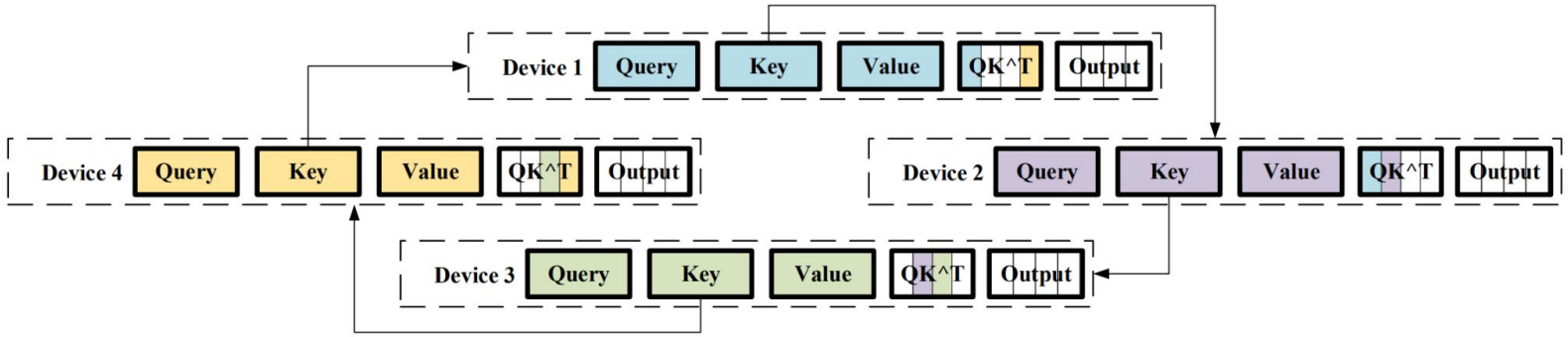
Why Sequence Parallelism?

- Limitation: Transformer based models are required to hold the whole sequence on single device during training, and distribute the long sequence on multiple devices.

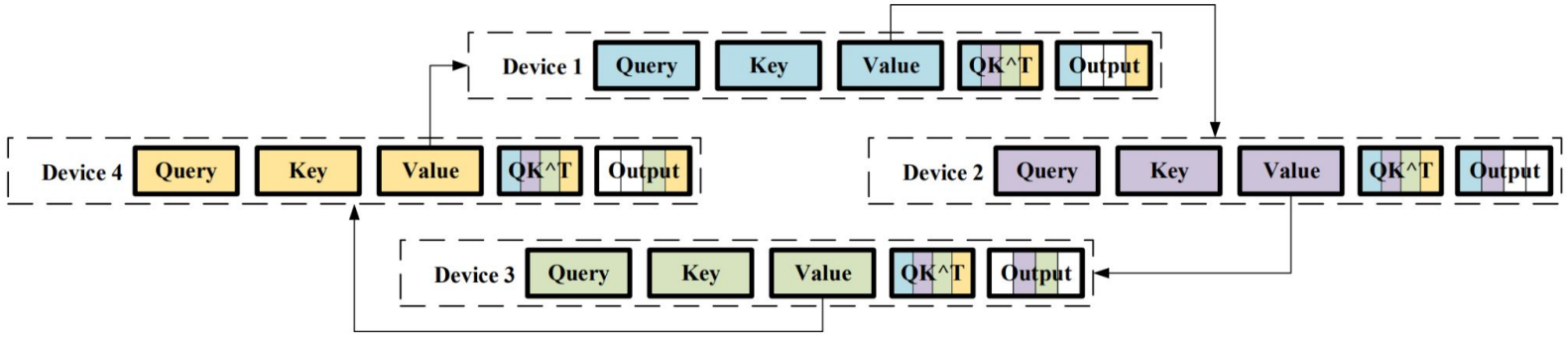


- Parallelize in the sequence dimension -> reduce memory consumption by input data and activation
- Model weights are replicated across devices

Ring Self-Attention



(a) Transmitting key embeddings among devices to calculate attention scores

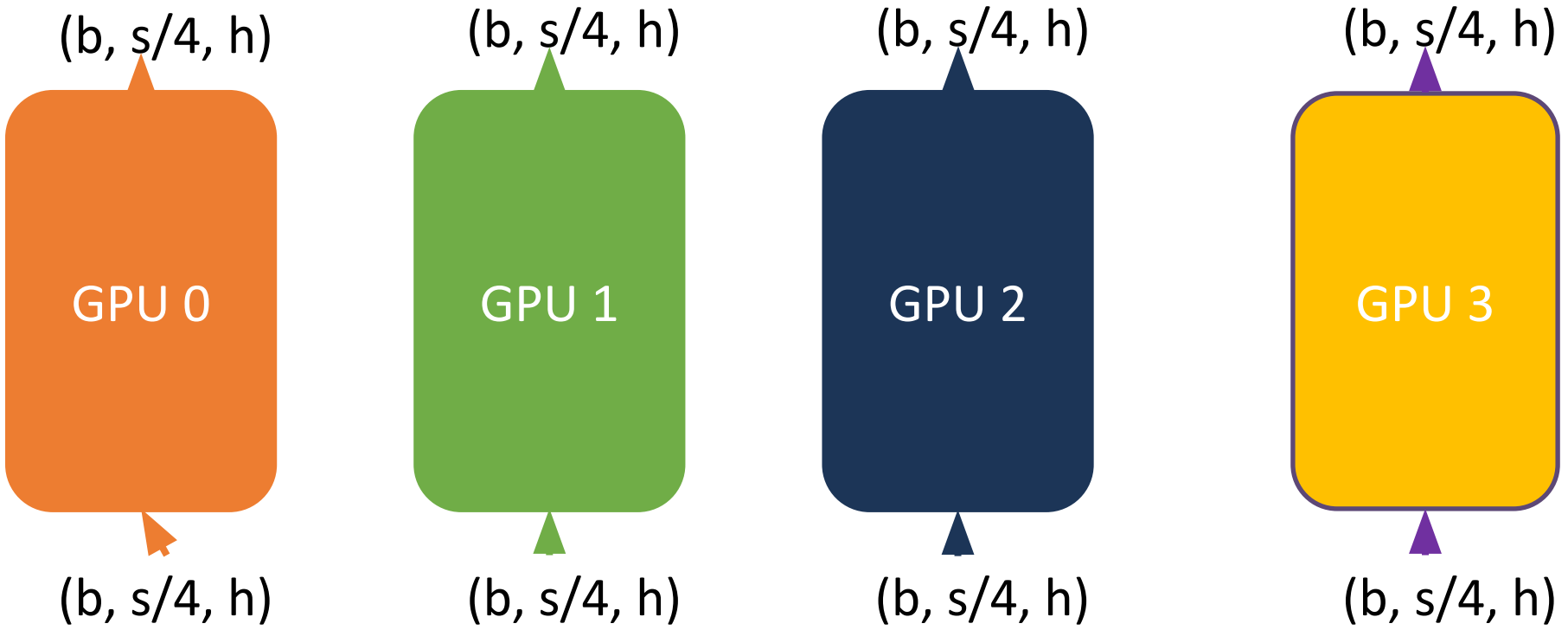


(b) Transmitting value embeddings among devices to calculate the output of attention layers

- Inspired by Ring All-reduce
- Communicate query, key and value embeddings for self-attention calculation

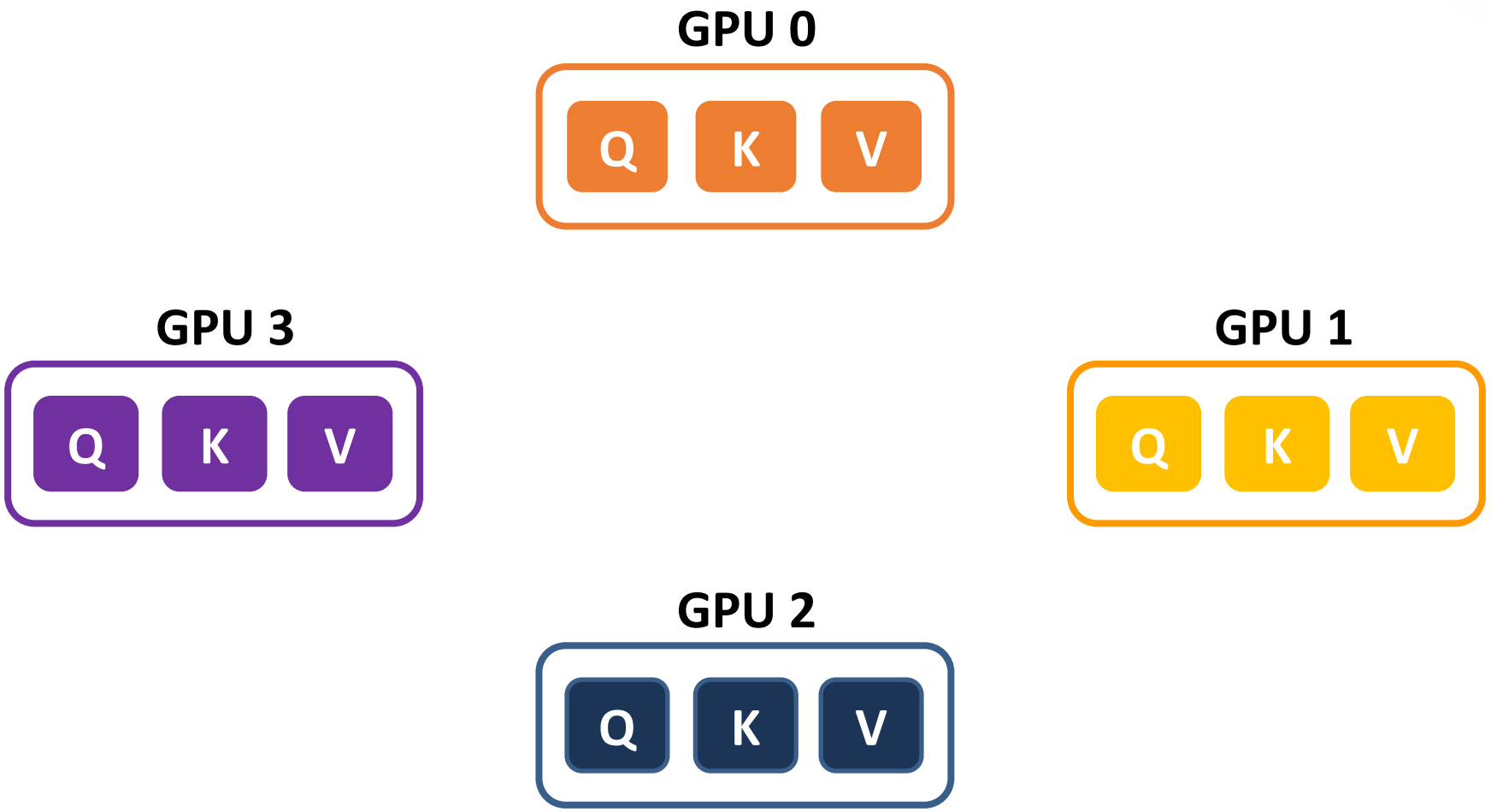
Sequence Parallelism

Ring Self-Attention



Sequence Parallelism

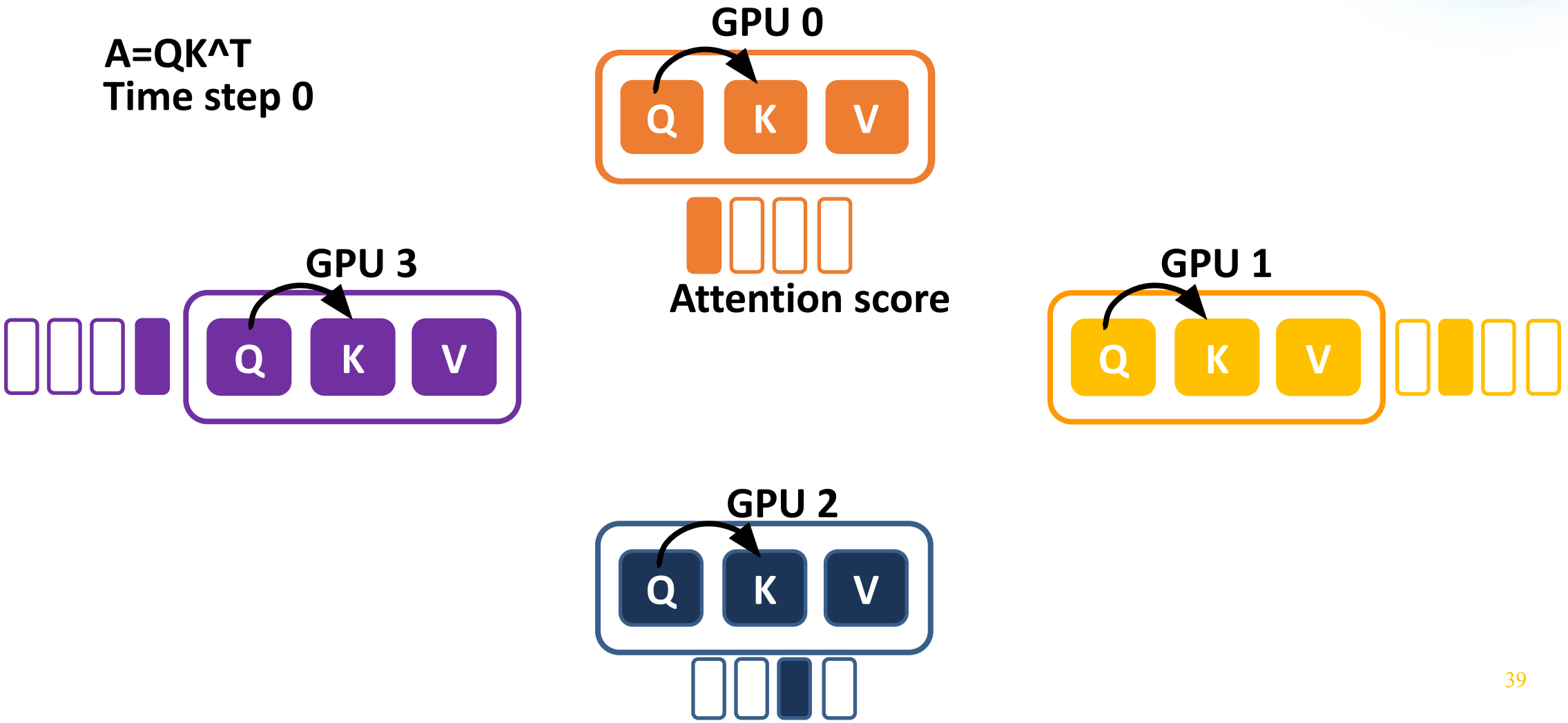
Ring Self-Attention



Sequence Parallelism

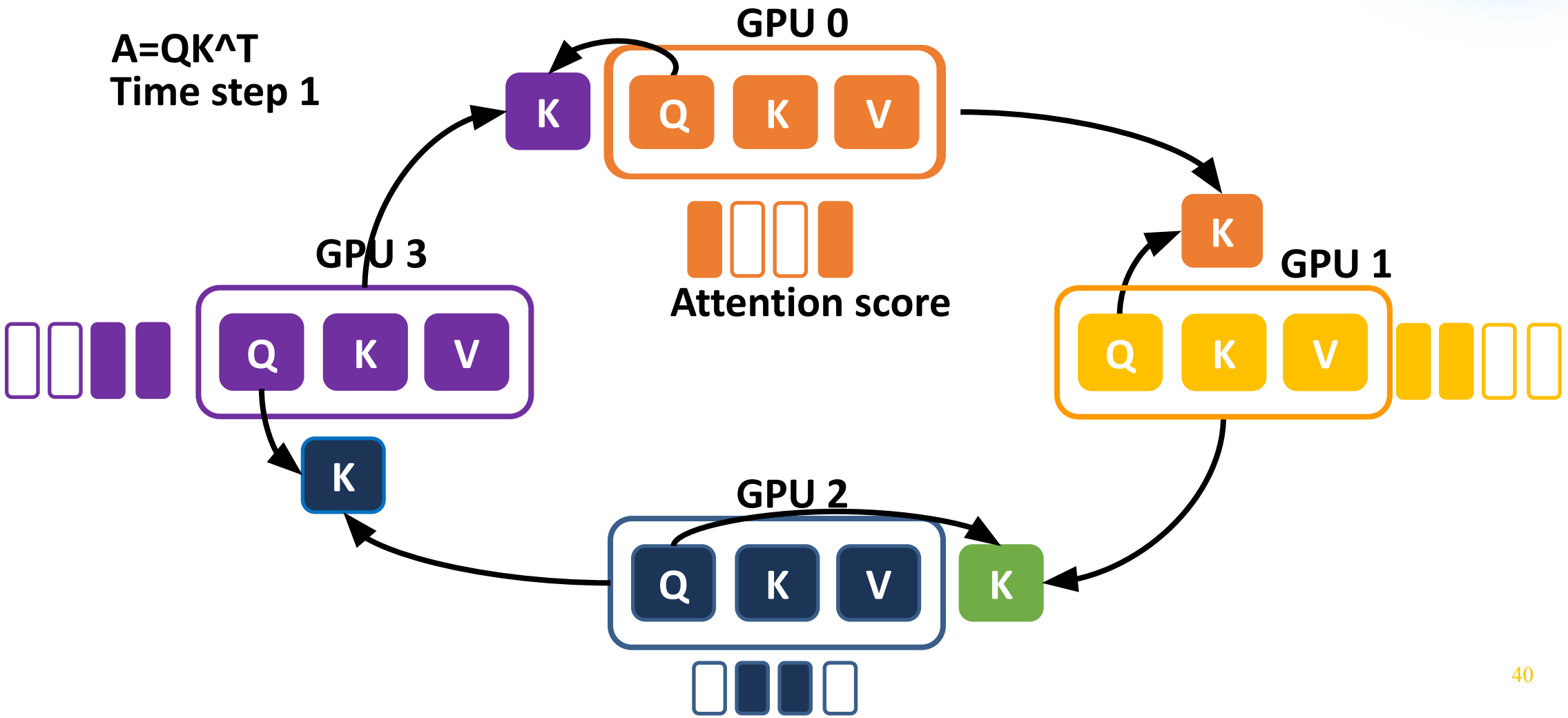
Ring Self-Attention

A=QK^T
Time step 0



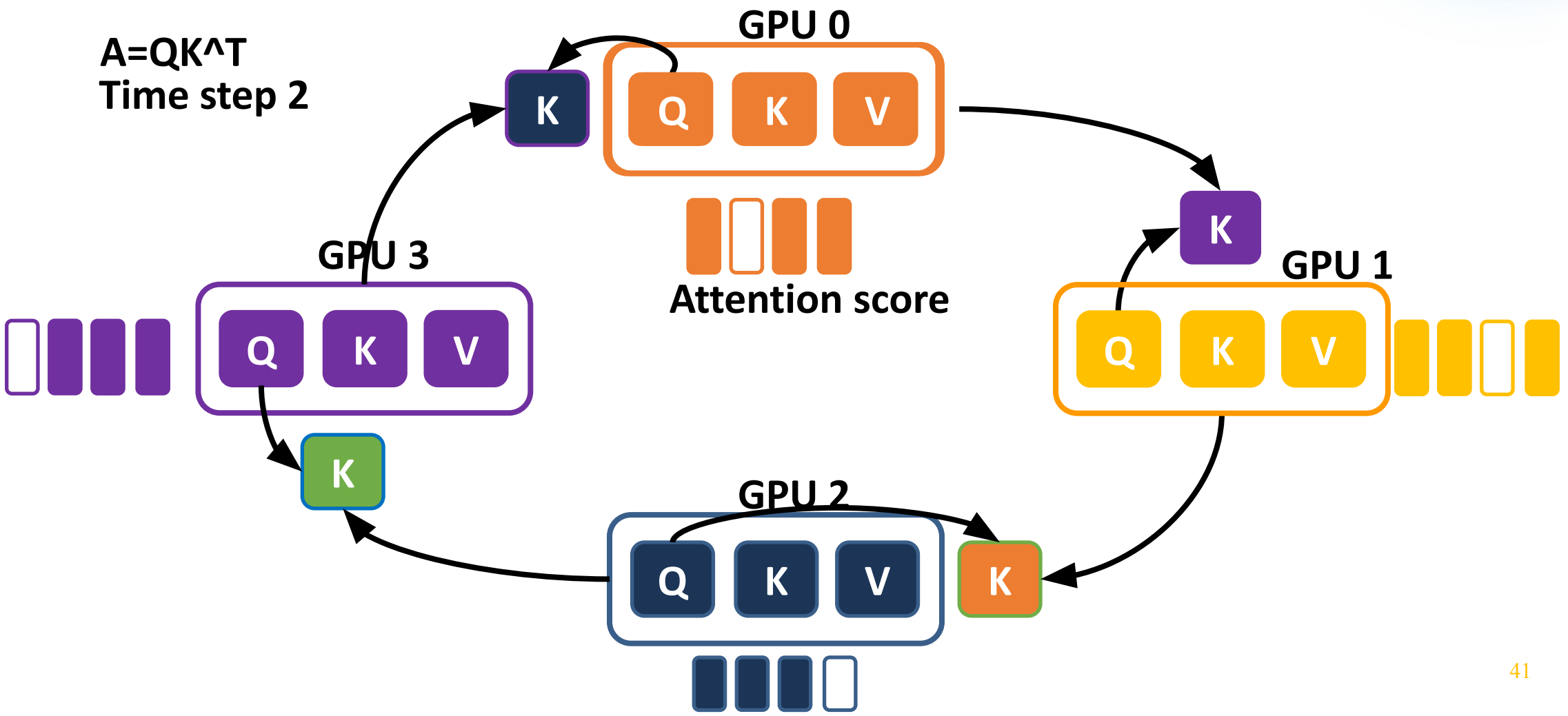
Sequence Parallelism

Ring Self-Attention



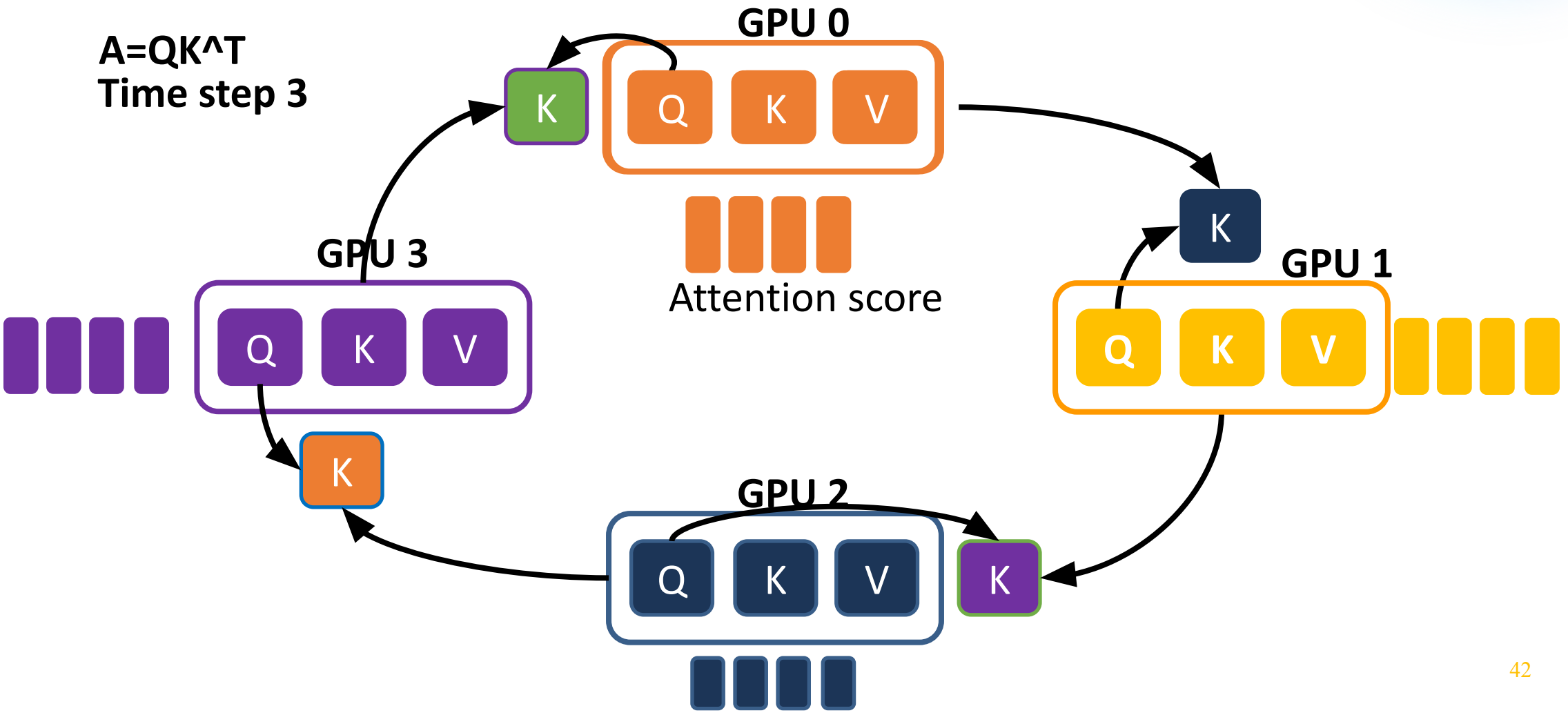
Sequence Parallelism

Ring Self-Attention

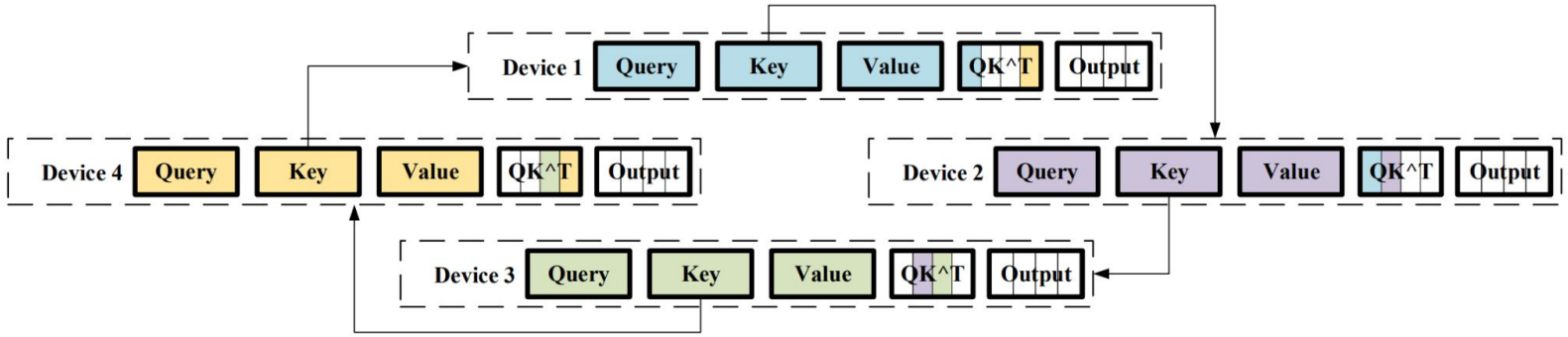


Sequence Parallelism

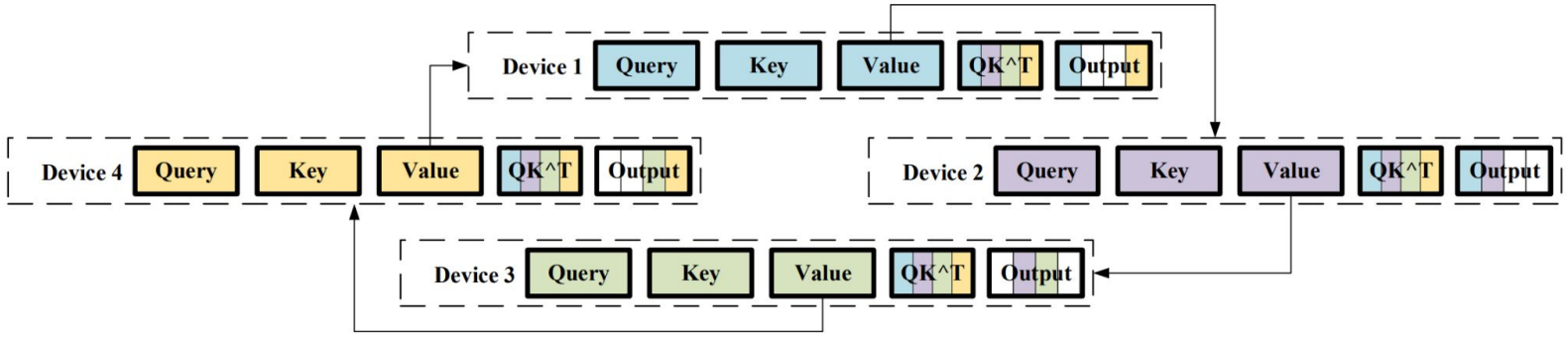
Ring Self-Attention



Ring Self-Attention



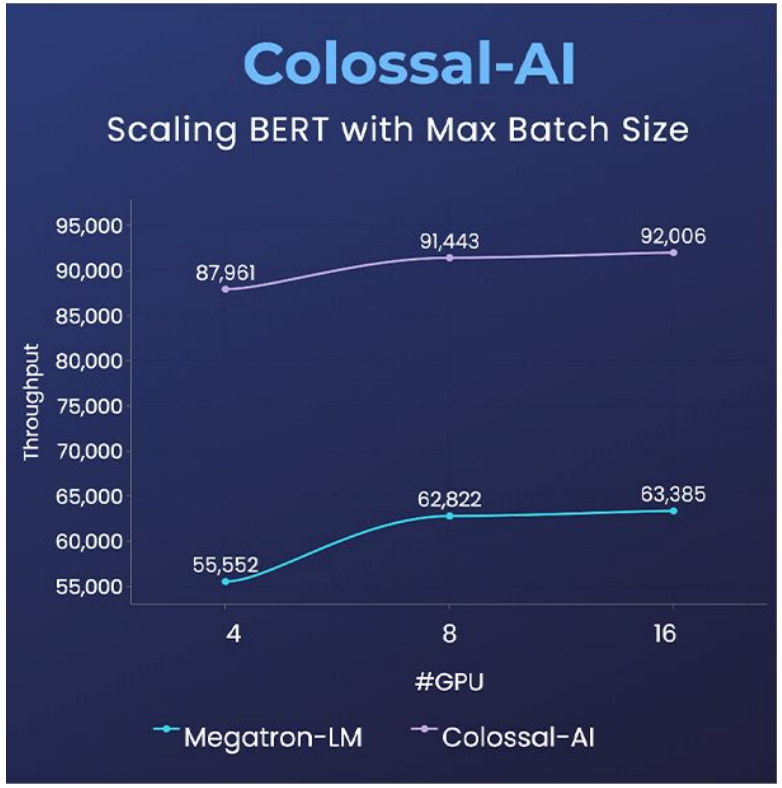
(a) Transmitting key embeddings among devices to calculate attention scores

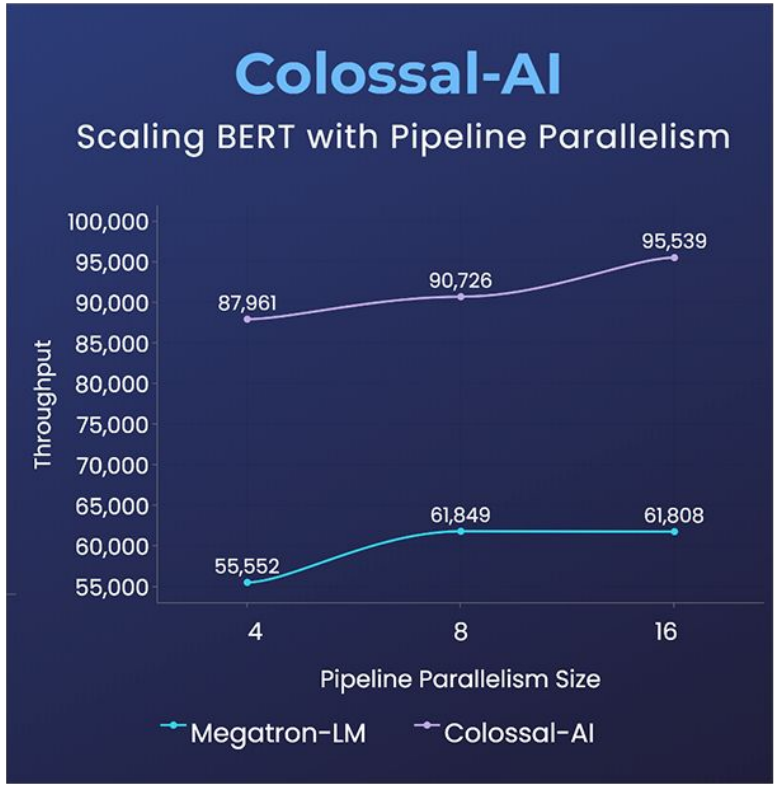


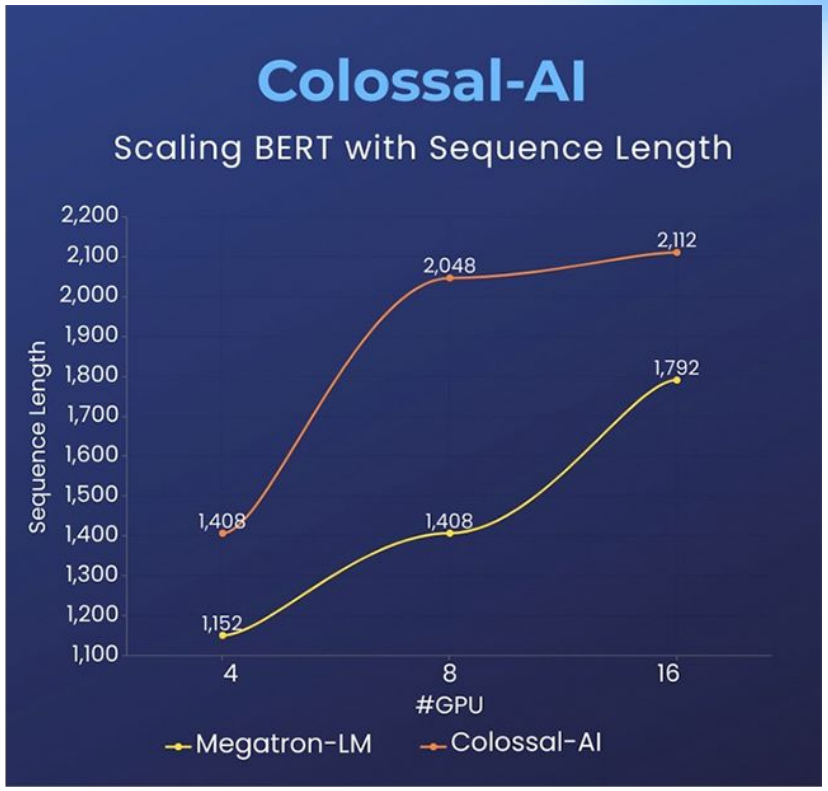
(b) Transmitting value embeddings among devices to calculate the output of attention layers

- Inspired by Ring All-reduce
- Communicate query, key and value embeddings for self-attention calculation

BERT



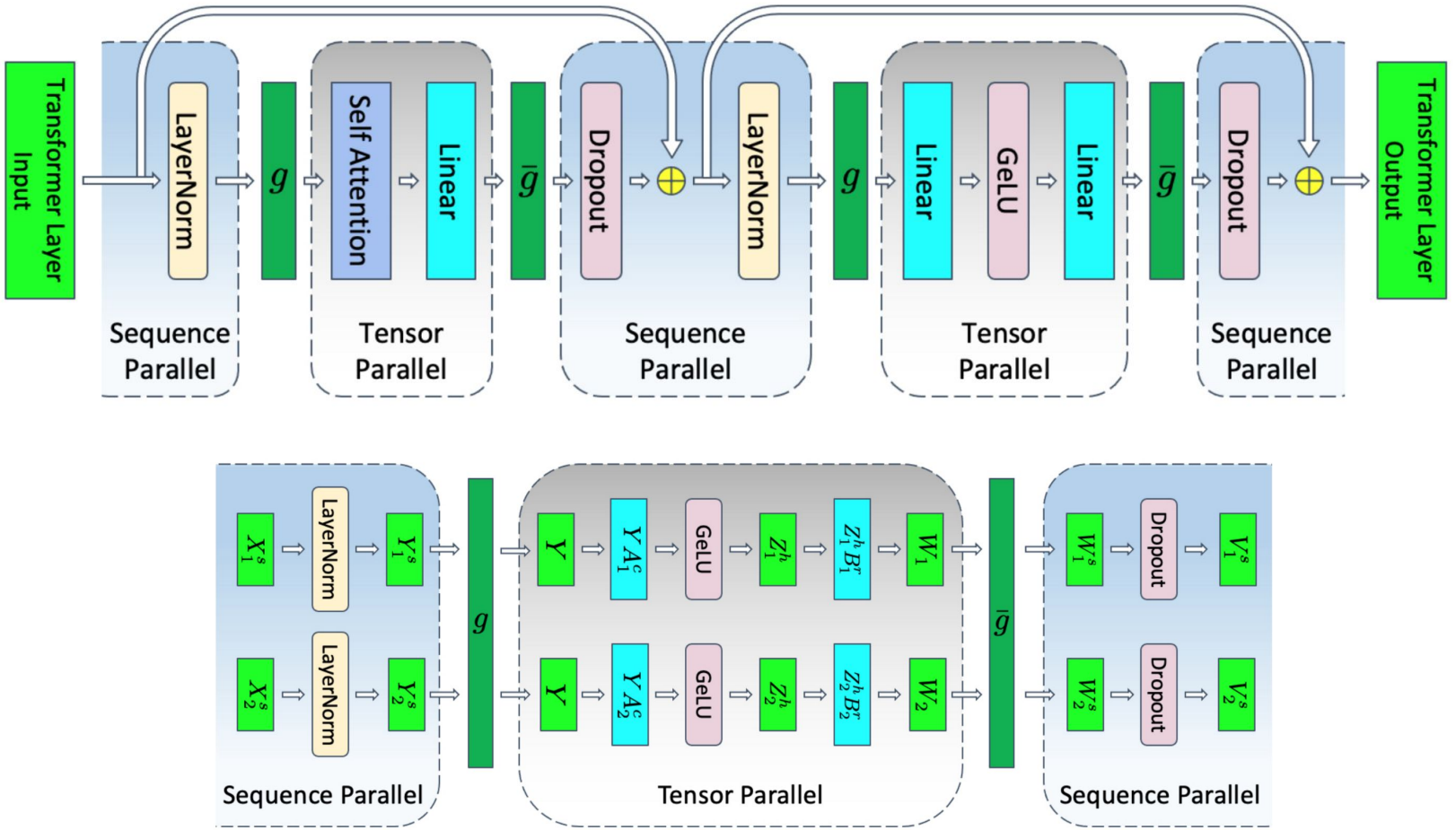




- The full advanced sample is available on the project homepage https://github.com/hpcaitech/ColossalAI

- Up to 2x faster training, or 50% longer sequence length

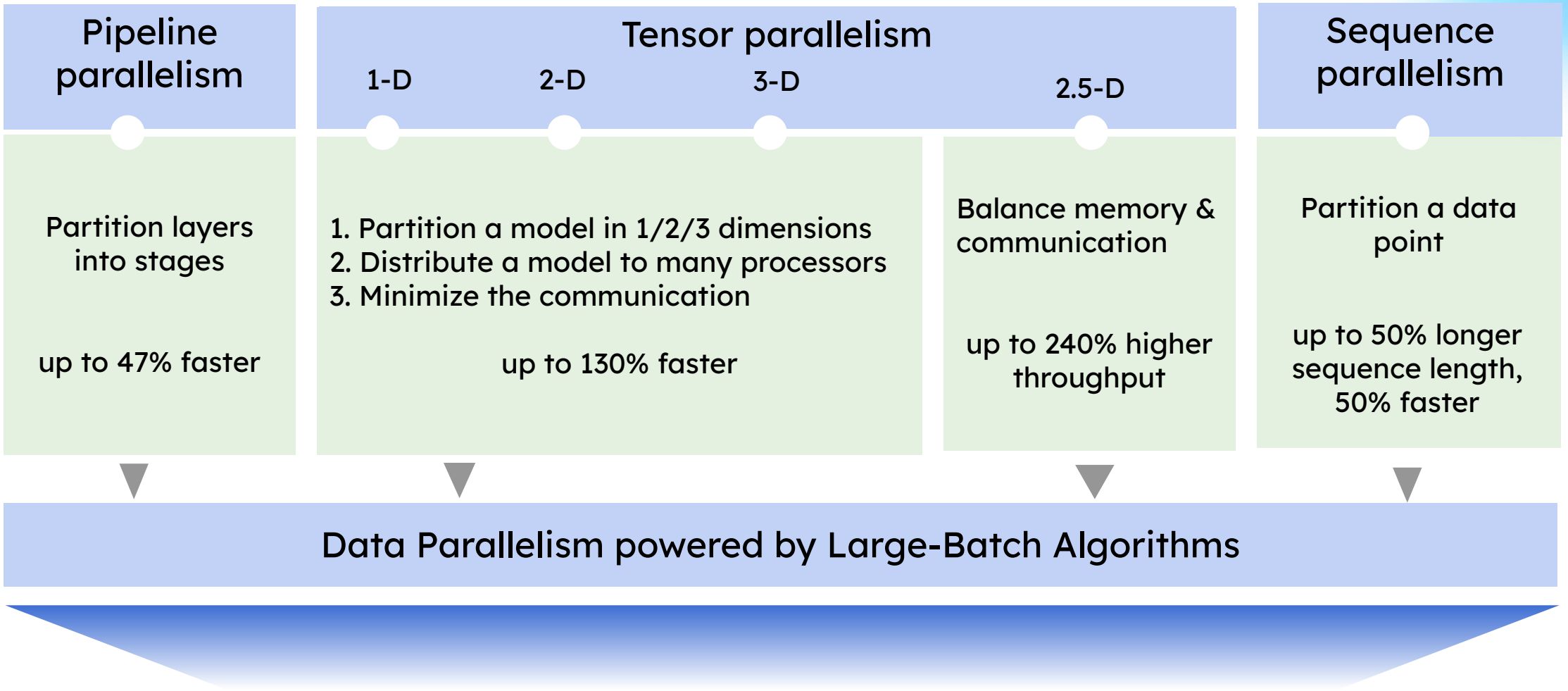
Sequence Parallelism in model layer



Transformer layer with tensor and sequence parallelism.

Korthikanti V, Casper J, Lym S, et al. Reducing Activation Recomputation in Large Transformer Models[J]. arXiv preprint arXiv:2205.05198, 2022.

Overview of N-Dim Parallelism System



All in N-Dim Parallelism System
from Colossal-AI

Outline

1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Hands-on videos

- Components Overview and Environment Setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism

Join Colossal-AI Slack!
Get this slides and Q&A

- The hands-on recordings and sample code are available on the project homepage https://github.com/hpcaitech/ColossalAI

We'll be back soon

https://github.com/hpcaitech/ColossalAI

Join Colossal-AI Slack!
Get this slides and Q&A

Outline

1. Introduction of Big Model Era

2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

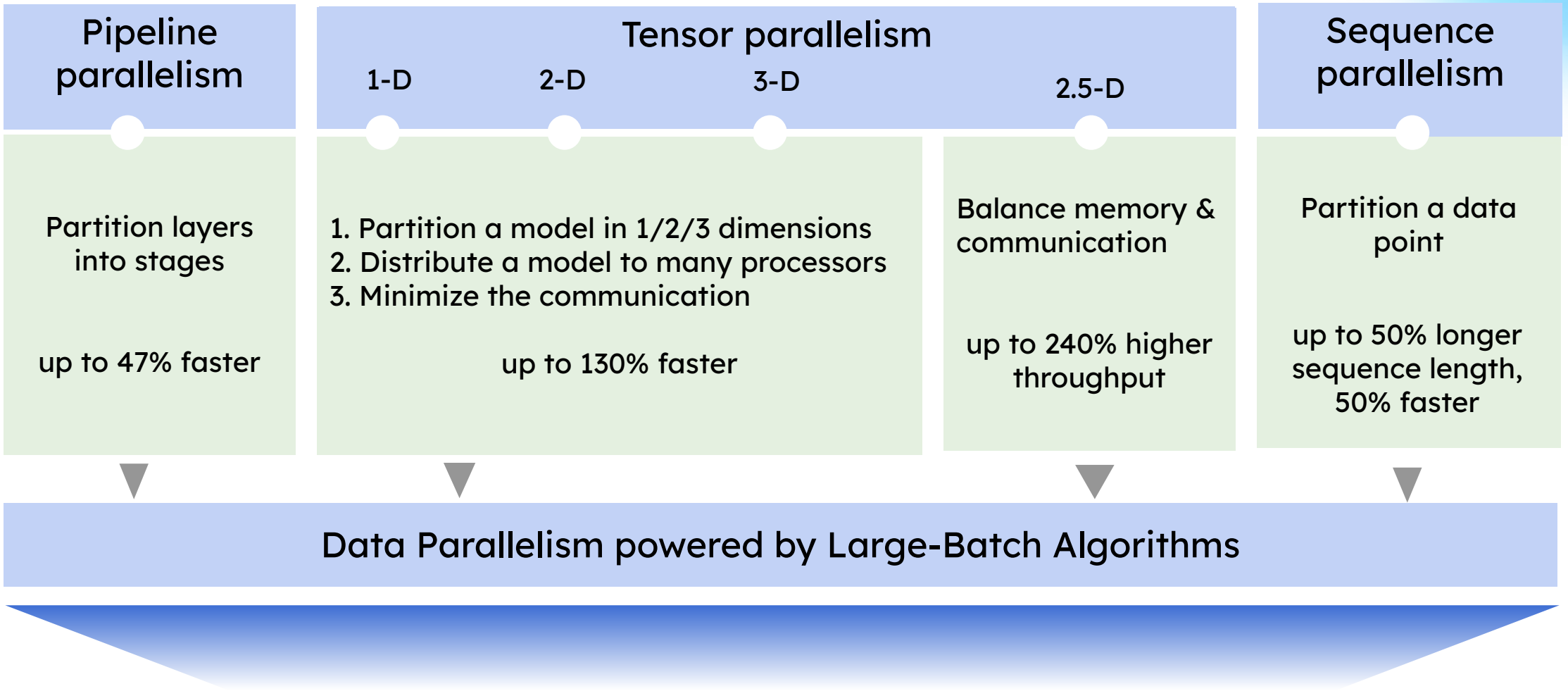
3. Automatic Parallelism System

4. Efficient Memory System

5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

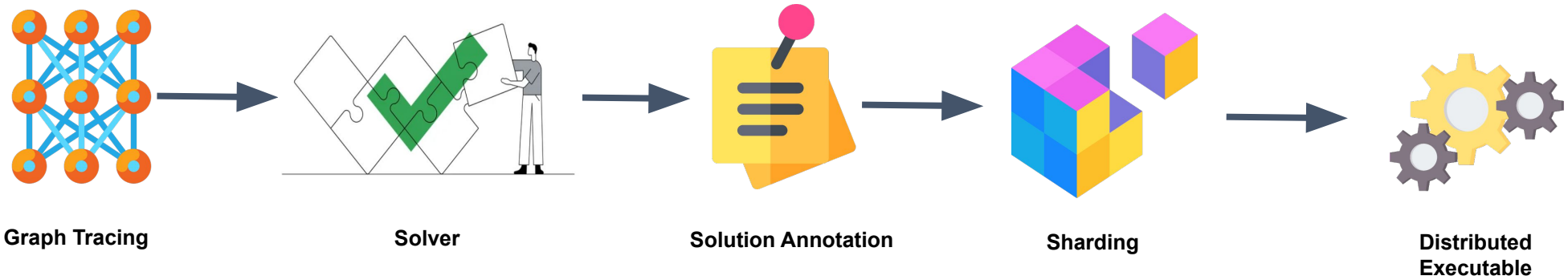
Overview of N-Dim Parallelism System



All in N-Dim Parallelism System from Colossal-AI

Auto-Parallelism

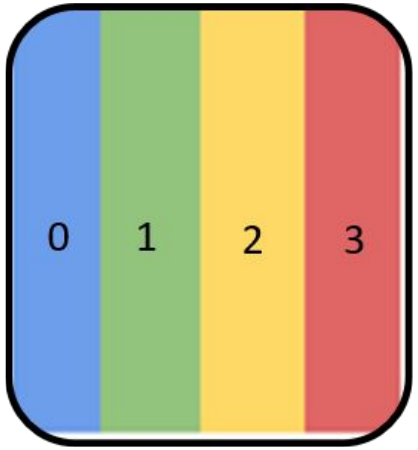
- First automatically search for parallel strategies on PyTorch (static graph analysis)
  - Maximize compute efficiency
  - Minimize communication time
- Minimum code change required —— One Line of Code
- Seamlessly integrates with Hugging Face and Timm




```
# wrap the model using auto_engine
model, optimizer = auto_engine(model, optimizer, cluster_info)
# normal training loop
...
```

Infrastructure

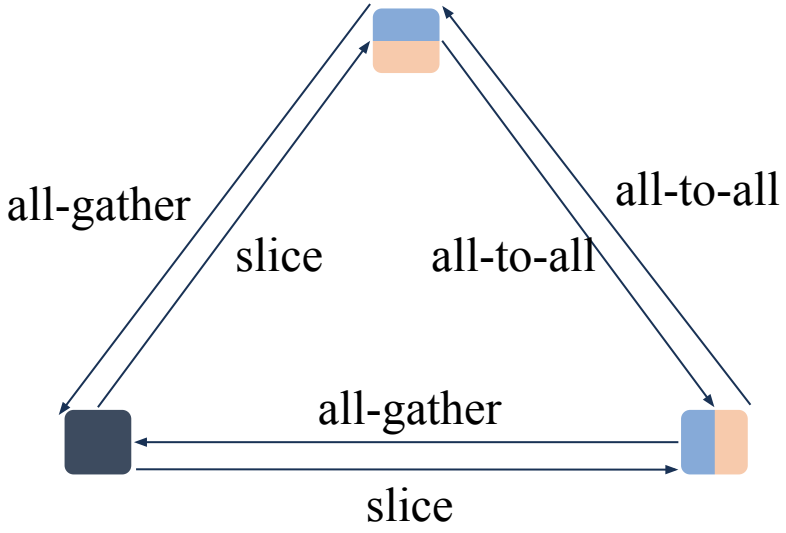
- Distributed Tensor Management
  - Device mesh
    &
    Sharding Spec



An example of sharding spec RS01

  - Tensor Layout Management

	extensibility	efficiency
Enumerating coversion	X	O
Dim by dim conversion	O	X
heuristic conversion (our solution)	O	O



Enumerating conversion for 1D mesh

Infrastructure

- Distributed Tensor Management
  - Tensor Layout Management

Source/target sharding spec pairs	One step transform	Best sharding spec	Transform path
S01RR, RS01R	S0RR, S0RS1, S0S1R	S0RR	S01RR-> S0RR
S0RR, RS01R	RRR, S0S1R, S0RS1, RS0R, RRS0	RS0R	S01RR-> S0RR -> RS0R
RS0R, RS01R	RRR, RS01R, S1S0R, RS0S1, S0RR, RRS0	RS01R	S01RR-> S0RR -> RS0R -> RS01R

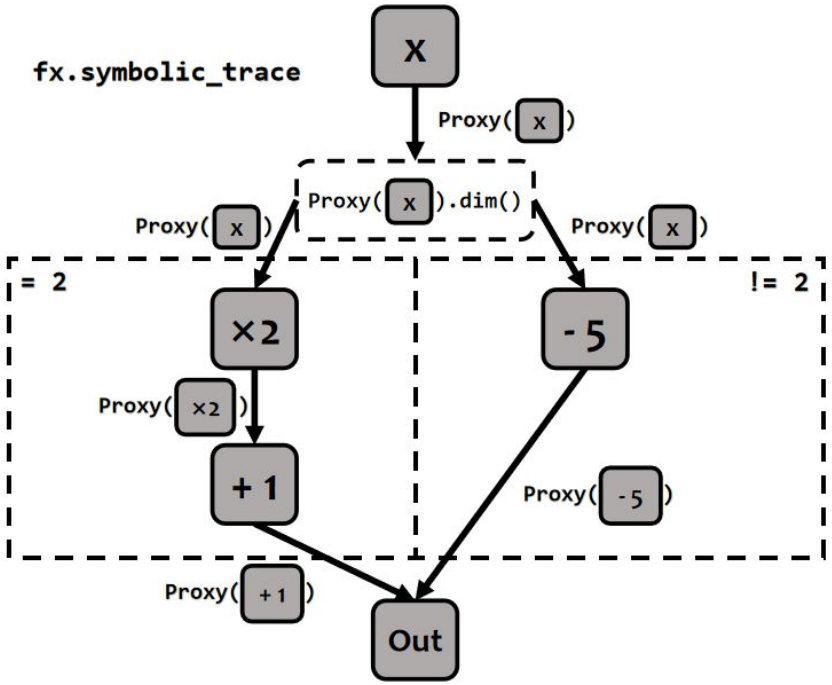
conversion path for S01RR to RS01R

Algorithm 1 shape consistency transform path

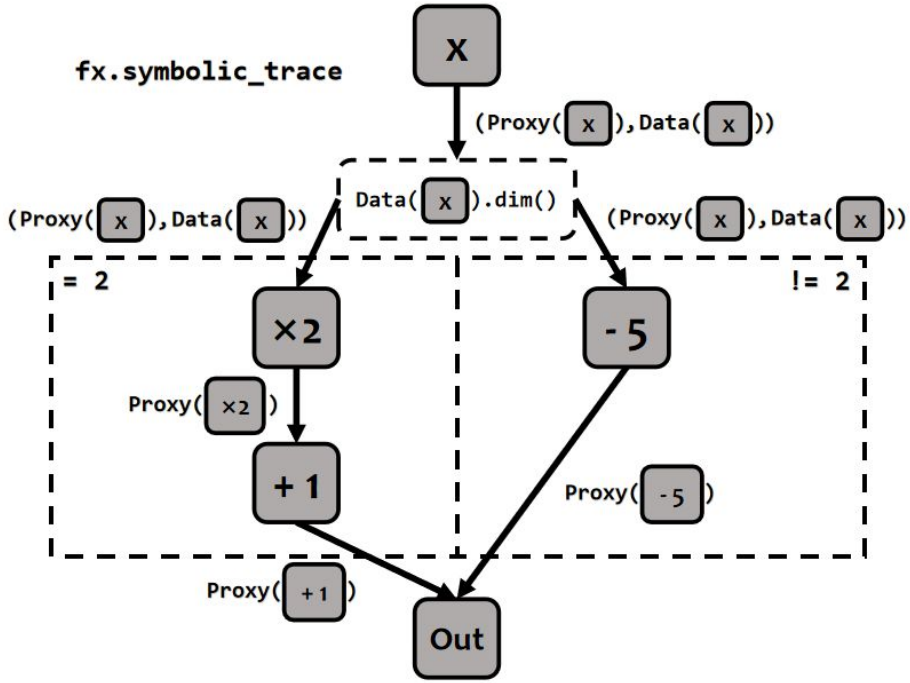
```
1:  Input: source_spec, target_spec
2:  Output: transform_path
3:  procedure ShapeConsistencyTransform
4:      total_step ← 0
5:      transform_path(total_step) ← source_spec
6:      repeat
7:          min_diff ← ∞
8:          valid_transforms ← Transform(source_spec)
9:          for all sharding_spec ∈ valid_transforms do
10:             diff ← Heuristic(sharding_spec, target_spe
11:             if diff < min_diff then
12:                 source_spec ← sharding_spec
13:                 min_diff ← diff
14:             end if
15:         end for
16:         total_step+ = 1
17:         transform_path(total_step) ← source_spec
18:     until min_diff == 0
19:     return transform_path
20: end procedure
```

Infrastructure

- Symbolic Analysis System
  - Symbolic Tracing



torch fx original tracer



ColoTracer

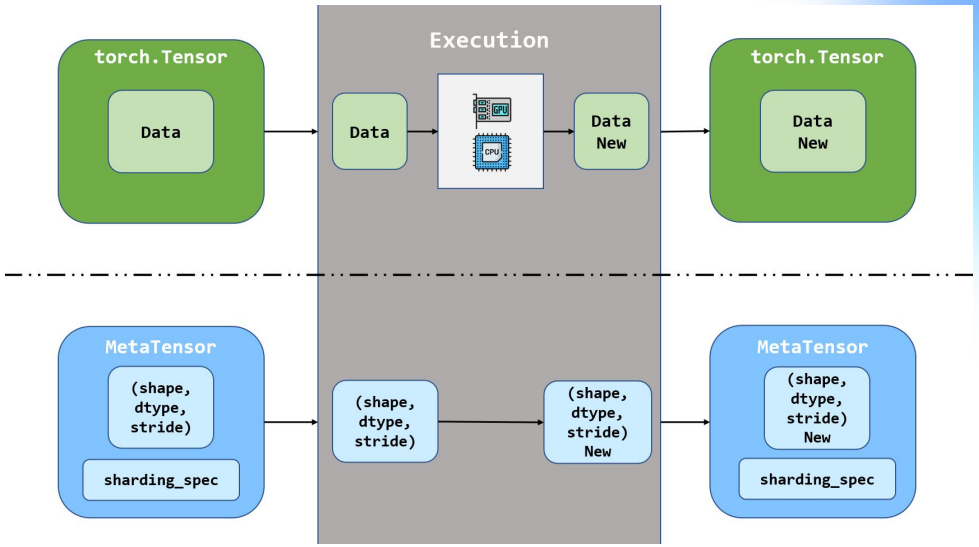
Infrastructure

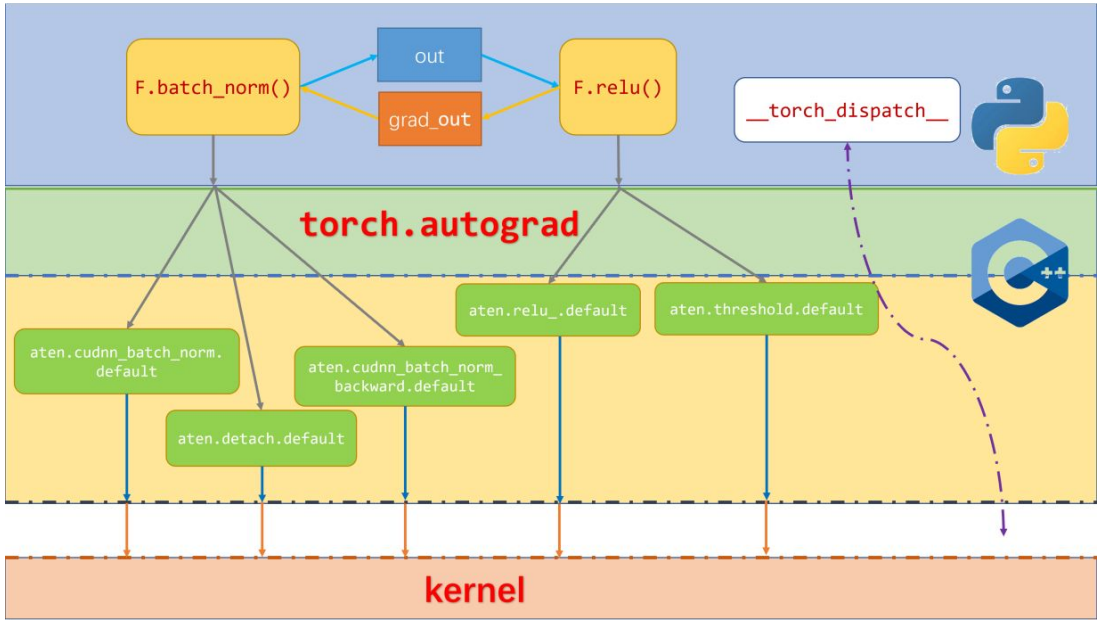
- Symbolic Analysis System
  - Static Profiler
    1. Symbolic execution
    2. Aten level analysis
    3. Complete computation graph(forward and backward)

OpCounter	Op Level	Backward FLOPs	is Meta
pytorch-OpCounter	nn	✗	✗
deepspeed（Github - DeepSpeed/profiler.py at master · microsoft/DeepSpeed ）	aten	✗	✗
Ideal FLOP Counter （The “Ideal” PyTorch FLOP Counter (with __torch_dispatch__) ）	aten	✓	✗
colossalai	aten	✓	✓

Infrastructure

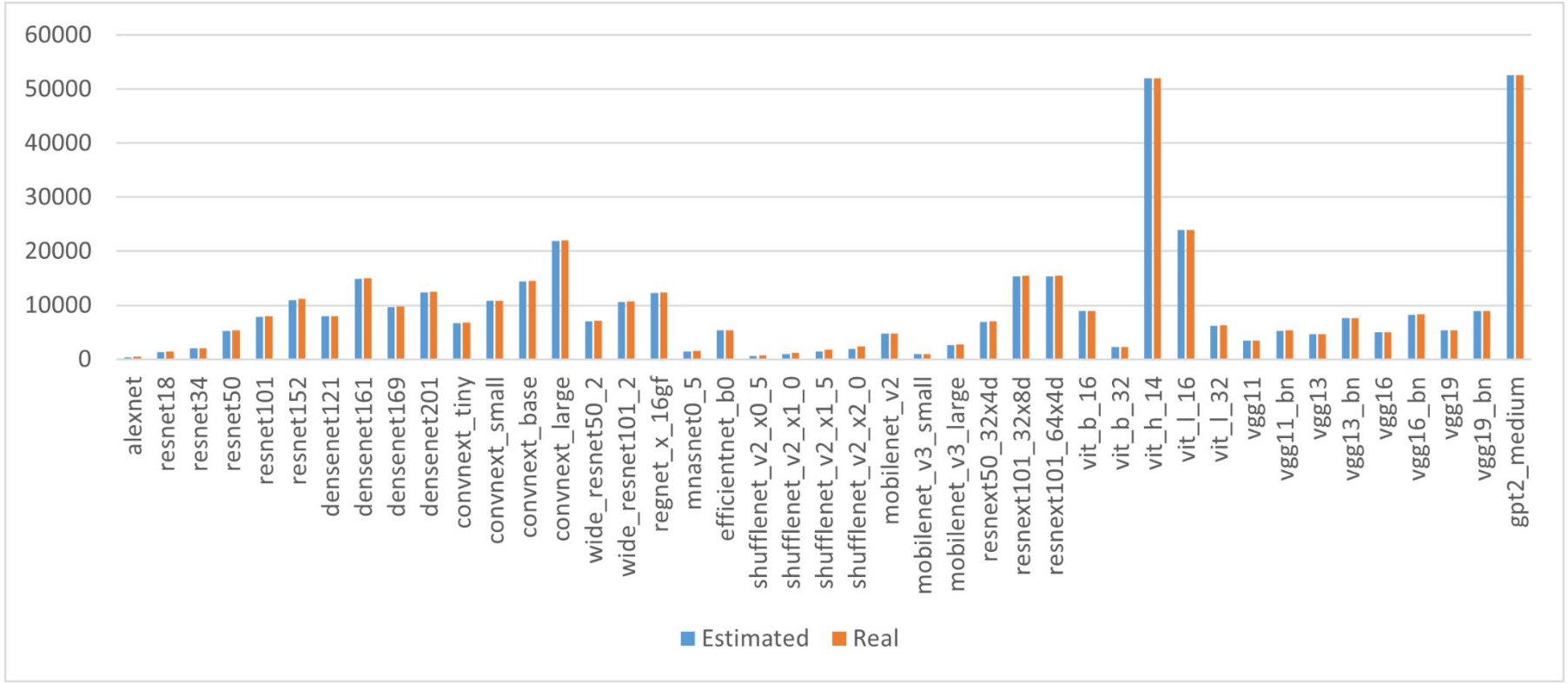
- Symbolic Analysis System
  - Meta Profiler
    1. Symbolic execution
    2. Complete computation graph
    3. Aten level analysis





Infrastructure

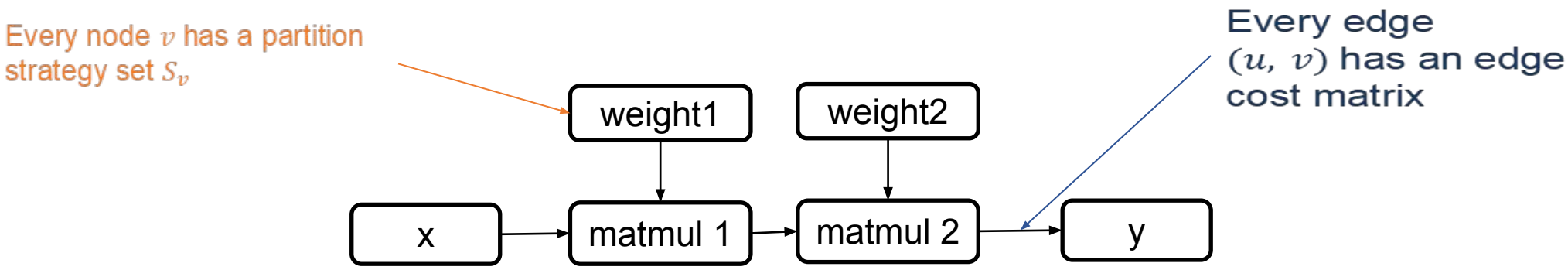
- Symbolic Analysis System
  - evaluation



SPMD Solver

SPMD Style: Partition every operator evenly across all GPUs in a device mesh. The SPMD style covers data parallelism, operator parallelism, ZeRO optimizer, and their combinations.

Goal: Find a partition strategy for every operator.

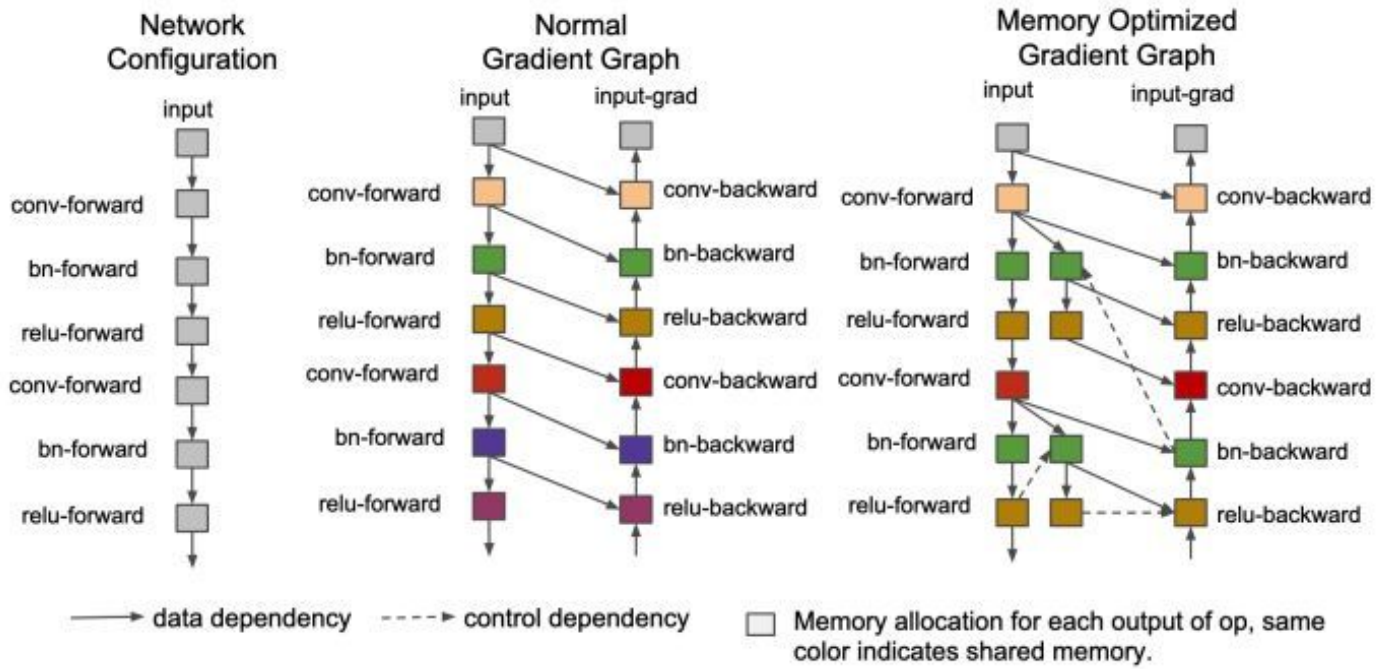


Total time cost = node-cost (computation cost+communication cost inside node) + edge-cost (communication cost among nodes)

Automatic Activation Checkpoint

Activation Checkpoint: During training, drop some of the intermediate results, and recover them from an extra forward computation when needed. This will break the memory wall of a single GPU by trading off space and time.

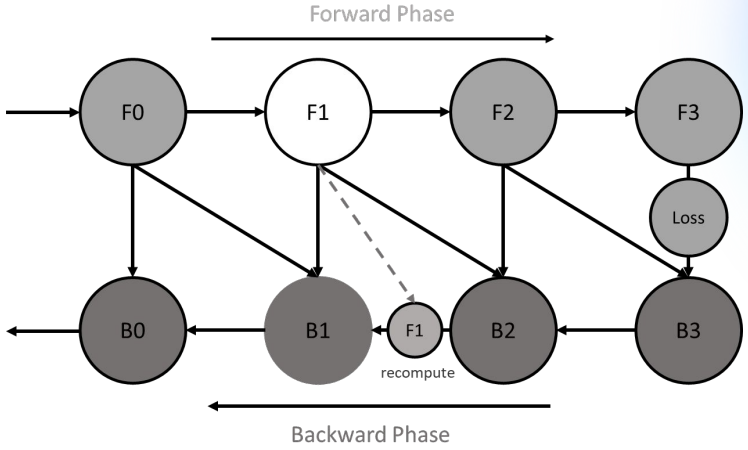
Goal: Search optimal activation checkpoint with given memory budget.



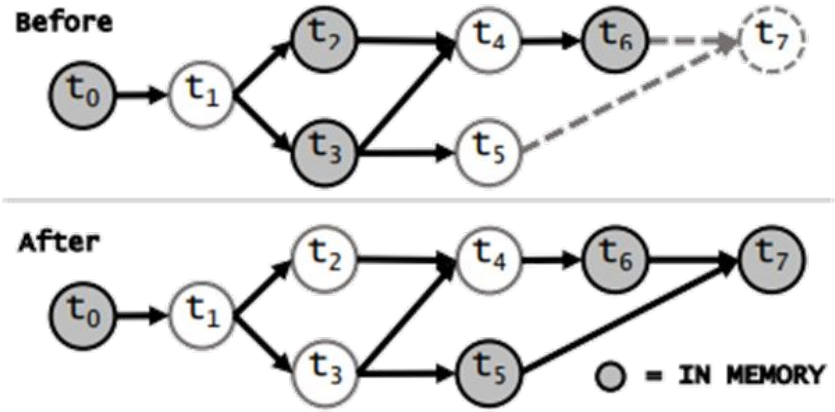
Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." arXiv preprint arXiv:1604.06174 (2016).

Existing Solutions

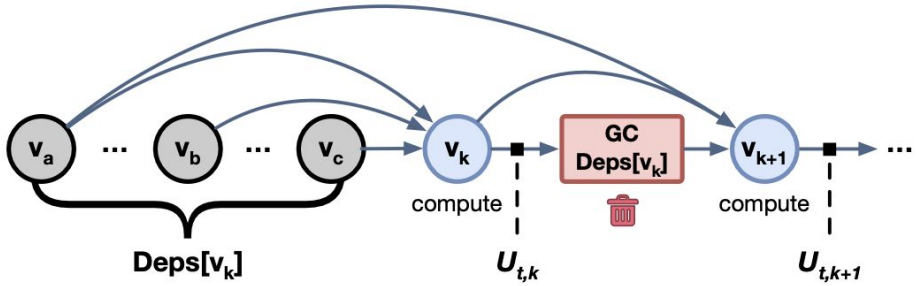
Rotor: Dynamic programming on linearized graph



Dynamic Tensor Rematerialization: Online scheduling algorithm on arbitrary graph



Checkmate: Optimal tensor rematerialization as a mixed integer programming problem



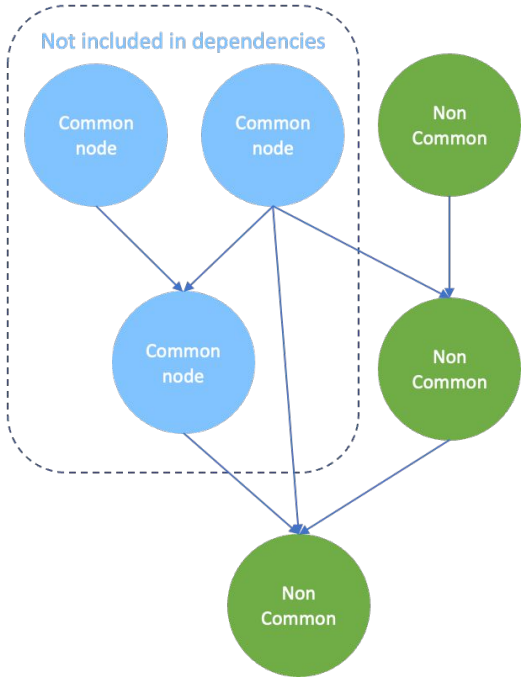
Kirisame, Marisa, et al. "Dynamic tensor rematerialization." arXiv preprint arXiv:2006.09616 (2020).

Jain, Paras, et al. "Checkmate: Breaking the memory wall with optimal tensor rematerialization." Proceedings of Machine Learning and Systems 2 (2020): 497-511.

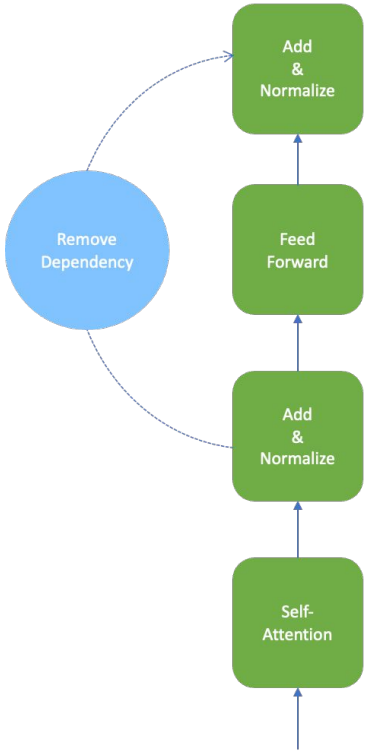
Beaumont, Olivier, Lionel Eyraud-Dubois, and Alena Shilova. "Efficient Combination of Rematerialization and Offloading for Training DNNs." Advances in Neural Information Processing Systems 34 (2021): 23844-23857.

Our Solution

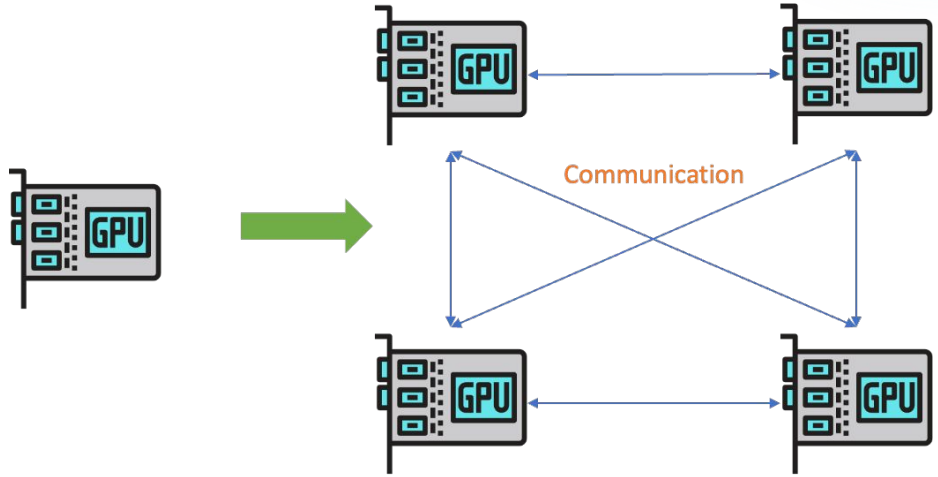
Common node propagation



Remove residual connection dependency

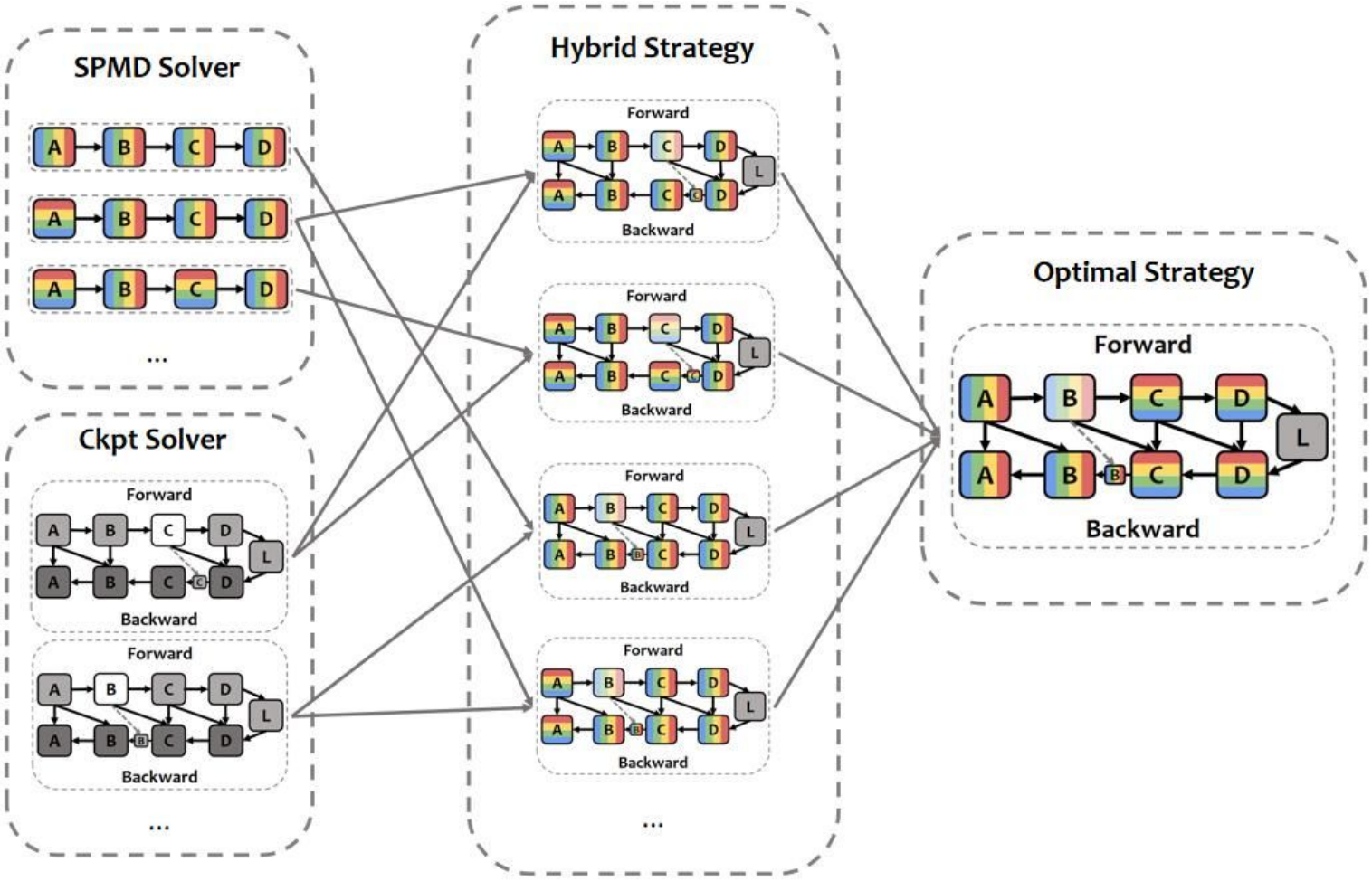


Communication-aware modeling



Larger search space
Easier to Combine with SPMD Solver

Integration



Outline

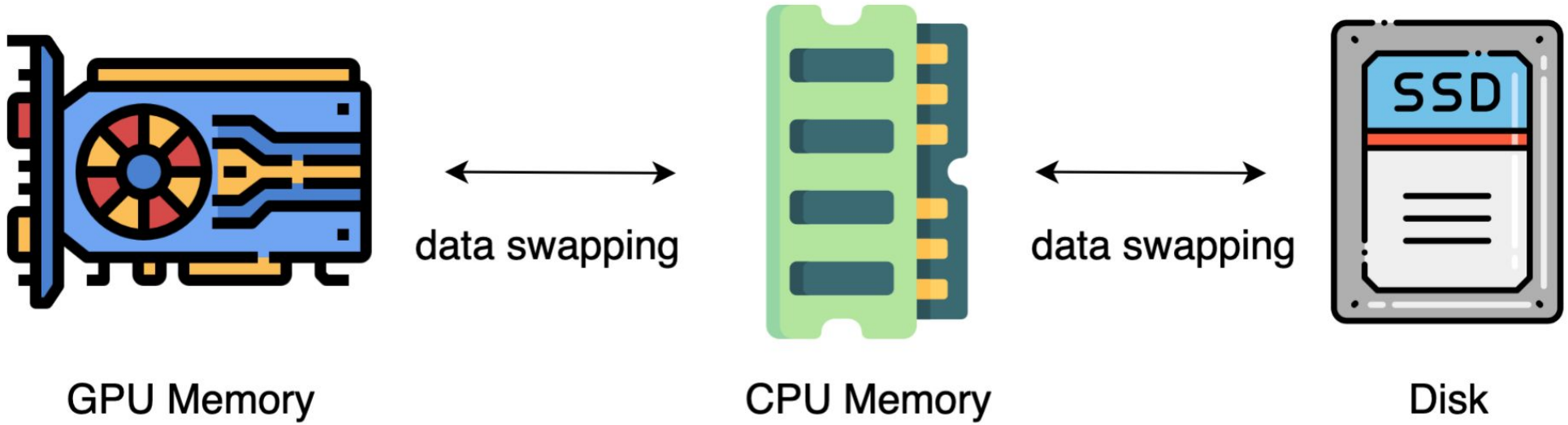
1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

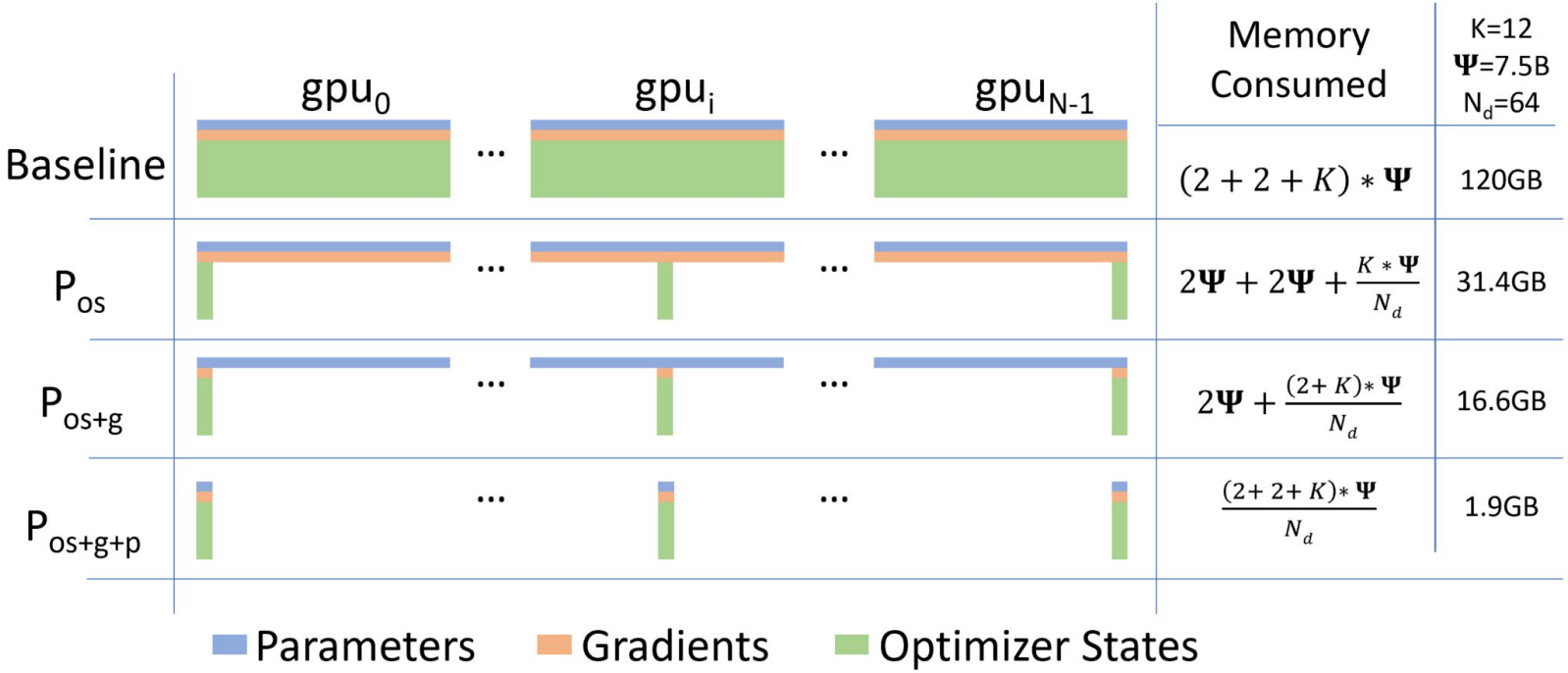
Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Data Movement on a Heterogeneous System



- Heterogeneous system illustration

ZeRO (Zero Redundancy Optimizer)

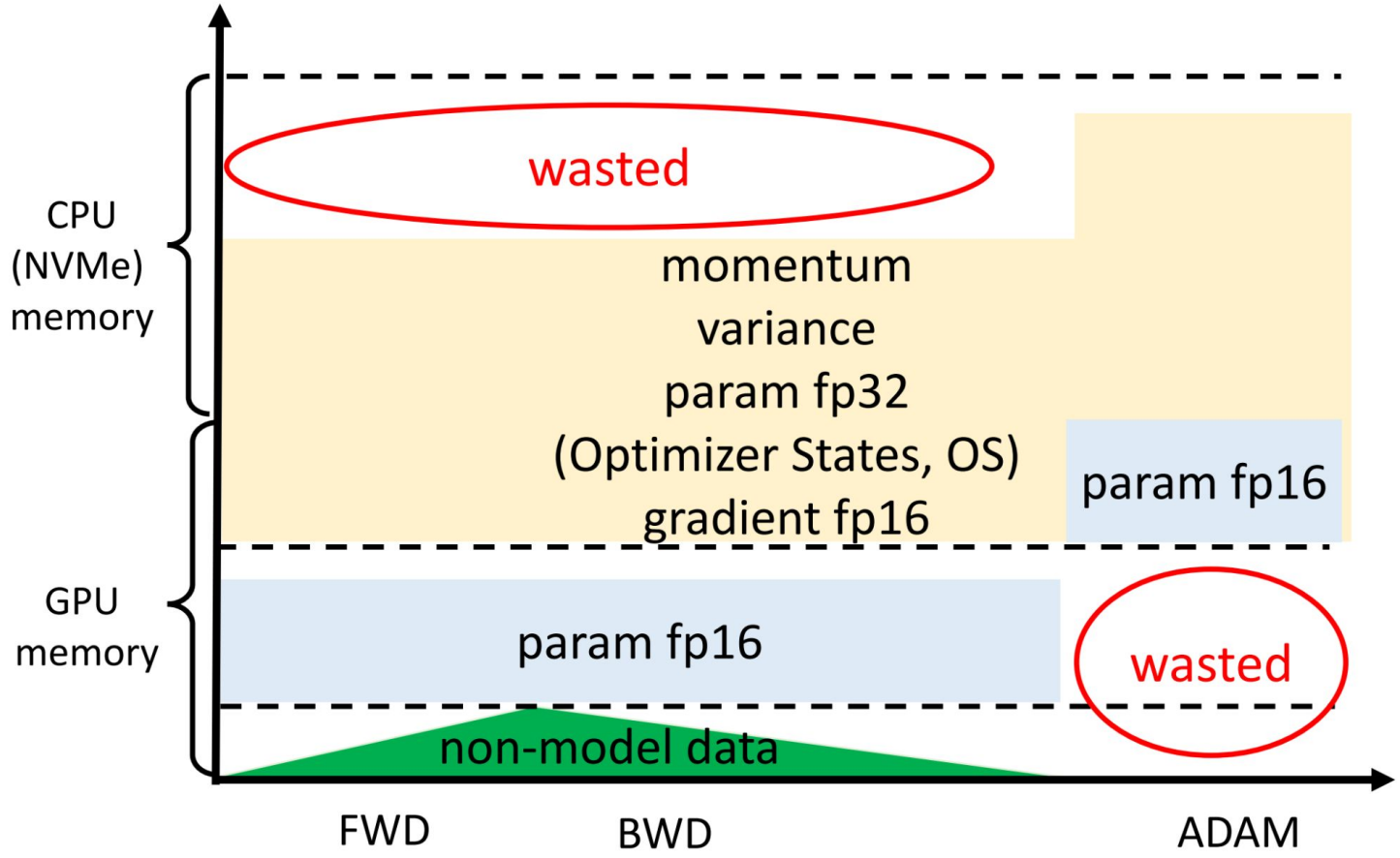


- Partition the model states (weights, gradients, and optimizer states) across available devices.
- Offload GPU memory to both CPU and NVMe memory for huge memory savings.

 Eliminate memory redundancies in data and model parallelism.

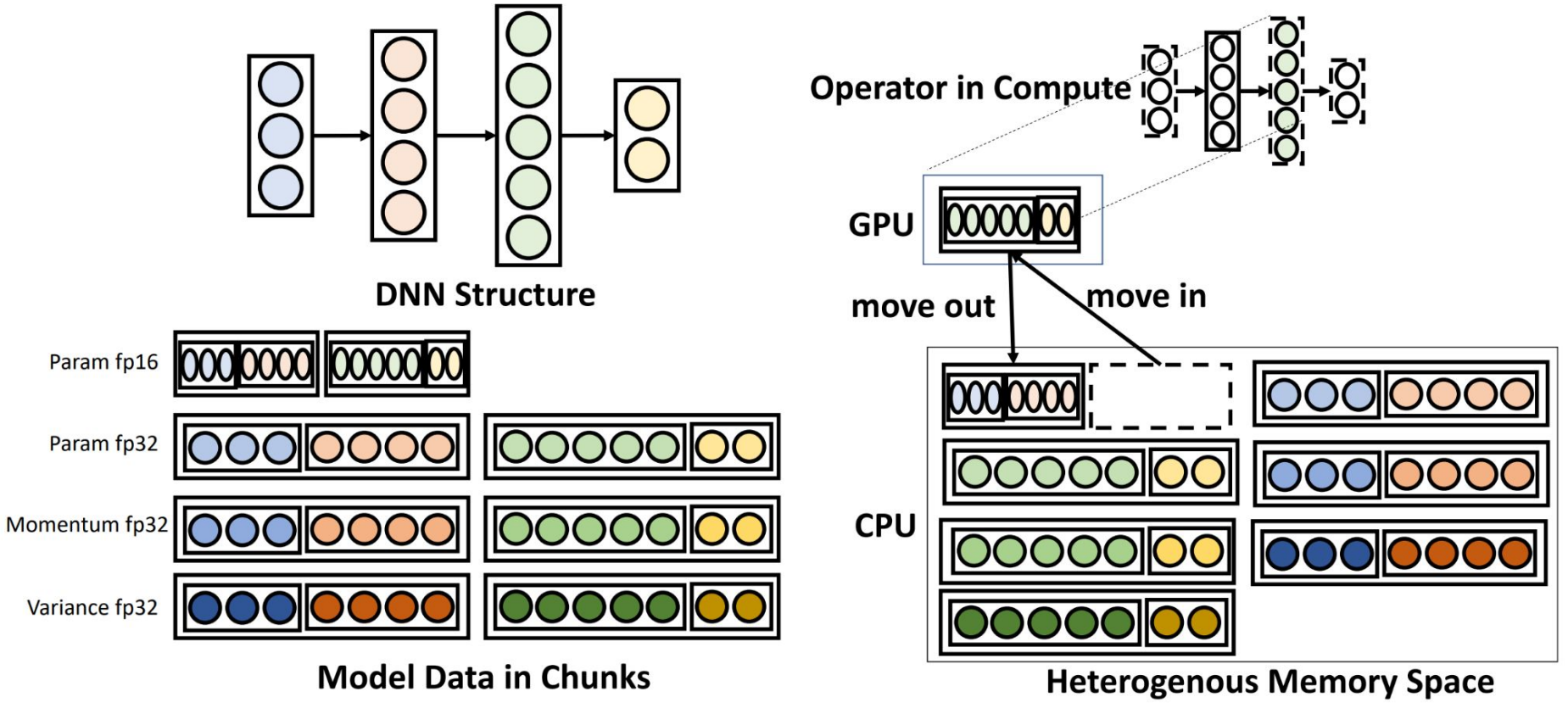
Rajbhandari, Samyam, et al. "Zero: Memory optimizations toward training trillion parameter models." SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020.

Existing Solution



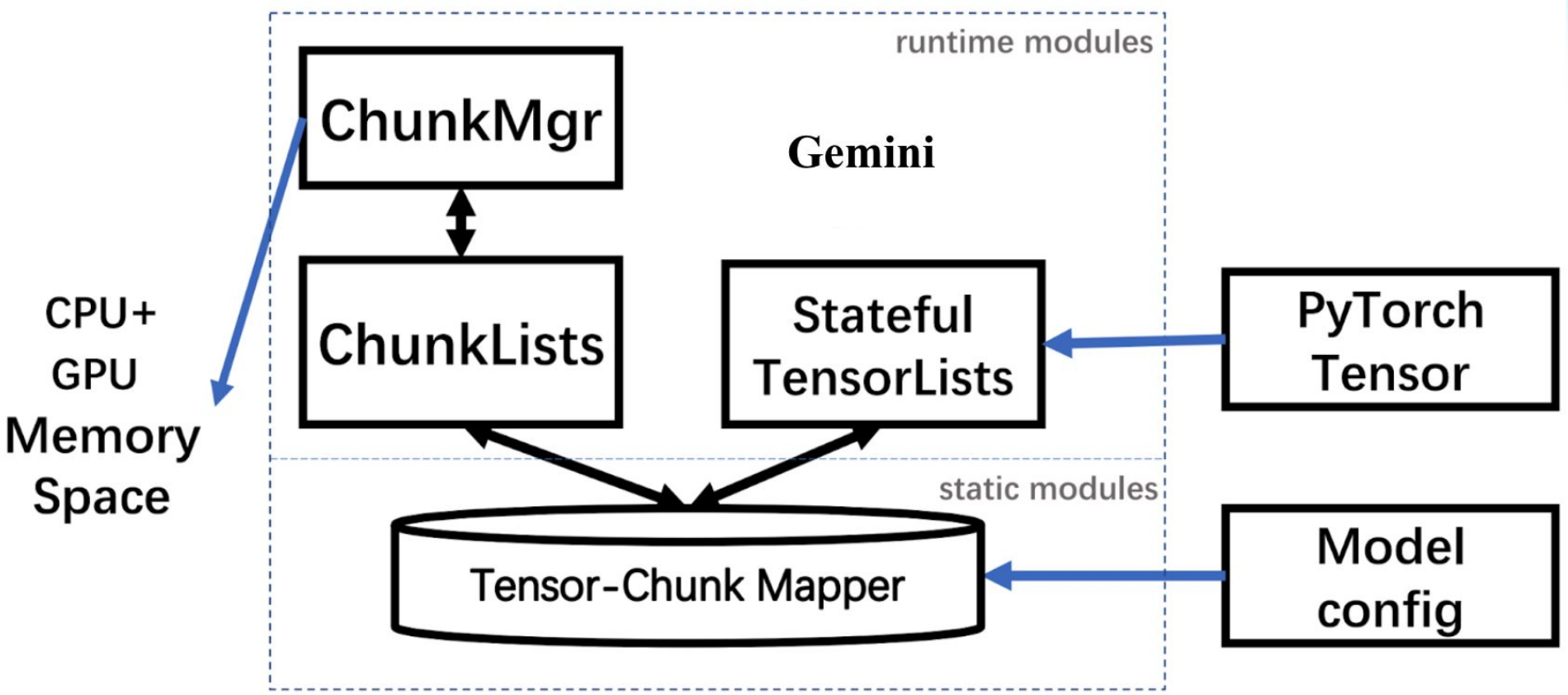
- The static memory partition in DeepSpeed

Colossal-AI: Heterogeneous Memory Management



- Chunk strategy of Colossal-AI

Colossal-AI: Heterogeneous Memory Management

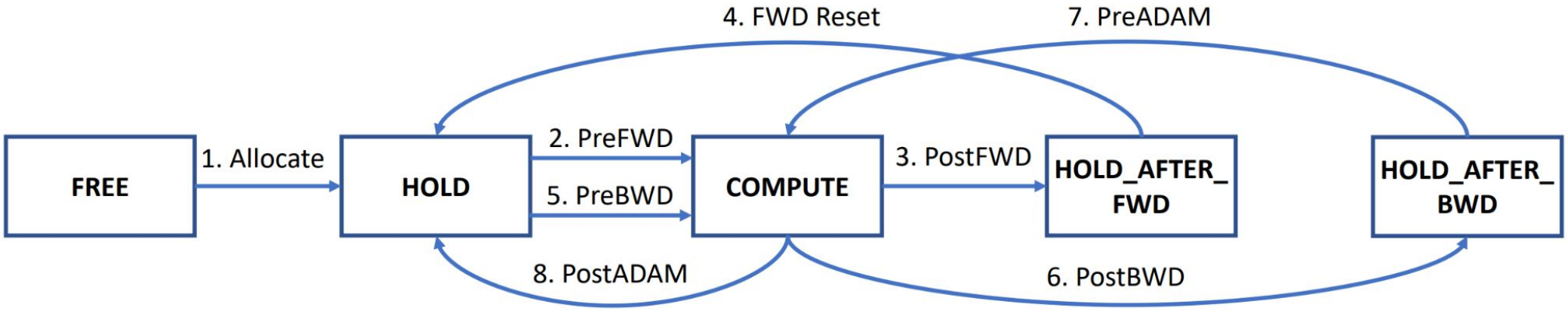


- The Gemini architecture of Colossal-AI

Colossal-AI: Heterogeneous Memory Management

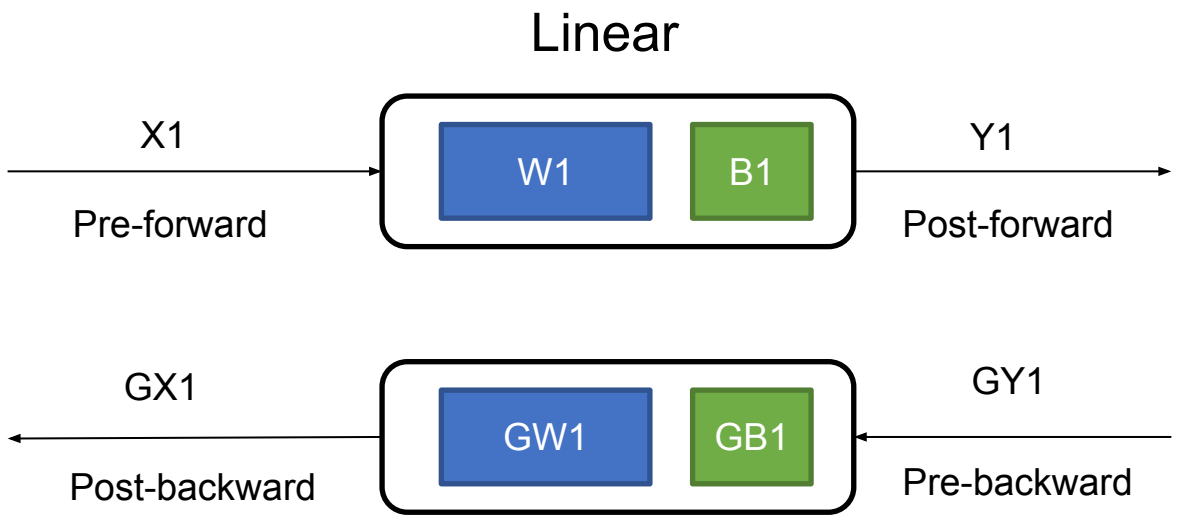
STATE NAME	EXPLANATION	PLACEMENT
FREE	No payload space	-
COMPUTE	Participate in computing	Computing Device
HOLD	Hold payload	CPU or GPU
HOLD_AFTER_FWD	Hold payload after FWD	CPU or GPU
HOLD_AFTER_BWD	Hold payload after BWD	CPU or GPU

- States of Tensor in Colossal-AI



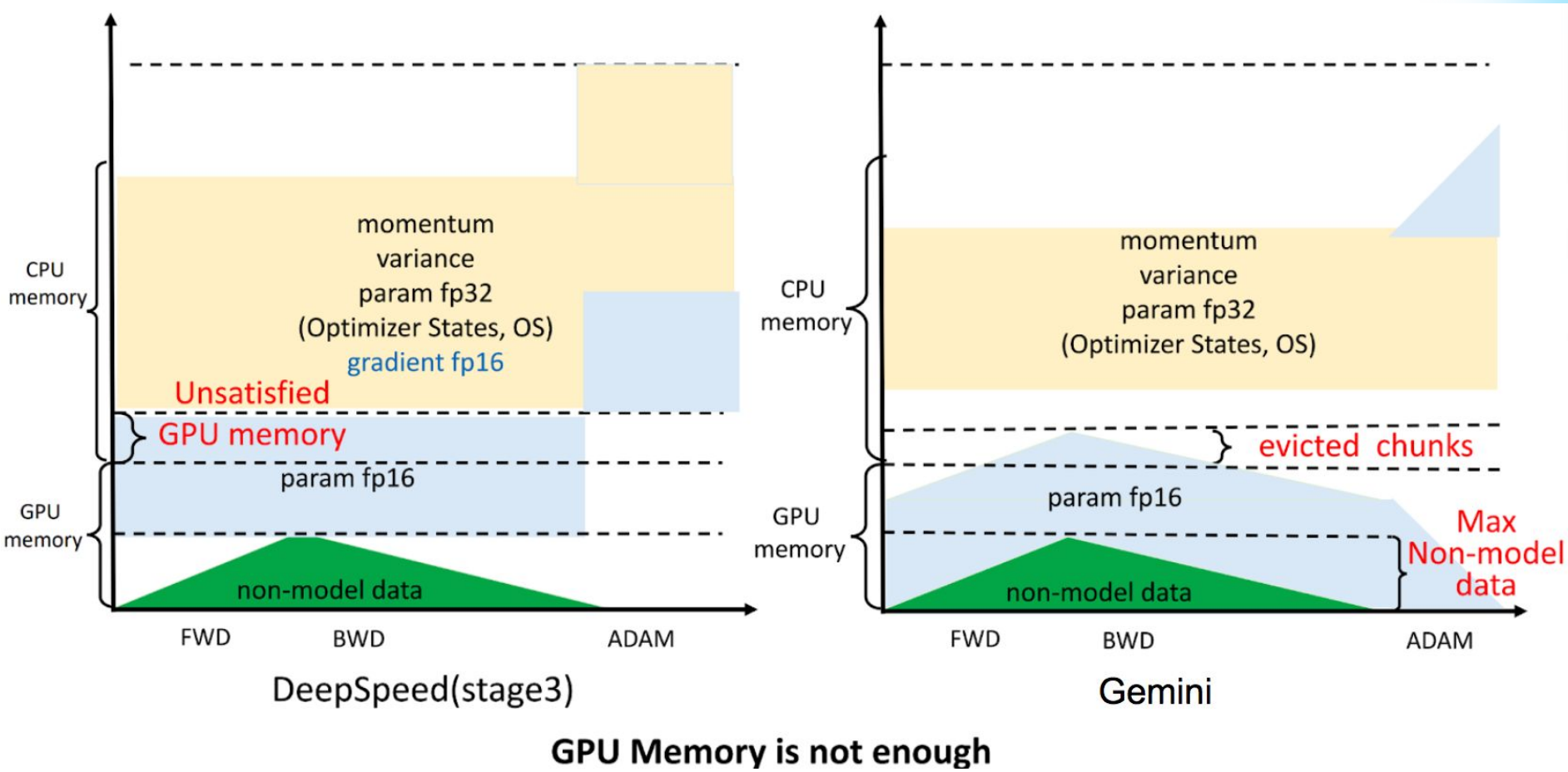
- The state transition diagram of a param fp16

Colossal-AI: Heterogeneous Memory Management



- The OP hook of Gemini

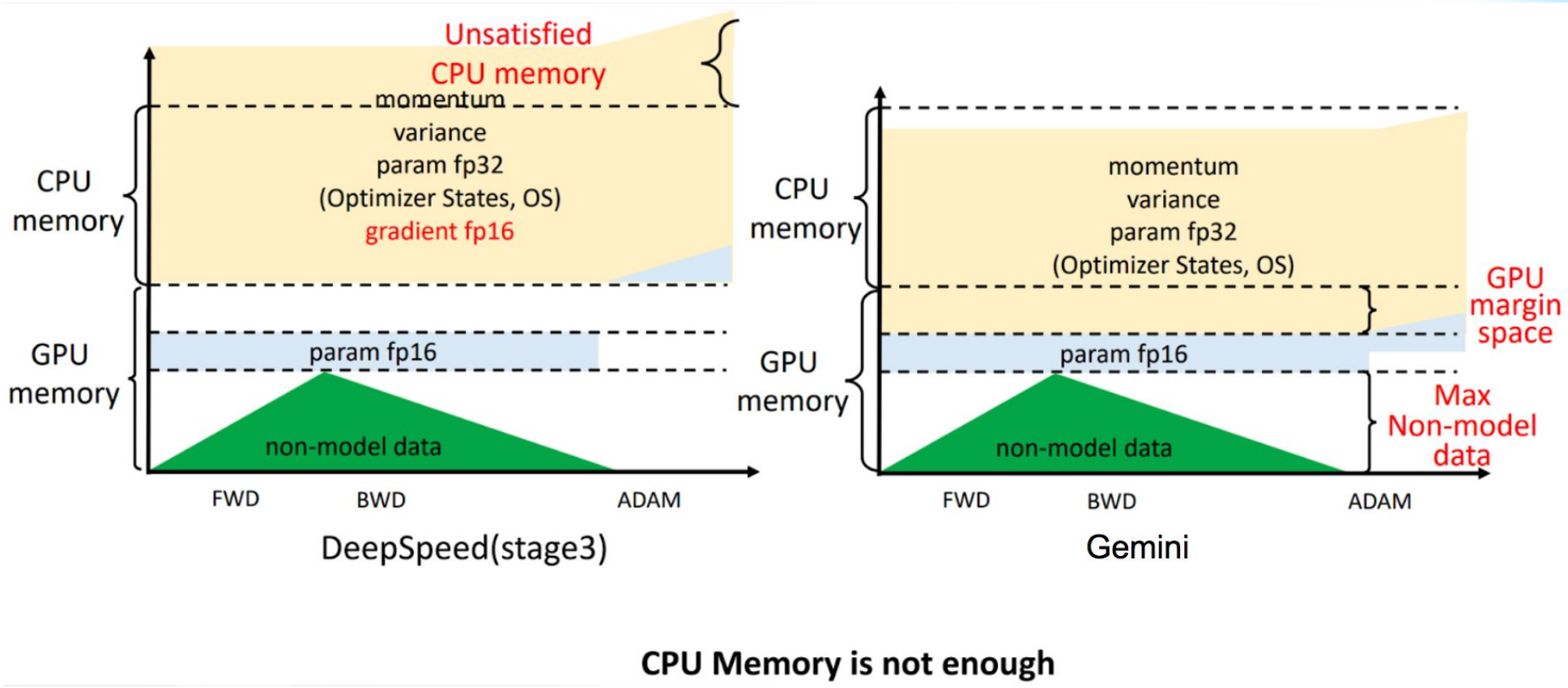
Colossal-AI: Heterogeneous Memory Management



GPU Memory is not enough

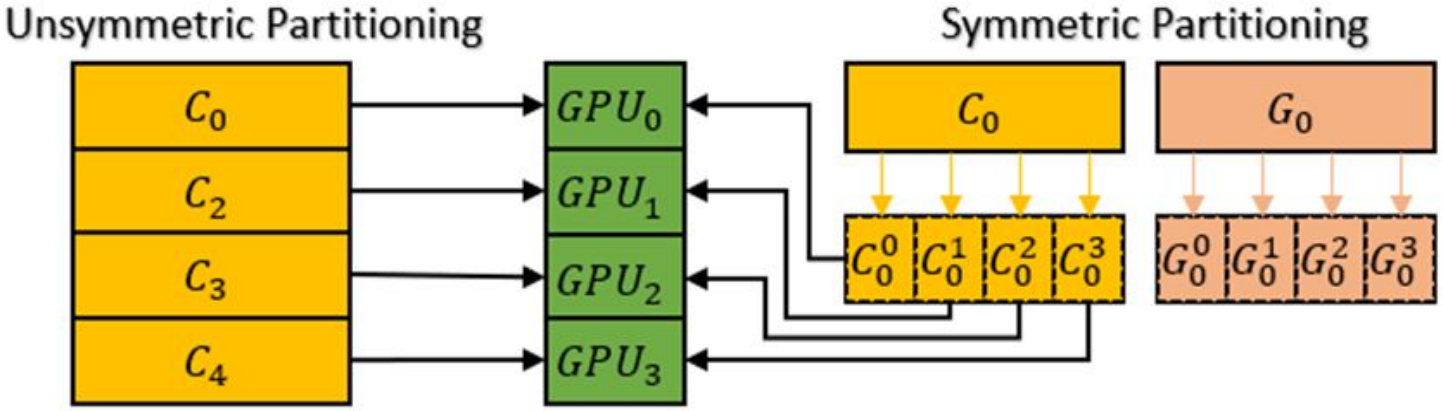
- Colossal-AI can handle situations where DeepSpeed is incompetent

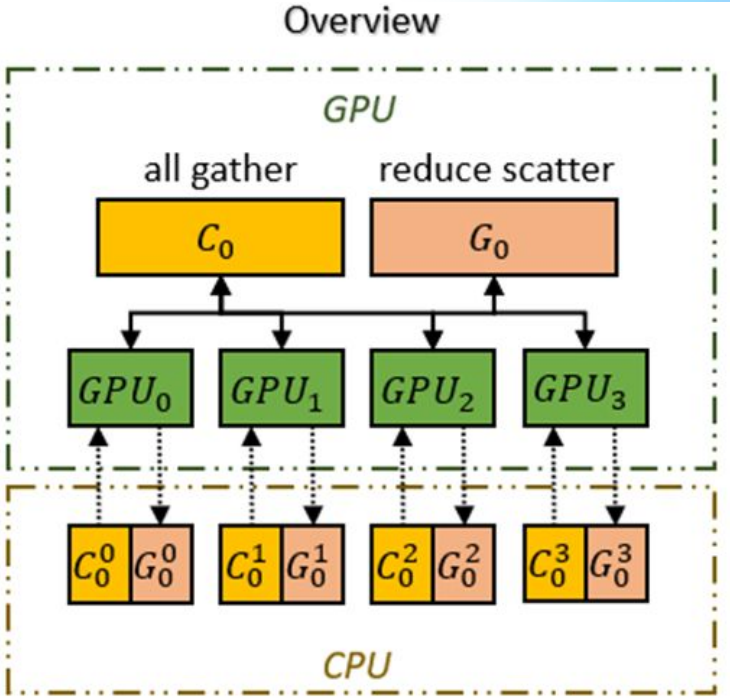
Colossal-AI: Heterogeneous Memory Management



- Colossal-AI can handle situations where DeepSpeed is incompetent

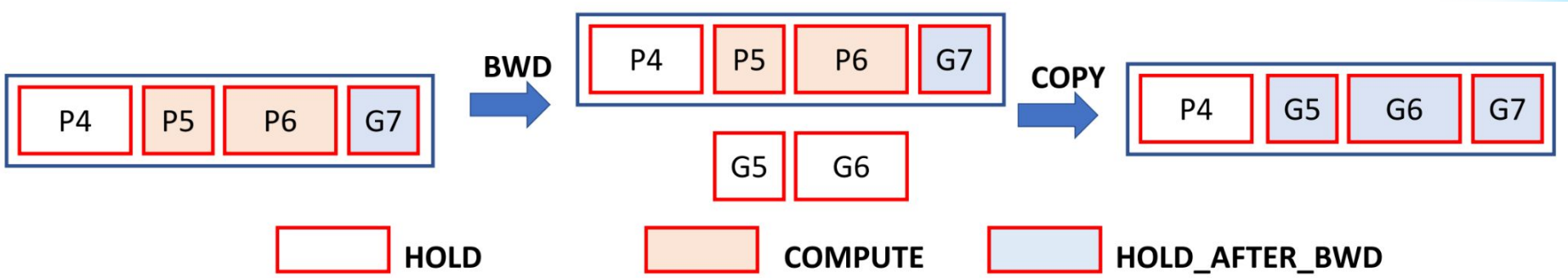
Colossal-AI: Heterogeneous Memory Management



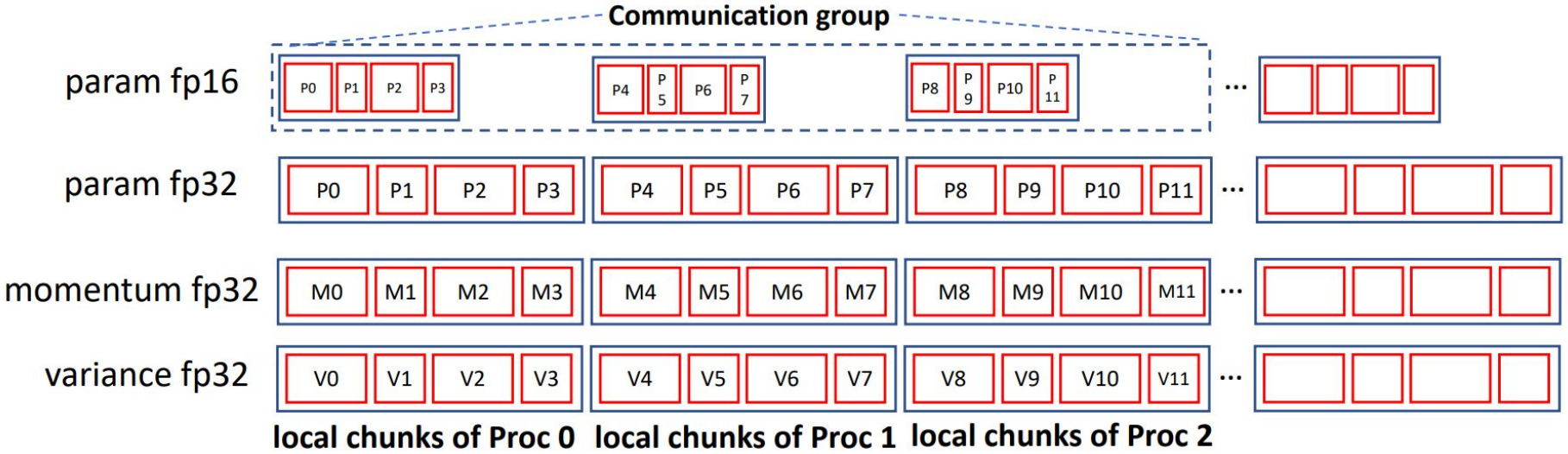


- Symmetric partitioning is applied to all chunks.
- Communication pattern for fetching remote param fp16 chunks. (right)

Colossal-AI: Heterogeneous Memory Management



- Reusing param fp16 chunk with grad fp16. Operator computes on param 5 and 6. P is param and G is grad.



- Distributing chunks in a three GPU training task

Towards AI Democratization and Low-cost Fine-tuning of Large AI Model

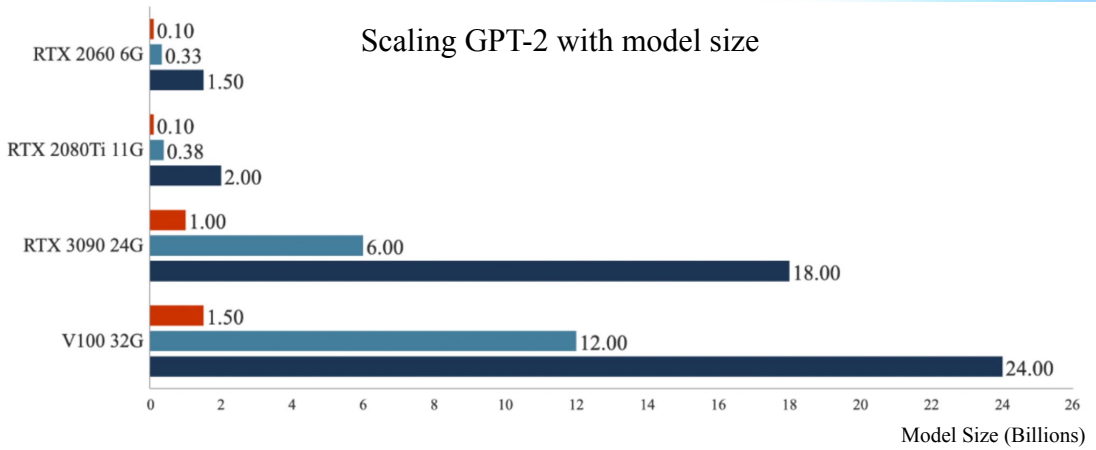
Break the GPU's Memory Barrier

A Single GPU « A Large Model

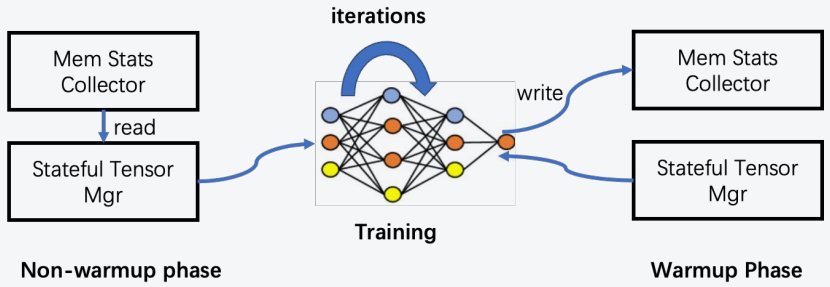
CPU Memory « GPU Memory

Scaling up to 34x Larger Model Size

Colossal-AI



- Dynamic
- Fast
- Low Cost

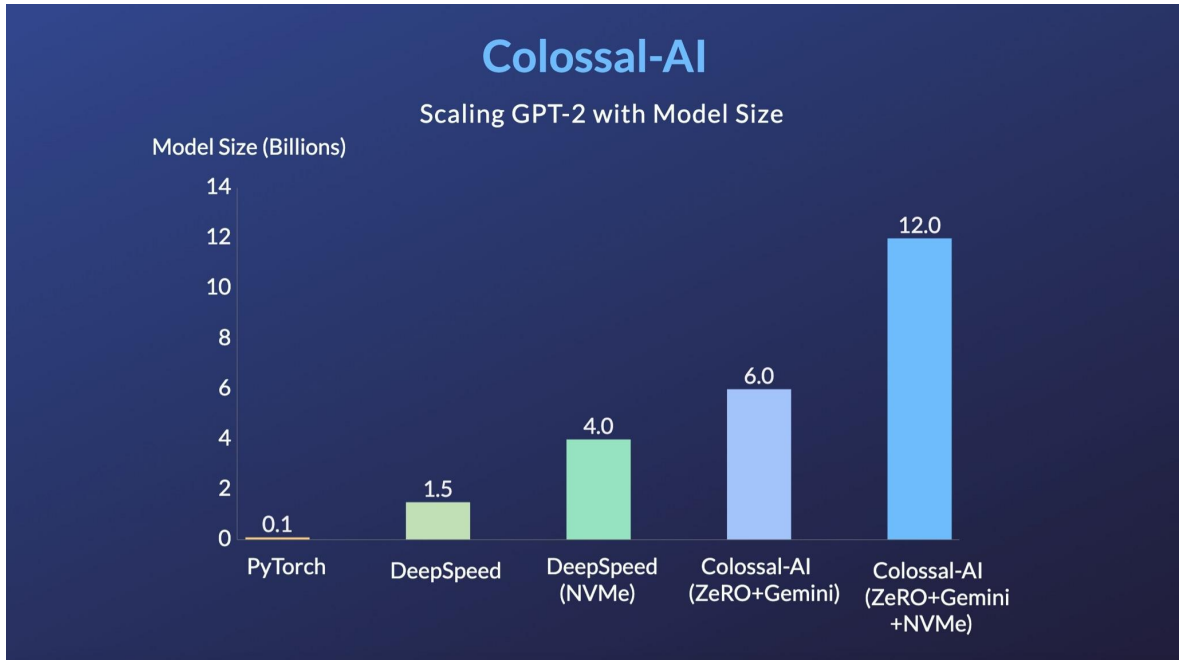


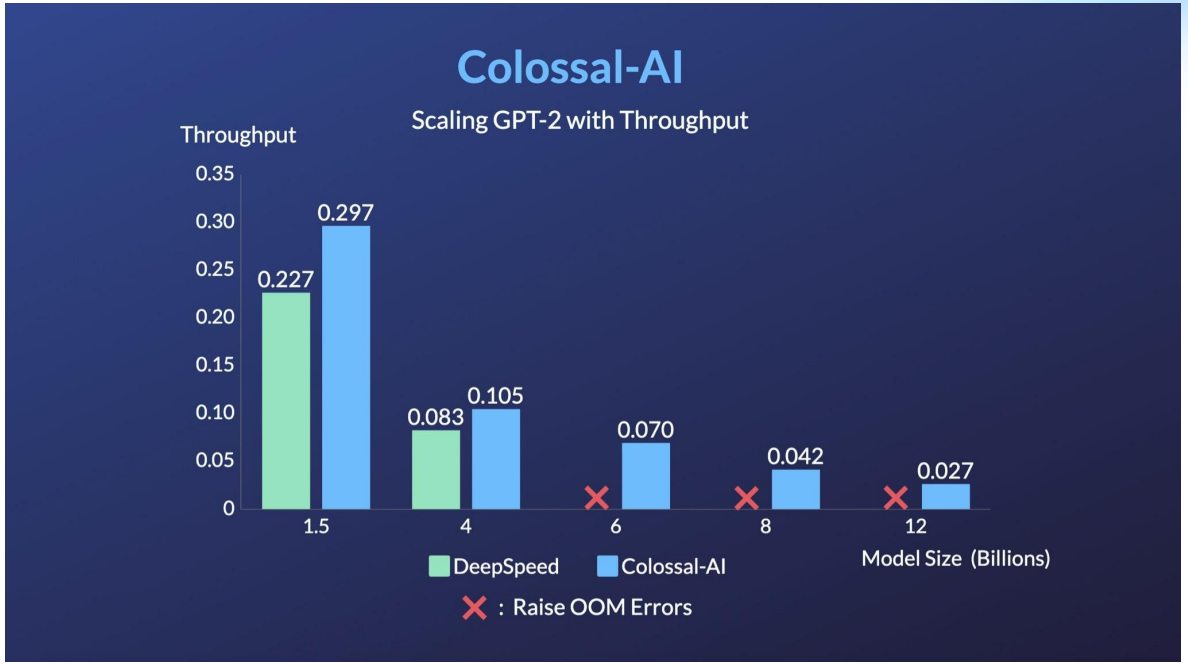
✔ **Remove the bottlenecks of ZeRO**
Dynamic storage space management of CPU-GPU

✔ **Training acceleration under limited hardware**
Minimization of CPU-GPU data movement

✔ **Maximizing model capacity**
Heterogeneous memory management of CPU-GPU

GPT-2





- 120x larger model size on the same hardware, higher acceleration
- One RTX 3080

Outline

1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

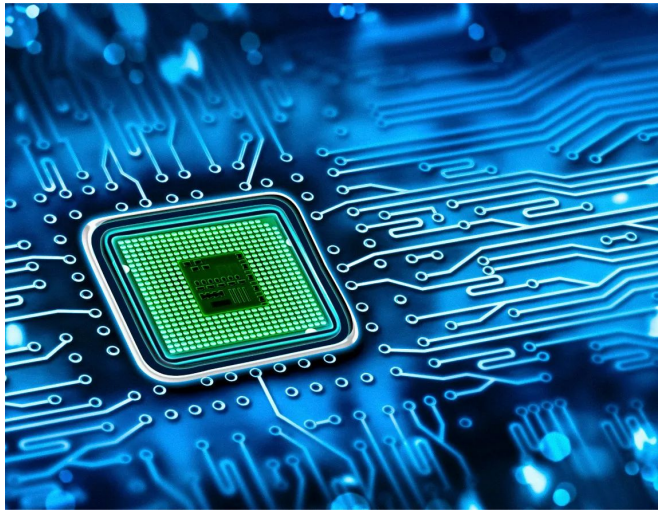
3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Colossal-AI Cases

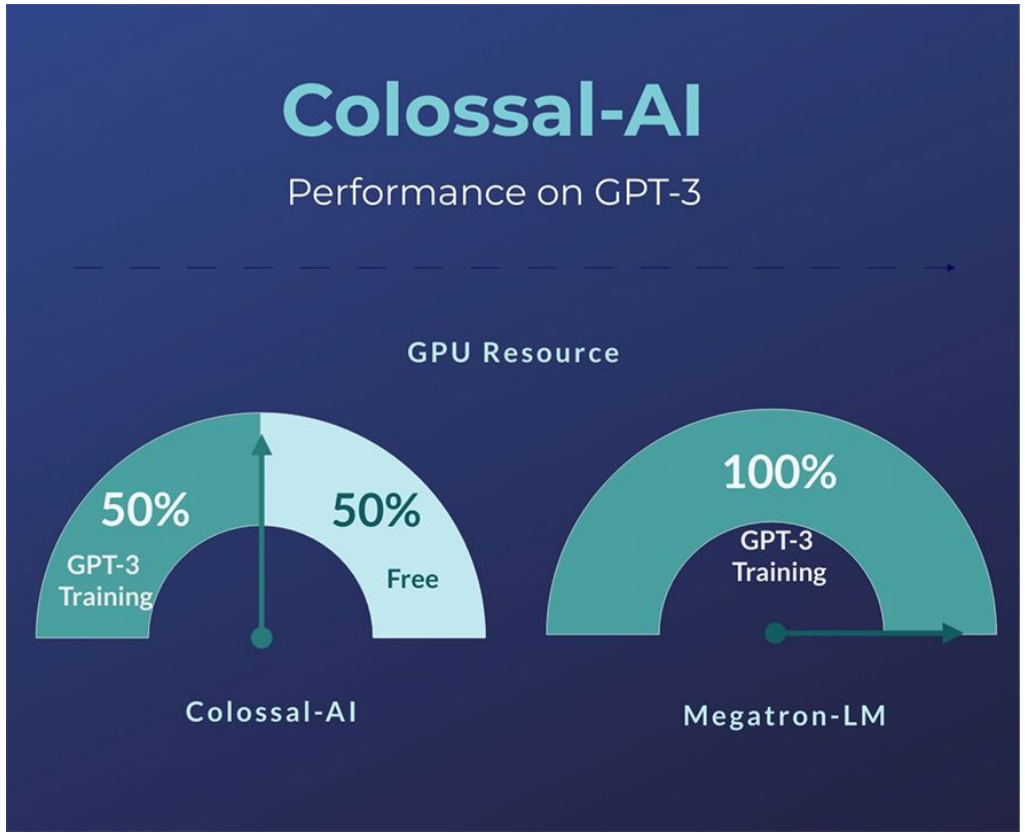
What we can do with Colossal-AI ?

CV, NLP, autonomous driving, retail, medicine, chip, etc.



https://www.megainteresting.com/techno/article/this-artificial-intelligence-chip-recognizes-images-in-nanoseconds-761584442074
https://www.deepnorth.com/resource_posts/a-short-guide-on-computer-vision-and-its-various-applications/
https://www.fia.com/autonomous-vehicles
https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

GPT-3

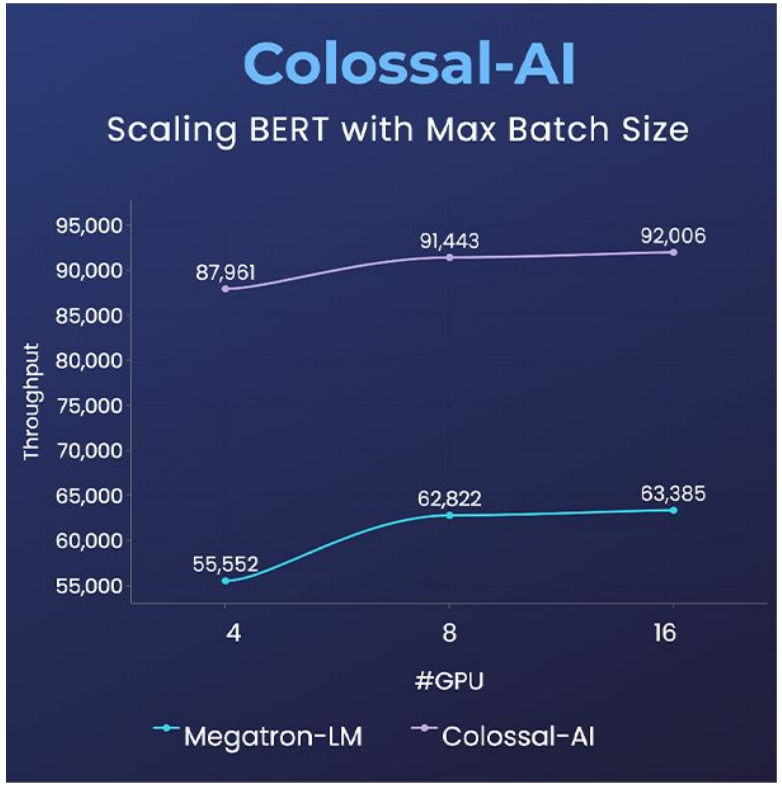


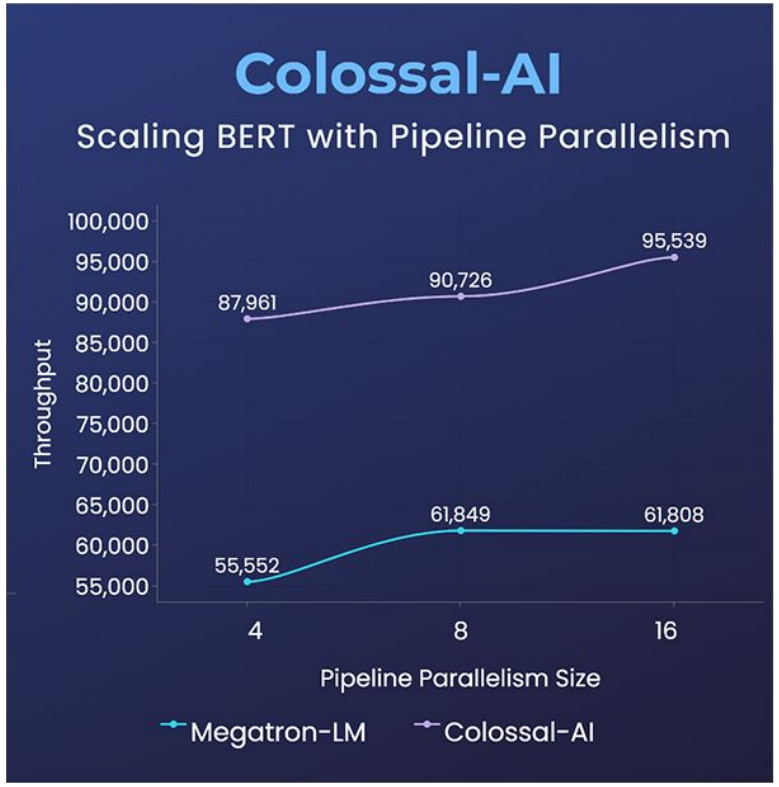
Solutions	# GPU	Sec/Iter	GPU RAM	Throughput
Megatron-LM	128	43.1	36 GB	4.45
Colossal-AI	128	**38.5**	36 GB	**4.99**
Colossal-AI	**96**	48.5	40 GB	3.96
Colossal-AI Gemini	**64**	111	39 GB	1.73

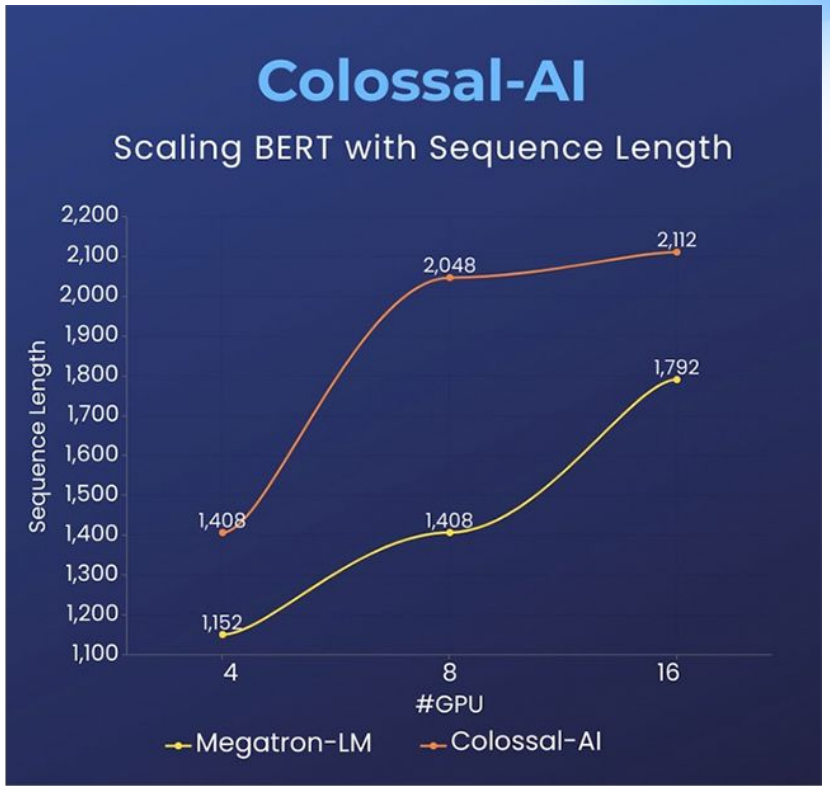
Colossal-AI for GPT-3

- Save 50% GPU resources, and 10.7% acceleration

BERT

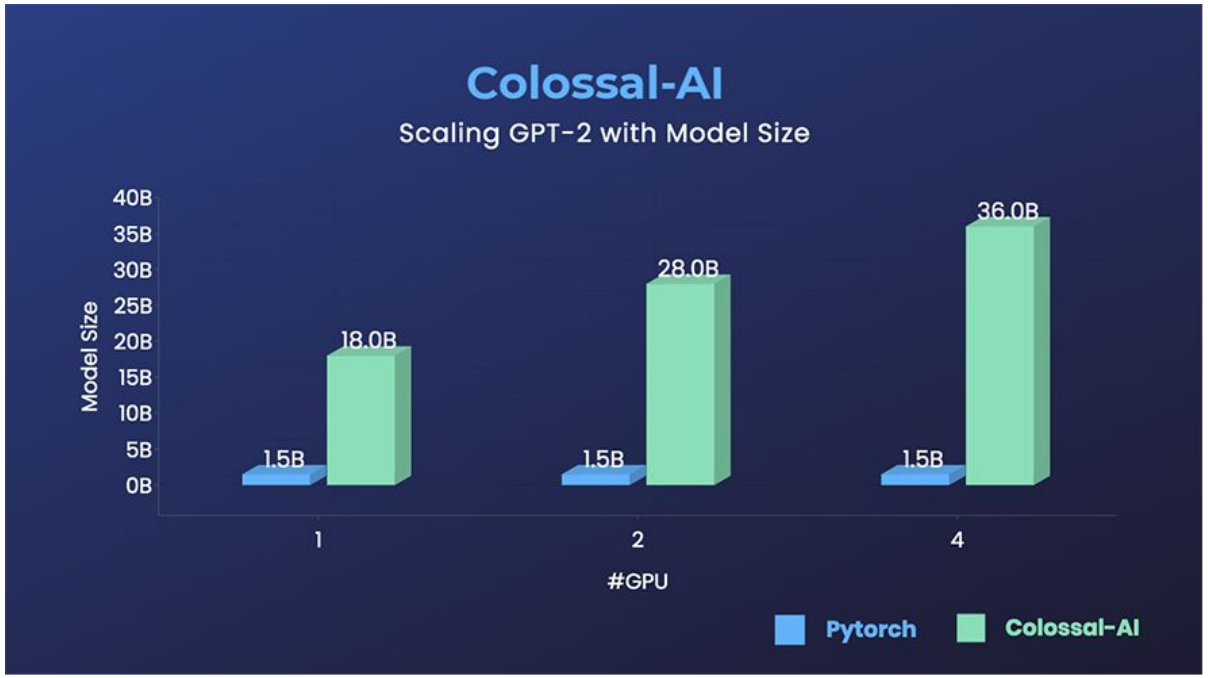


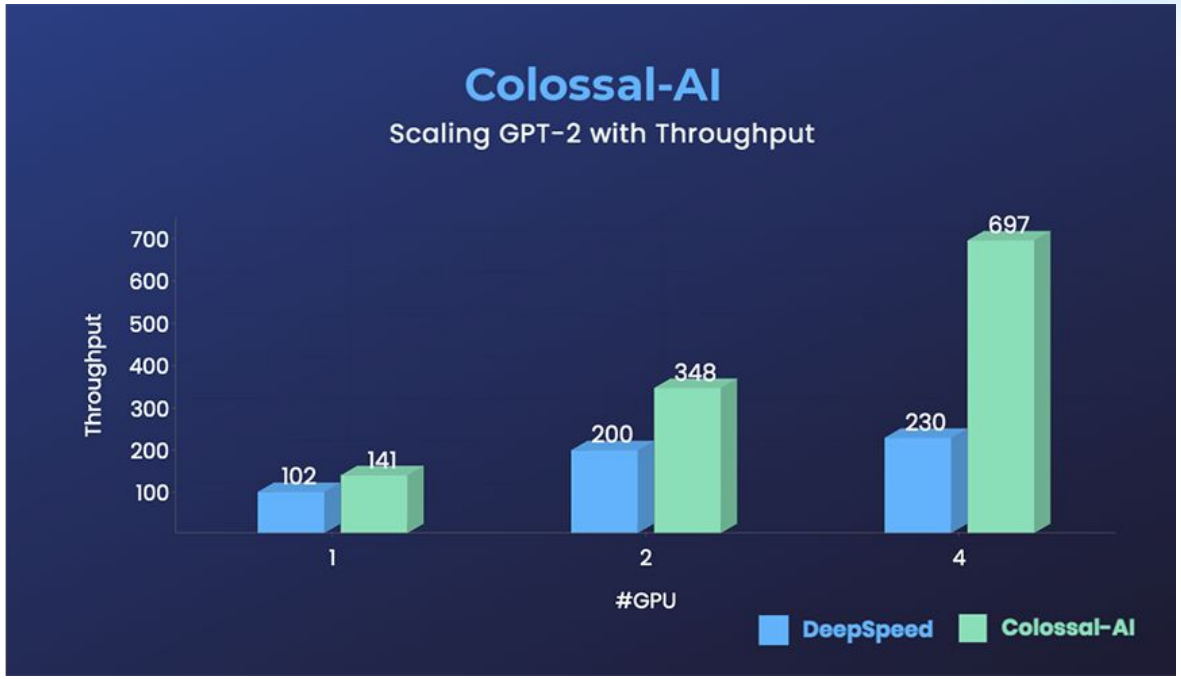




- Up tp 2x faster training, or 50% longer sequence length

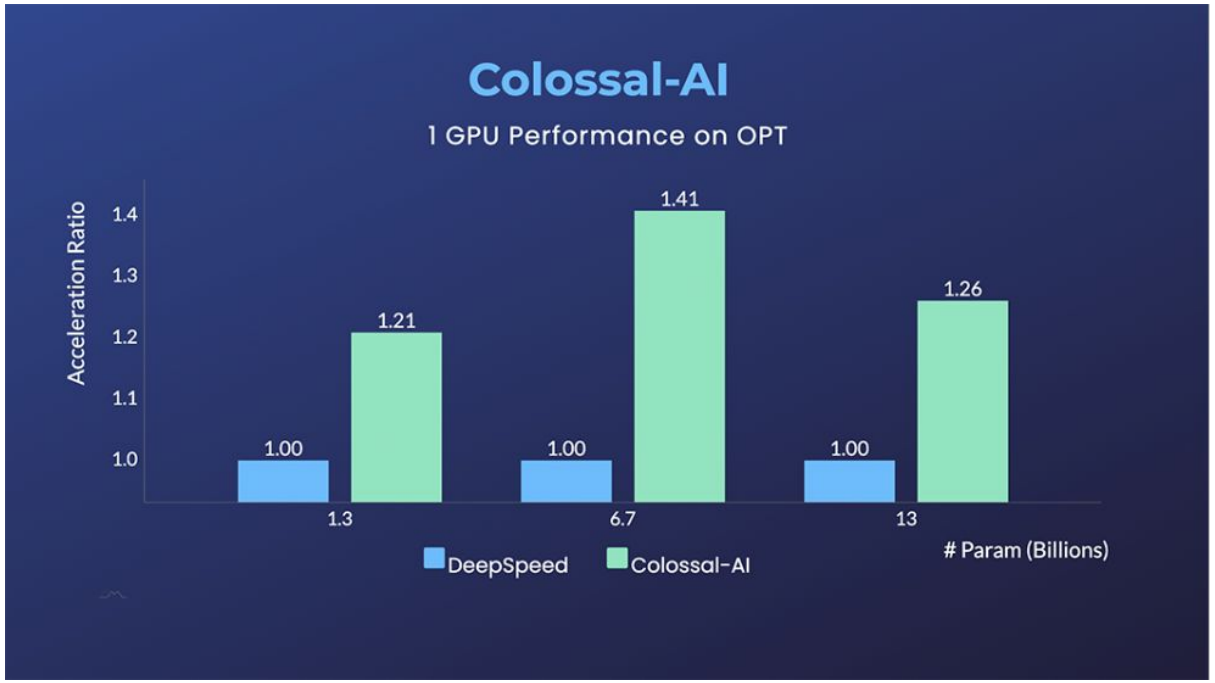
GPT-2

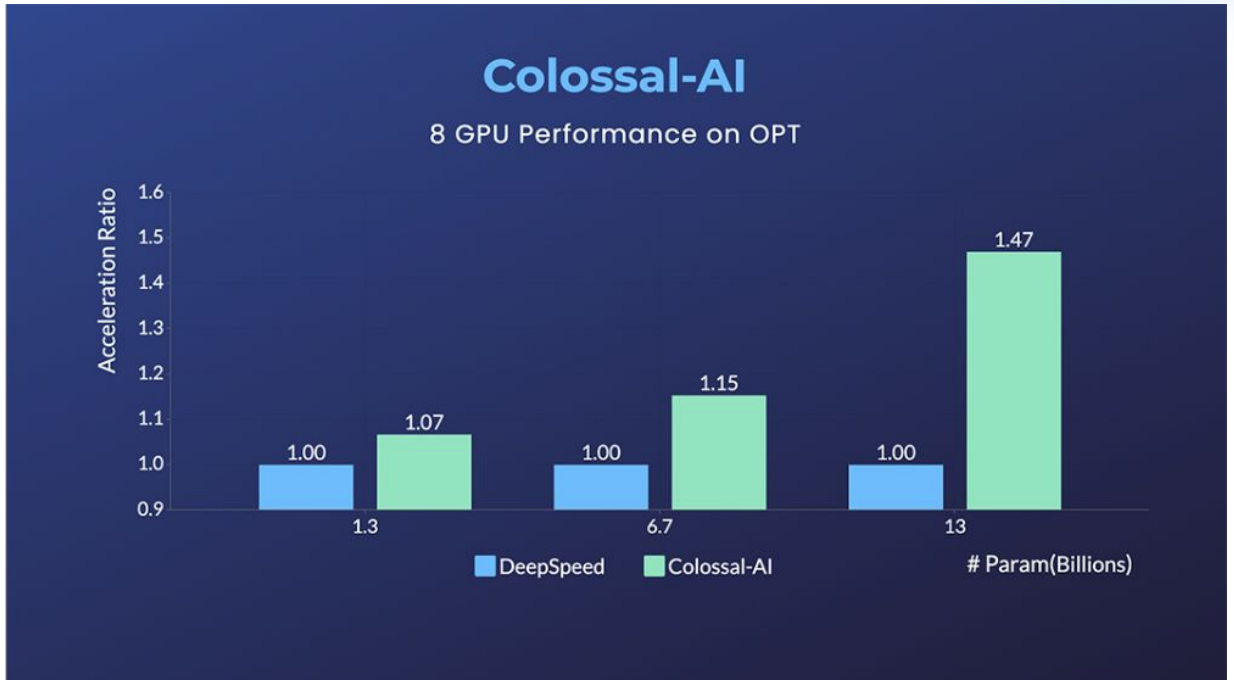




- Up to 24x larger model size on the same hardware, over 3x acceleration

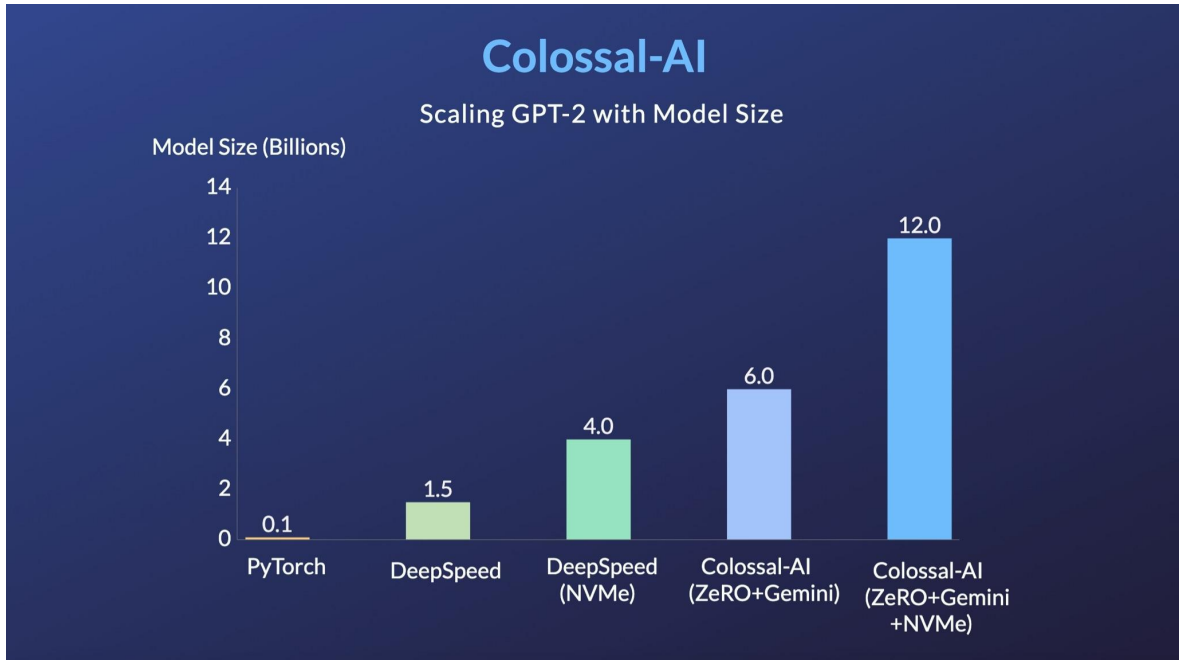
OPT

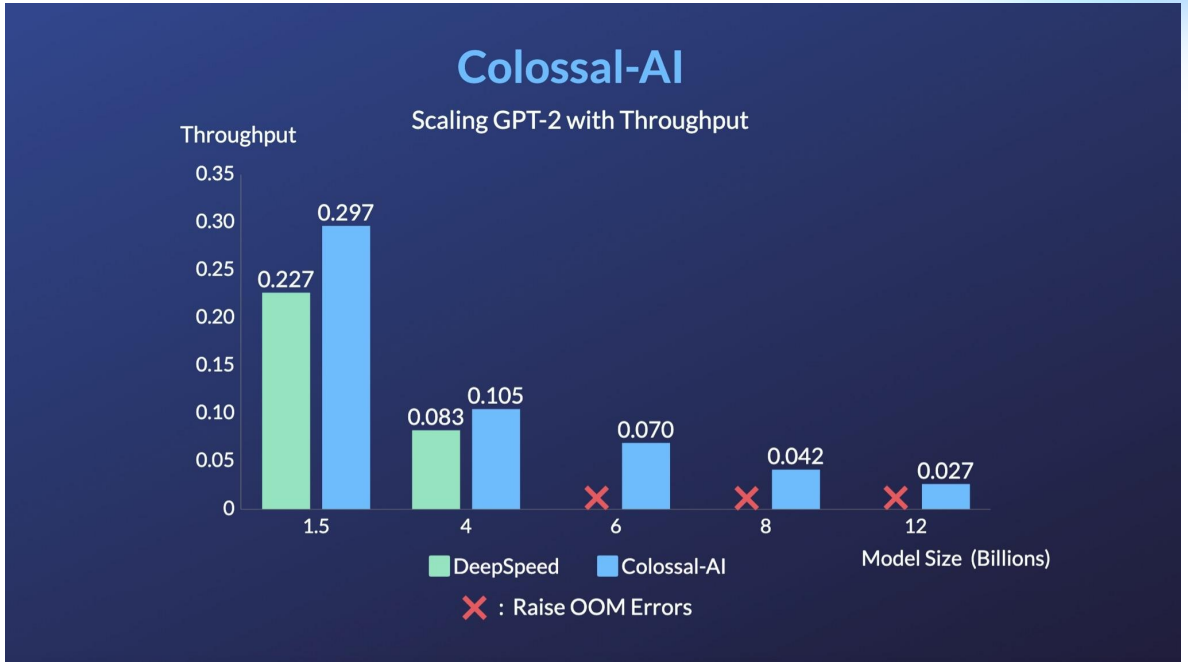




- Up to a 47% speedup

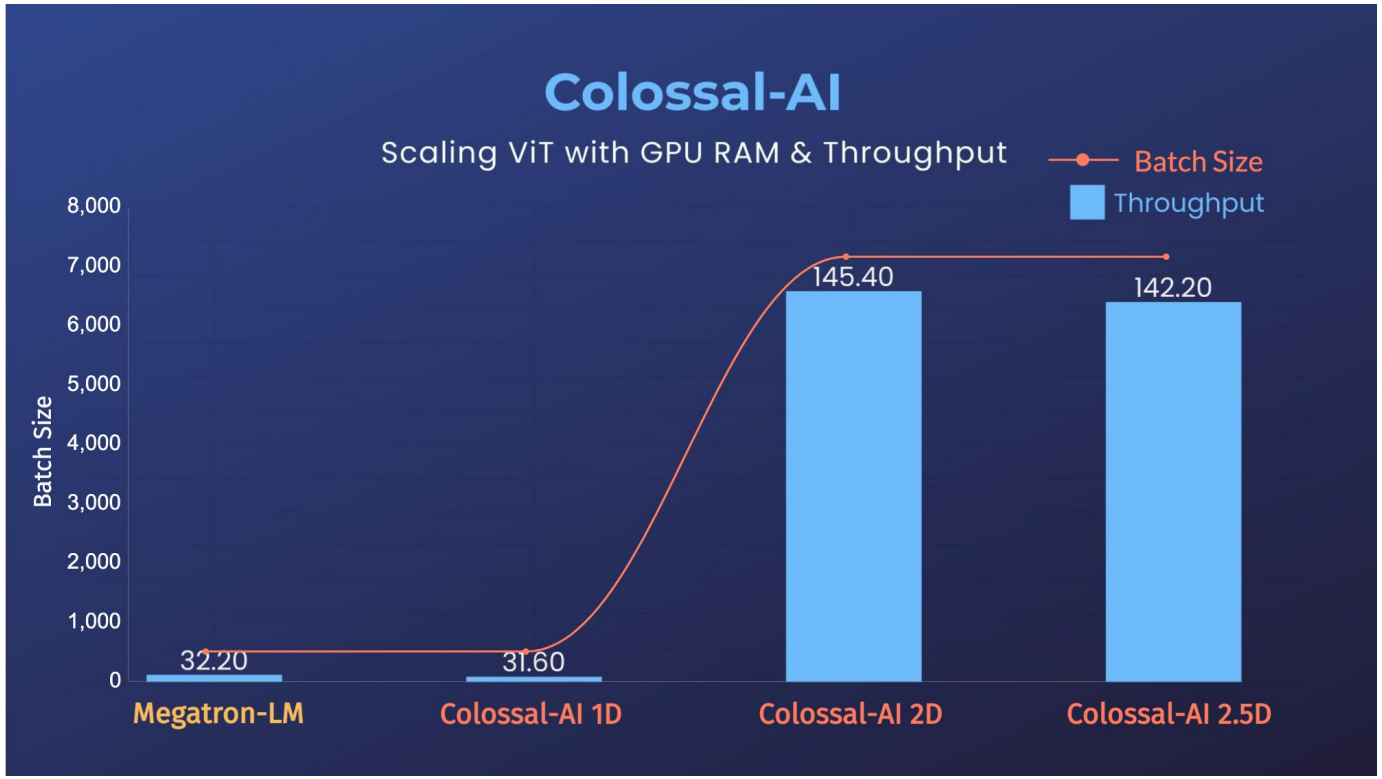
GPT-2





- Up to 120x larger model size on the same hardware, higher acceleration
- One RTX 3080

ViT



- Up to 14x larger batch size, and 5x faster training for Tensor Parallelism = 64.

Stable Diffusion - Training

Acceleration of Colossal-AI to Stable Diffusion

Strategy	Device	# GPU	Batch Size	Precision	GPU RAM(GB)	Speed Up
Stable Diffusion	A100	4	32	FP32	67	1
			64	FP16	65	2.11
Colossal-AI			64		30	3.51
			128		54	4.55
			256		78.2	6.47

Memory Reduction of Colossal-AI to Stable Diffusion

Strategy	Device	Batch Size	Precision	GPU RAM(GB)
Colossal-AI	3080	8	FP16	9
		4		6.9
		2		5.9
Stable Diffusion	3090	2		22

- Up to 6.5x faster training and pretraining cost saving
- The hardware cost of fine-tuning can be almost 7X cheaper

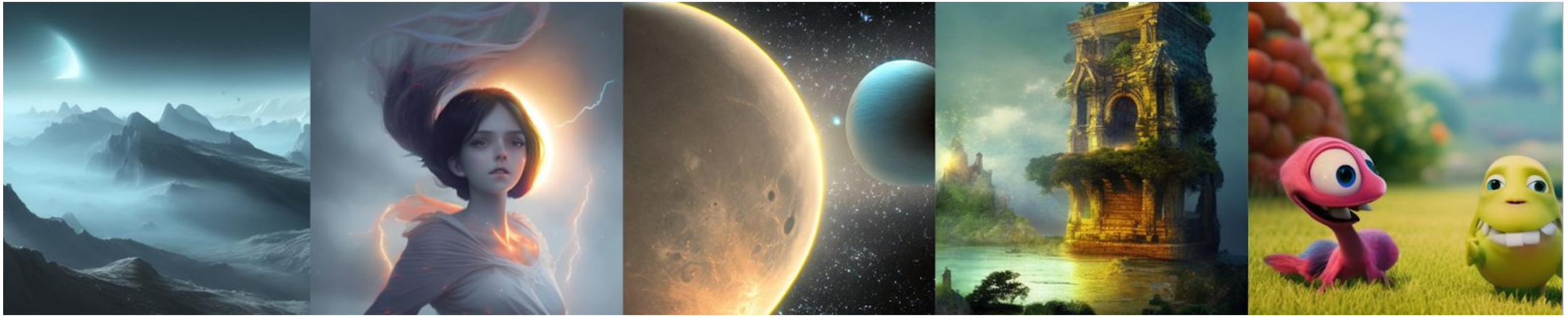
Stable Diffusion - Training

Model	Strategy	Device	# GPU	Flash Attention	Batch Size	GPU RAM (GB)
Stable Diffusion v1	DDP	A100	4	False	64	64.5
Stable Diffusion v2				True		31.9
Stable Diffusion v1	Colossal-AI			False		30.0
Stable Diffusion v2				True		11.6

Model	Strategy	Device	# GPU	Flash Attention	Batch Size	GPU RAM (GB)
Stable Diffusion v2	DDP	A100	1	True	16	23.0
			4		64	31.9
			8		128	41.3
	Colossal-AI		1		16	5.6
			4		64	11.6
			8		128	19.8

- Reduce GPU memory consumption by up to 5.6x
- hardware cost by up to 46x (from A100 to RTX3060)
- Can be extended to single or multiple GPUs in parallel

Stable Diffusion - DreamBooth Fine-tuning



- Uses just 3-5 images of a desired subject to personalize text-to-image models

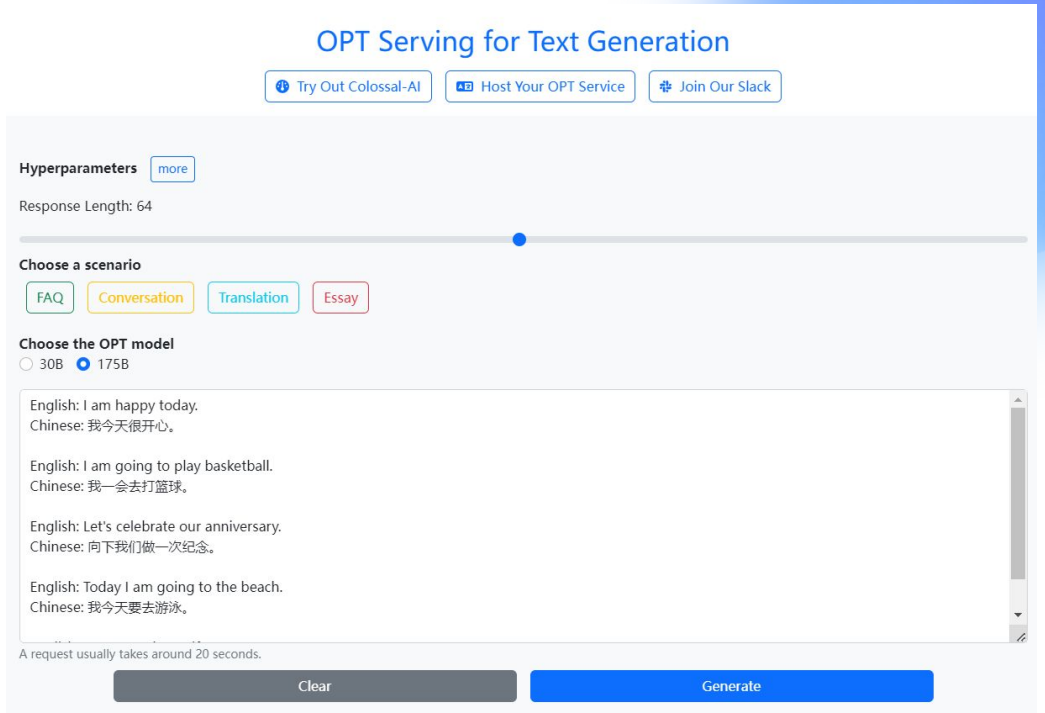
Stable Diffusion - Inference

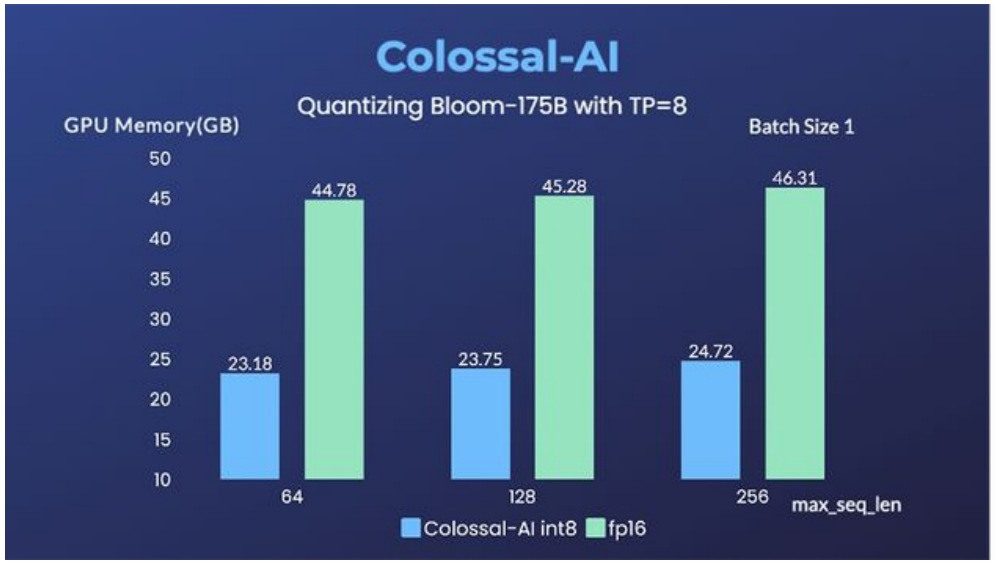
	FP32	FP32-Int8	FP16	FP16-Int8
GPU RAM (GB)	7.68	5.73	3.52	**3.10**
Performance				

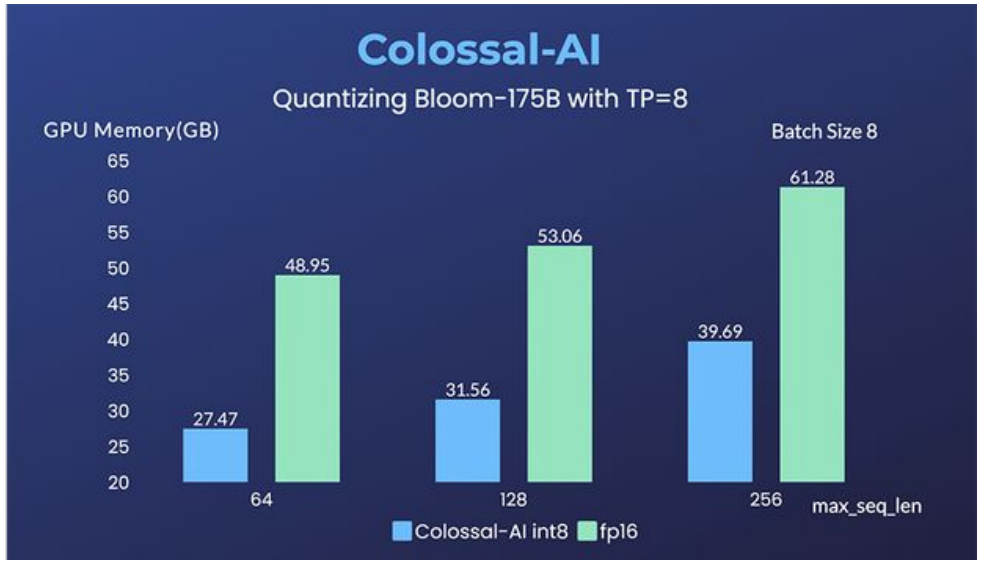
- 2.5 times lower memory consumption (3.1 GB memory required)
- Quantized for inference with a single line of code
  model = replace_module(model)

Colossal-AI Inference

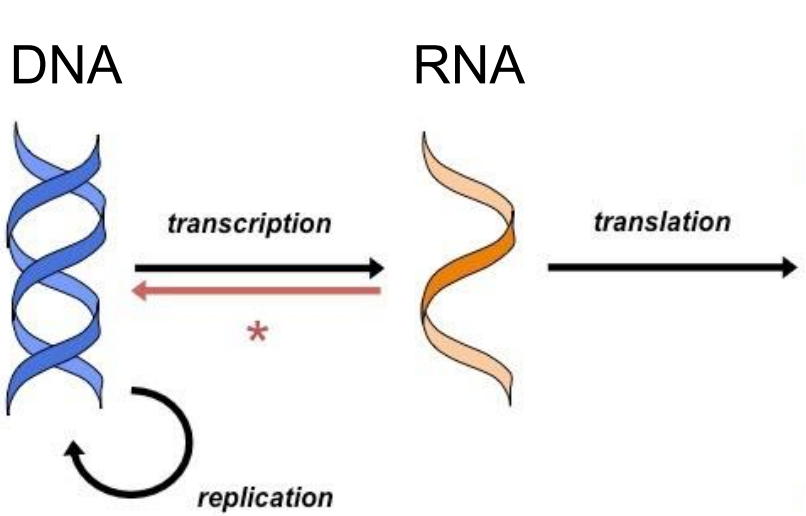
- Low-Cost Inference for 176B BLOOM
- Inference services on 8-GPU server using 3090/4090
- Reduce hardware deployment costs by more than 10x
- Online 175B OPT model serving demo

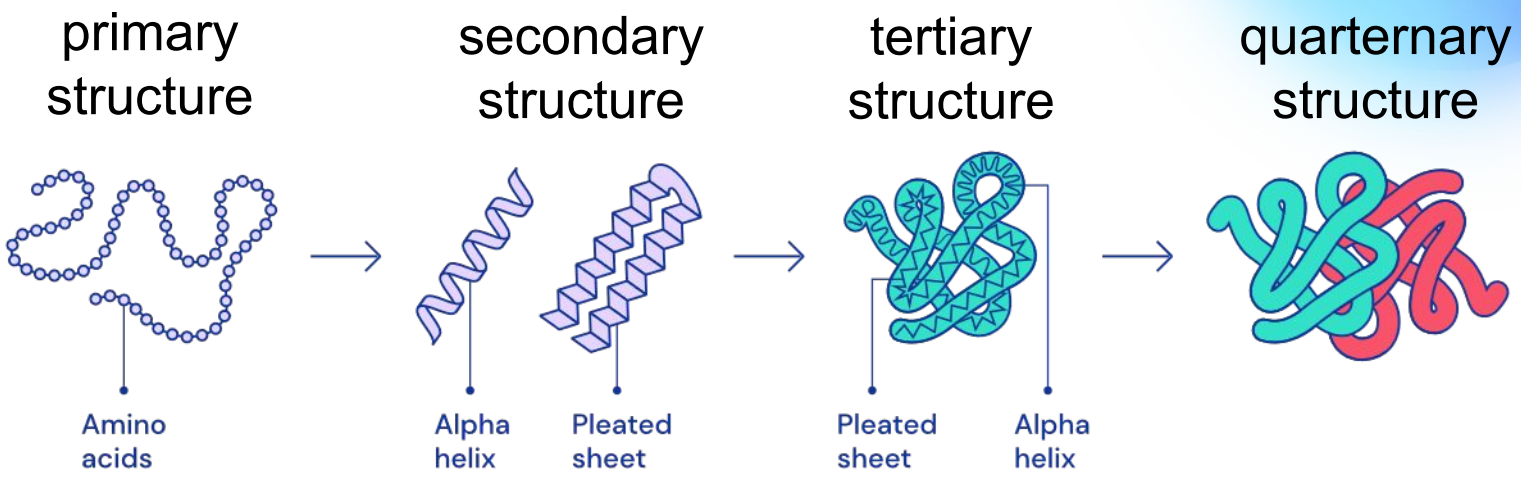






Protein Folding





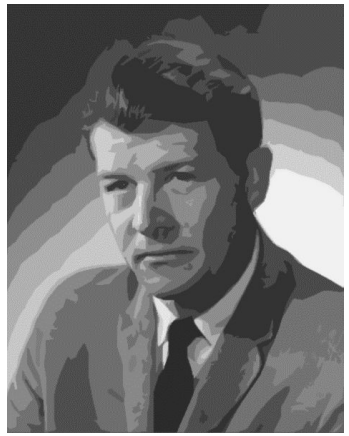
Process of Protein Folding

The central dogma of molecular biology

DNA → RNA → Protein

Anfinsen's Dogma (1972 Nobel Prize in Chemistry)

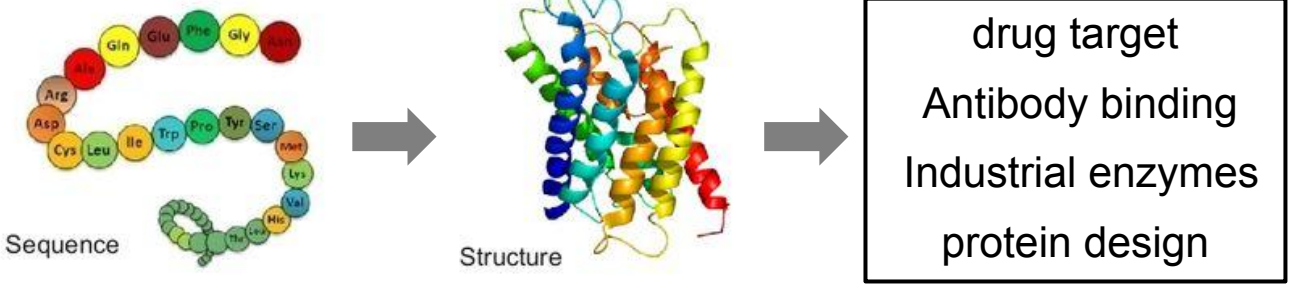
- **Uniqueness**
- **Stability**
- **Kinetical accessibility**

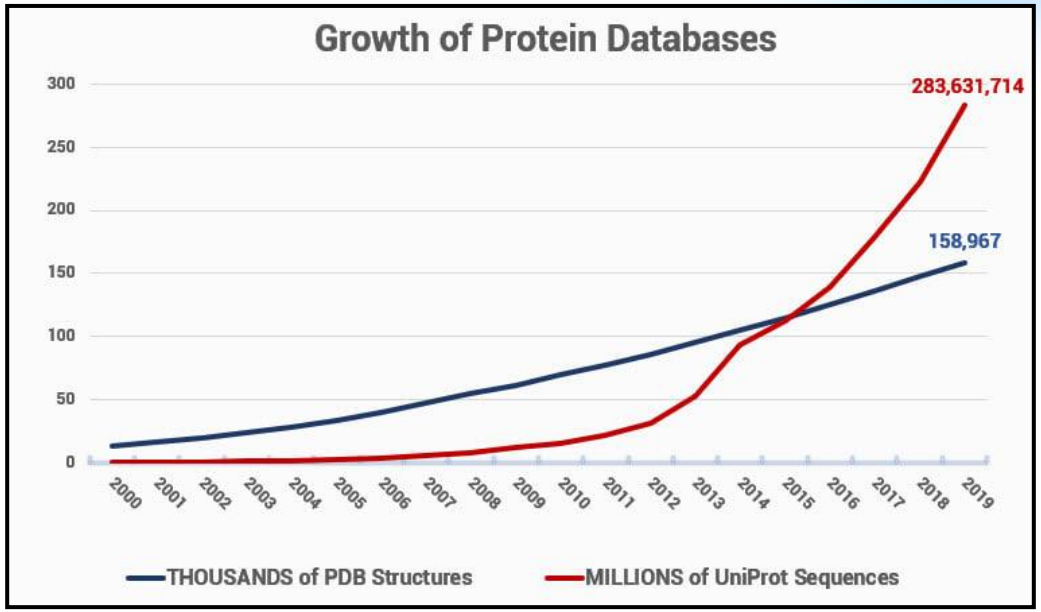


Picture from DeepMind

Anfinsen C B. Science, 1973, 181(4096): 223-230.

The Protein-Folding Problem, 50 Years On





Two approaches:

1. **Experimental Methods**: Low throughput, high accuracy
2. **Computational Methods**: High throughput, low accuracy

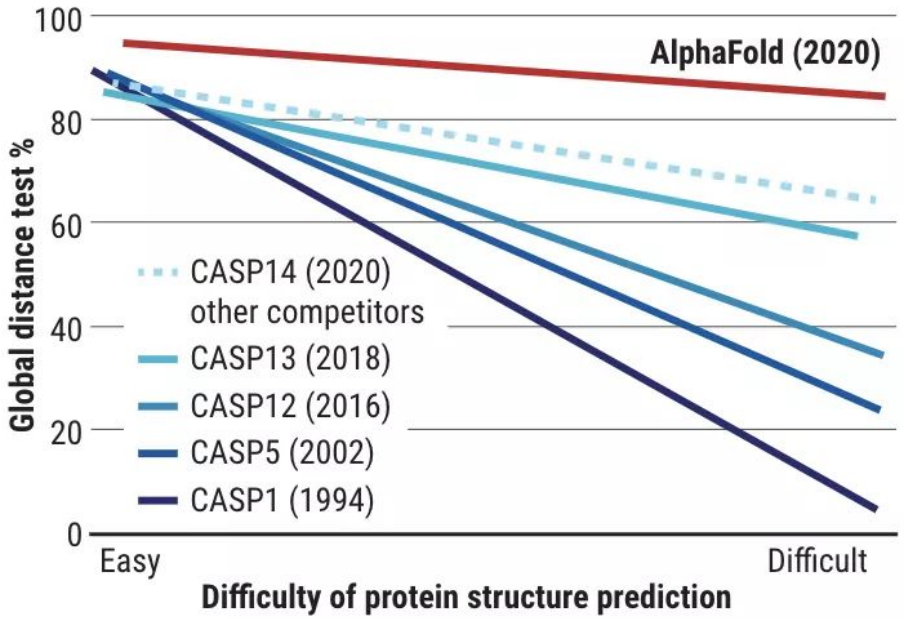
Massive sequence data
little structural data

Need high-precision, high-throughput protein structures predict methods!

https://www.dnastar.com/blog/structural-biology/why-structure-prediction-matters/

The Success of AlphaFold2

- 2016 : RaptorX-Contact (Convolution)
- 2018 : AlphaFold 1 (Convolution)
- 2020 : **AlphaFold 2 (Transformer)**



CASP

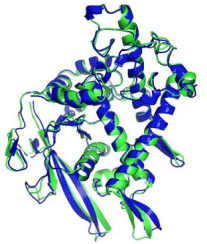
Critical Assessment of protein Structure Prediction

Experimental

Sequence

(unpublished)

Three Month



T1037 / 6vr4
90.7 GDT

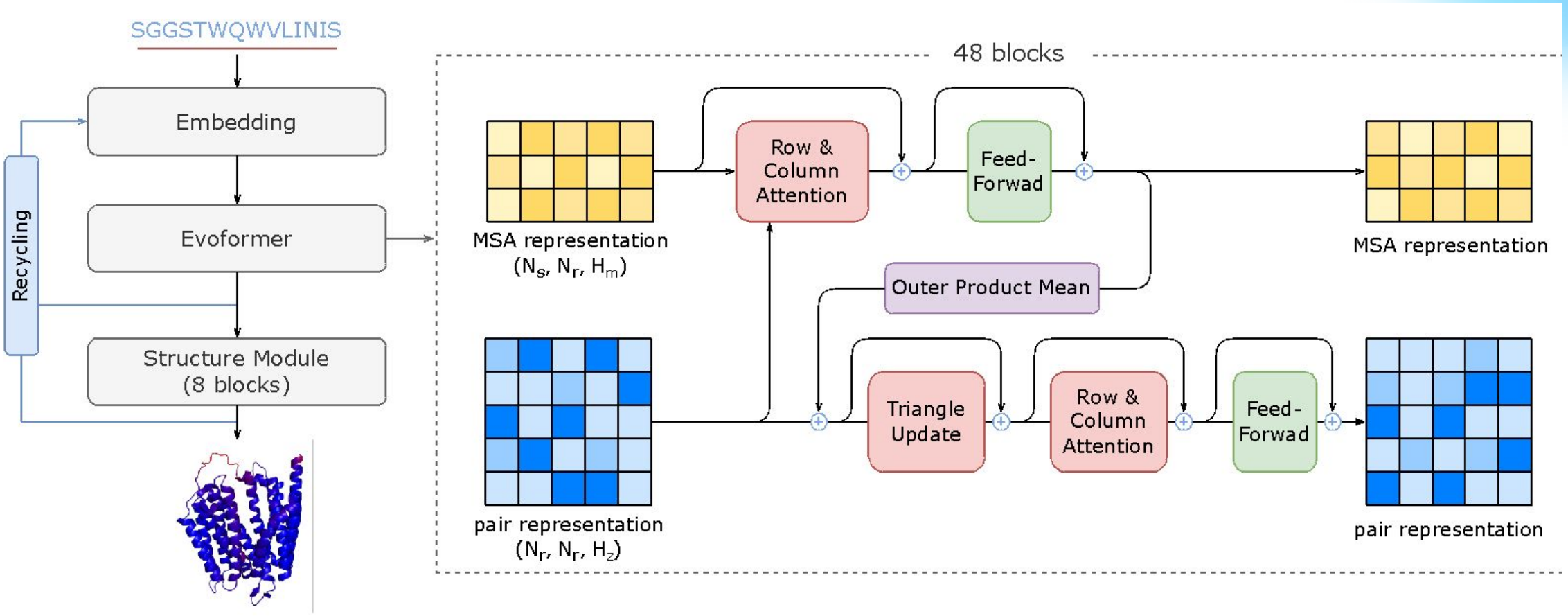
Announce Results on CASP

Computing

Evaluate Accuracy of Computing Methods

Picture from Jianyi Yang's Report and Science

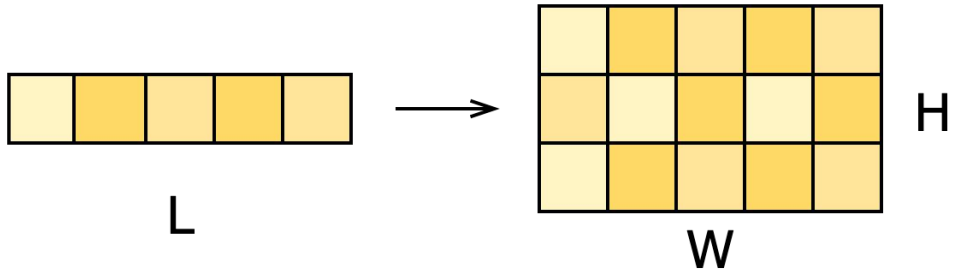
Architecture of AlphaFold



AlphaFold is an **end-to-end model** that uses amino acid sequences as model input and directly outputs the three-dimensional structure of the protein.

Computational Challenges of AlphaFold

Two sequence dimensions for intermediate representation



Computational Complexity ↑

Memory Consumption ↑

1) Time and Economic Costs — Training takes around **11 days** **on 128 Google TPUv3**

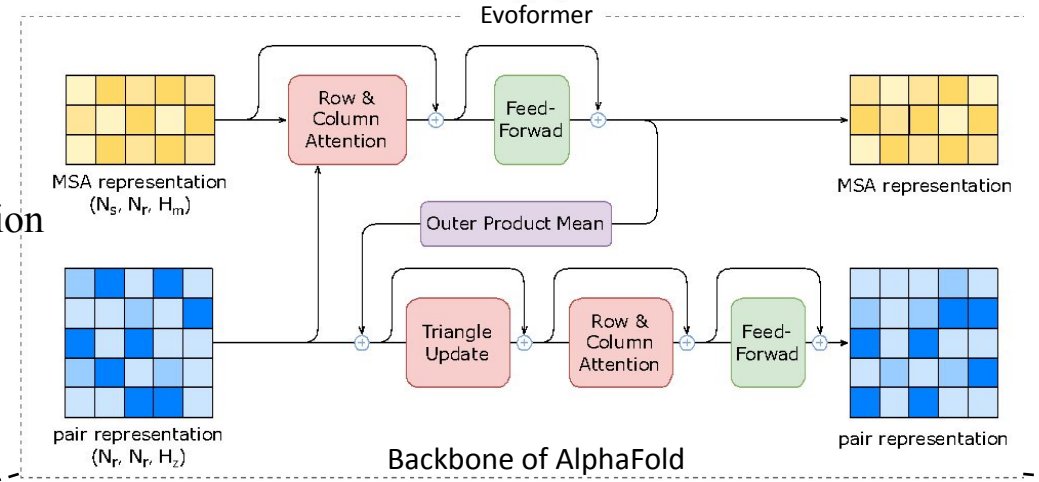
2) Memory Consumption — Storing all activation for attention context requires **several tens of GB**

3) Long Sequence Inference — Inference time can even reach **several hours**

Reducing AlphaFold Training Time (Drug Discovery) from 11 Days to 67 Hours

Challenges:

1) Time and costs
2) Memory Consumption
3) Long Sequence Inference



Backbone of AlphaFold

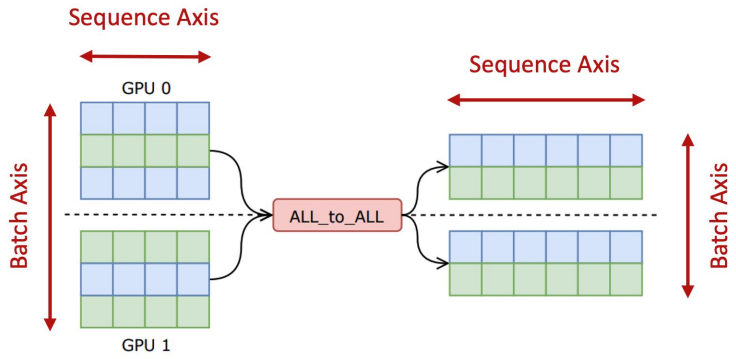
For Training

1) Overall training time: ***11 days -> 67 hours*** with significant *economic cost savings*
2) Scaling to 512 × A100 with ***6.02 PetaFLOPs***

For Inference

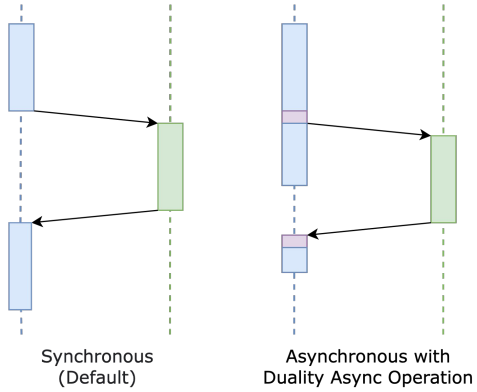
1) **7.5 ~ 9.5×** speedup for long sequences
2) makes it possible for inference over **extremely long sequences**

Parallel Evoformer



Dynamic Axial Parallelism

A novelty approach for efficient model parallelism



Duality Async Operation

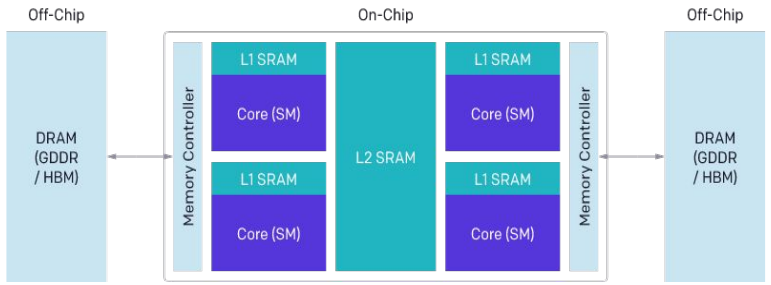
A PyTorch extension for communication optimization

Computational Optimizations

1) Merge GEMM in *Attention* and *Triangle Update Module*
2) JIT Fusion *(bias + dropout + add*, etc*)*
3) Optimized Kernel (LayerNorm, Fused Softmax)

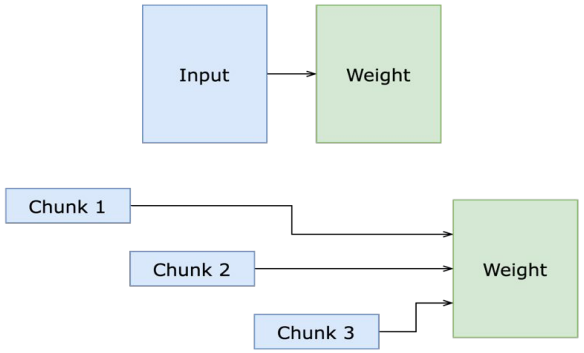
Singular-GPU Inference Sequence Exceeding 10,000, Covering 99.9999% of Proteins

Computation Optimization

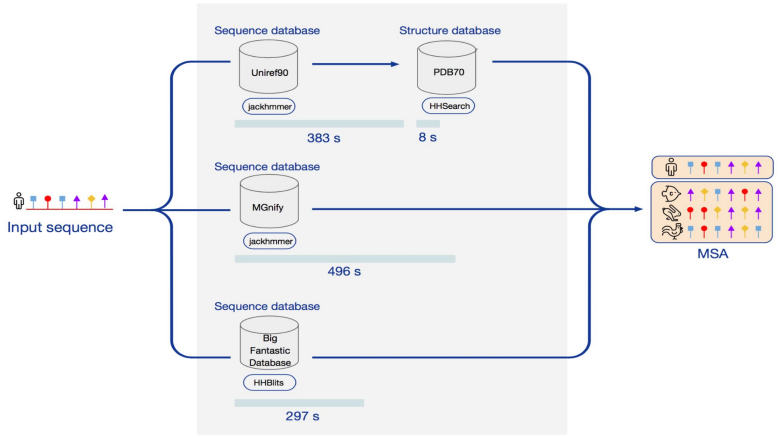


Optimized Kernels Based on Triton

Memory Optimization

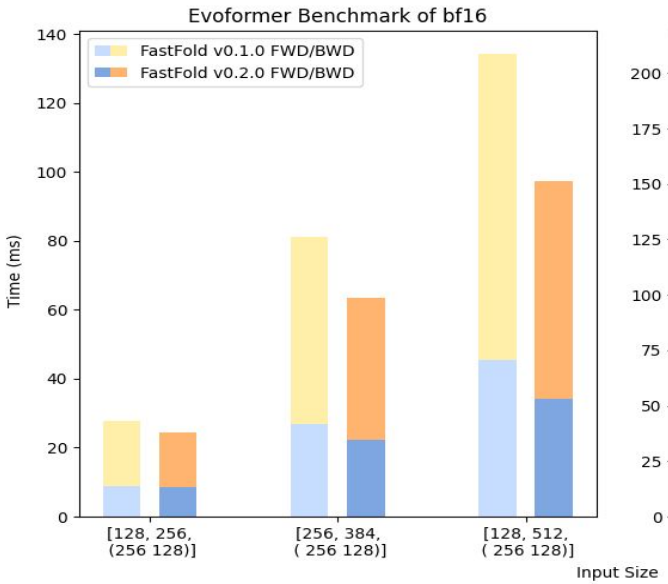


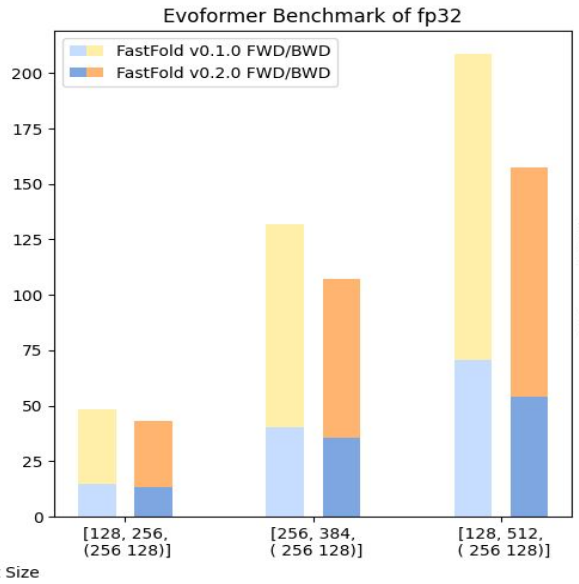
Chunk Optimization



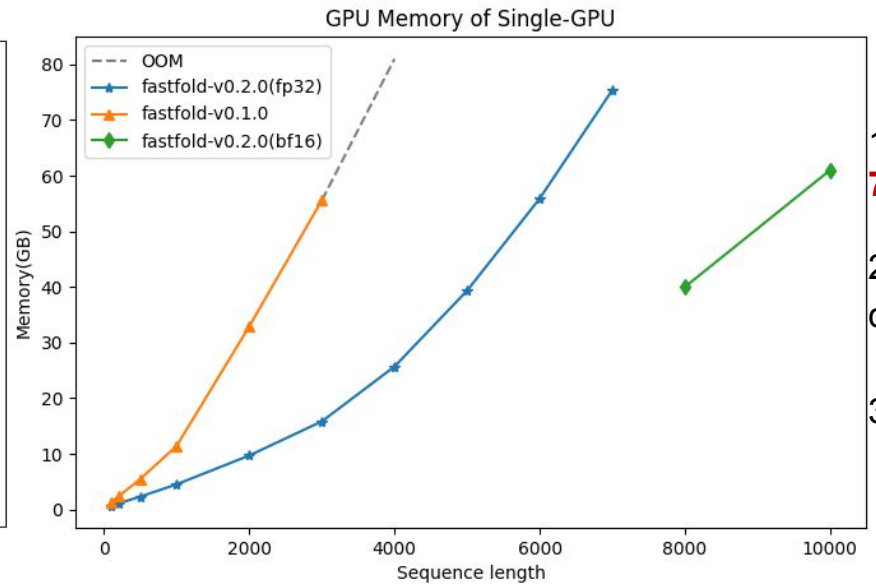
Parallel Acceleration of Pre-processing

Further Improvement





End-to-end Performance on NVIDIA A100



1) Reduce GPU memory by **75%**

2) Over **10K** sequence length covering **99.9999% of protein**

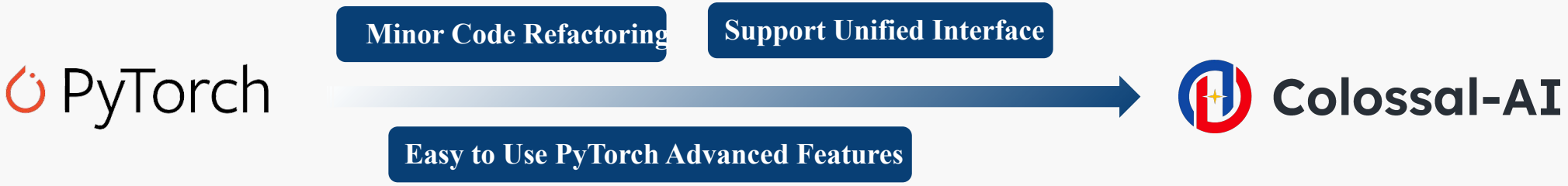
3) overall **5x** speedup

Image source: https://openai.com/blog/triton/

Ease of Use

- Provide unified interface for parallelism strategies in one config file
  - configure parallel size for data, tensor and pipeline parallelism
  - configure ZeRO hyperparameters
- Minor code refactoring
- Integrated other features such as gradient accumulation and mixed precision training

```python
import colossalai
from colossalai.utils import get_dataloader

colossalai.launch(
    config = my_config,
    rank = rank,
    world_size = world_size,
    backend = 'nccl',
    port = 29500,
    host = 'localhost'
)

model = ...

train_dataset = ...
train_dataloader = get_dataloader(dataset = dataset, shuffle = True)

optimizer = ...
criterion = ...

engine, train_dataloader, _, _ = colossalai.initialize(
    model = model,
    optimizer = optimizer,
    criterion = criterion,
    train_dataloader = train_dataloader
)

engine.train()
for epoch in range(NUM_EPOCHS):
  for data, label in train_dataloader:
    engine.zero_grad()
    output = engine(data)
    loss = engine.criterion(output, label)
    engine.backend(loss)
    engine.step()
```



Outline

1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Hands-on videos

- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Join Colossal-AI Slack!
Get this slides and Q&A

- The hands-on recordings and sample code are available on the project homepage https://github.com/hpcaitech/ColossalAI

Outline

1. Introduction of Big Model Era
2. N-Dim Parallelism in Big Model Era

Hands-on:
- Components Overview and environment setup
- Large Batch Training Optimization
- Multi-dimensional Parallelism
- Sequence Parallelism for BERT

3. Automatic Parallelism System
4. Efficient Memory System
5. Outstanding Performance & Use Cases

Hands-on:
- Auto-Parallelism with Colossal-AI
- Inference and Serving for Large Models
- AIGC: Acceleration of Stable Diffusion
- AI for Science: Acceleration of AlphaFold

Thanks for your time！

https://github.com/hpcaitech/ColossalAI

Join Colossal-AI Slack!
Get this slides and Q&A