Colossal-AI

Code & Tutorial

Slack Q&A

Scaling AI Models in Big Model Era

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 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)

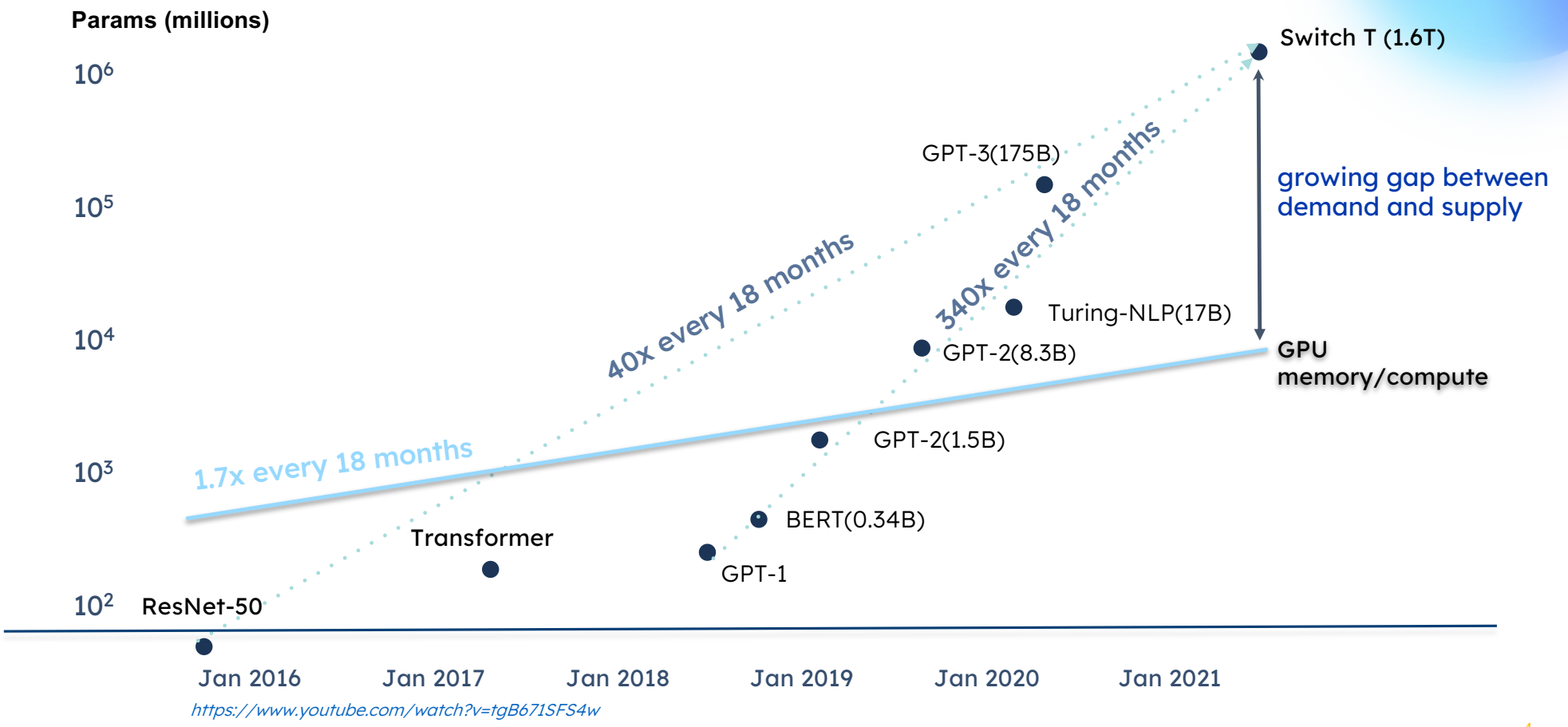
Berkeley NUS IEEE

Outline

1. Challenges & Opportunity in Big Model Era
2. N-Dim Parallelism System
3. Efficient Memory System
4. Outstanding Performance & Use Cases

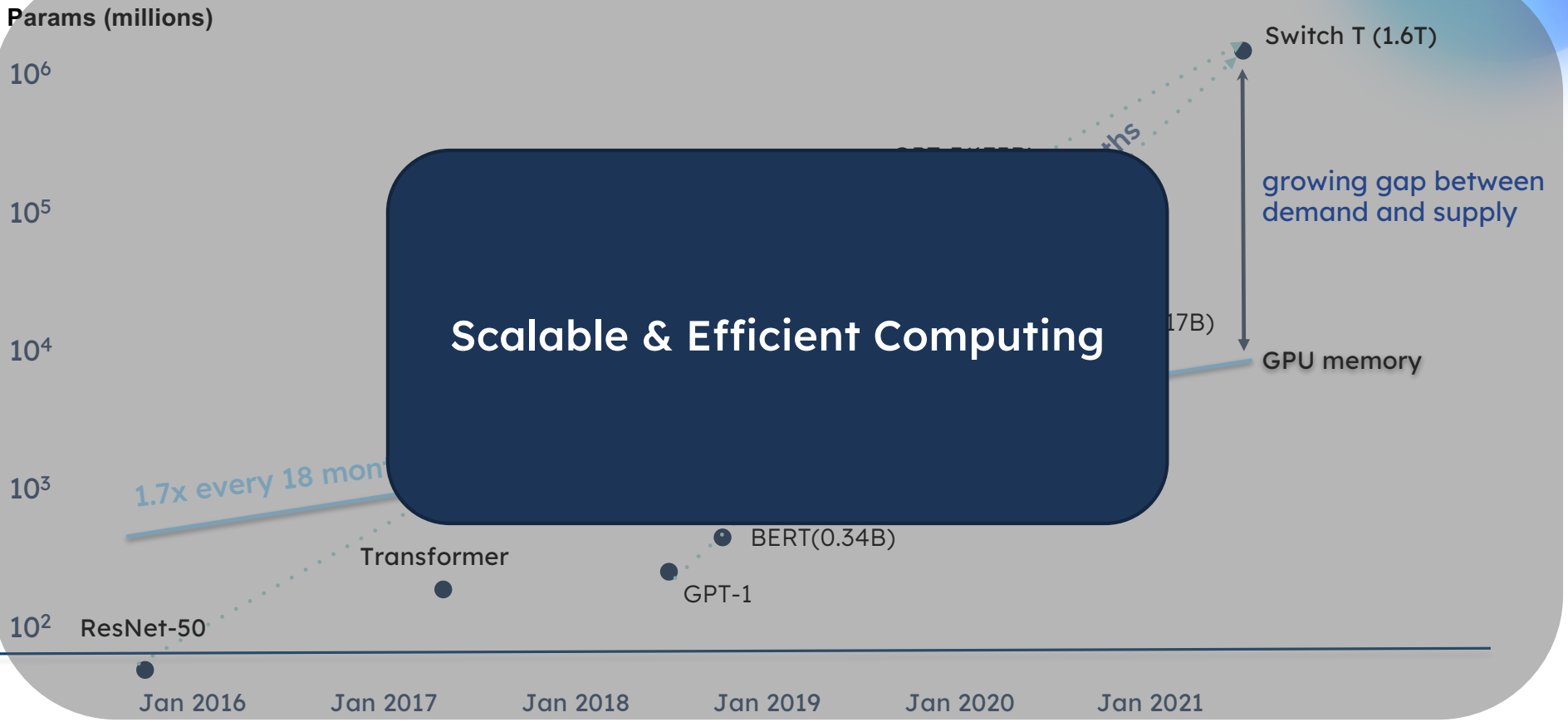
1 Challenges & Opportunity in Big Model Era

Problem



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

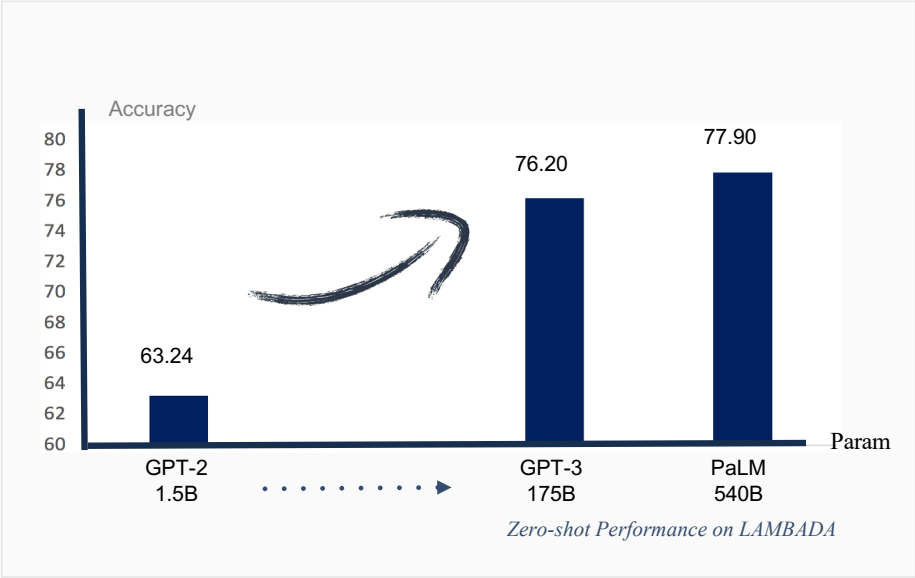
Problem



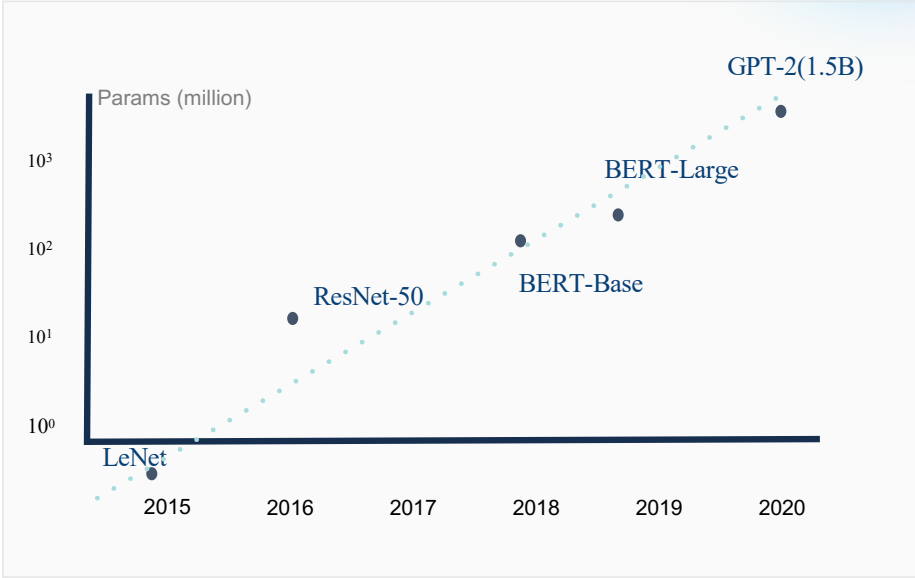
https://www.youtube.com/watch?v=tgB671SFS4w

Why Do We Believe in Large Models?

Larger Model: Better Performance

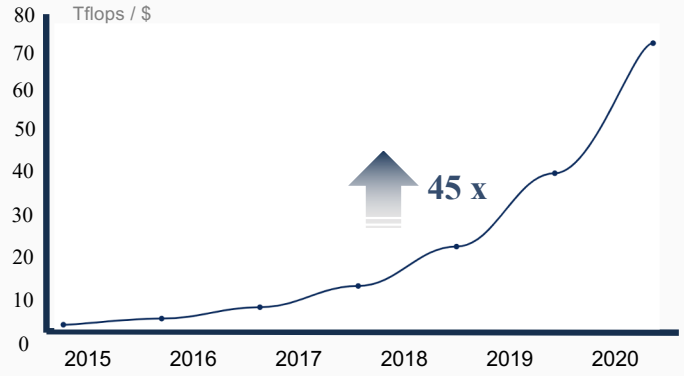


Models Used in SMEs are Growing Exponentially



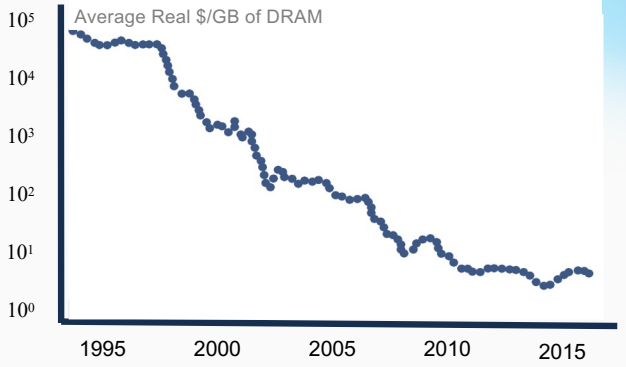
"Small Models" are Growing Exponentially

Why Do We Believe in Large Models?

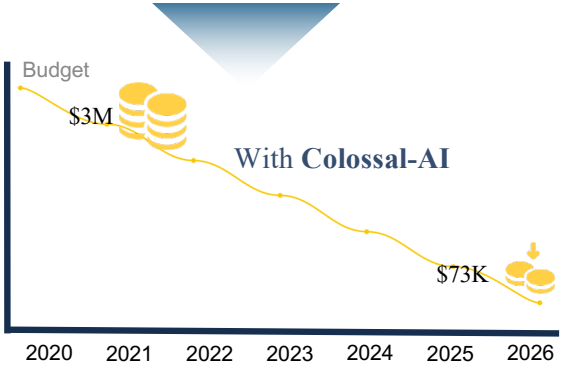


#Tflops per dollar is rising around 1.86x per year

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**

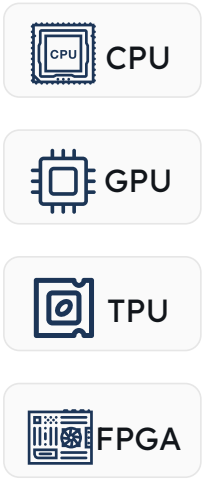
Deployment

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

Expensive Infrastructure and Systems

Colossal-AI = Performance + Efficiency + Cheapness

Hardware → Colossal-AI → Framework



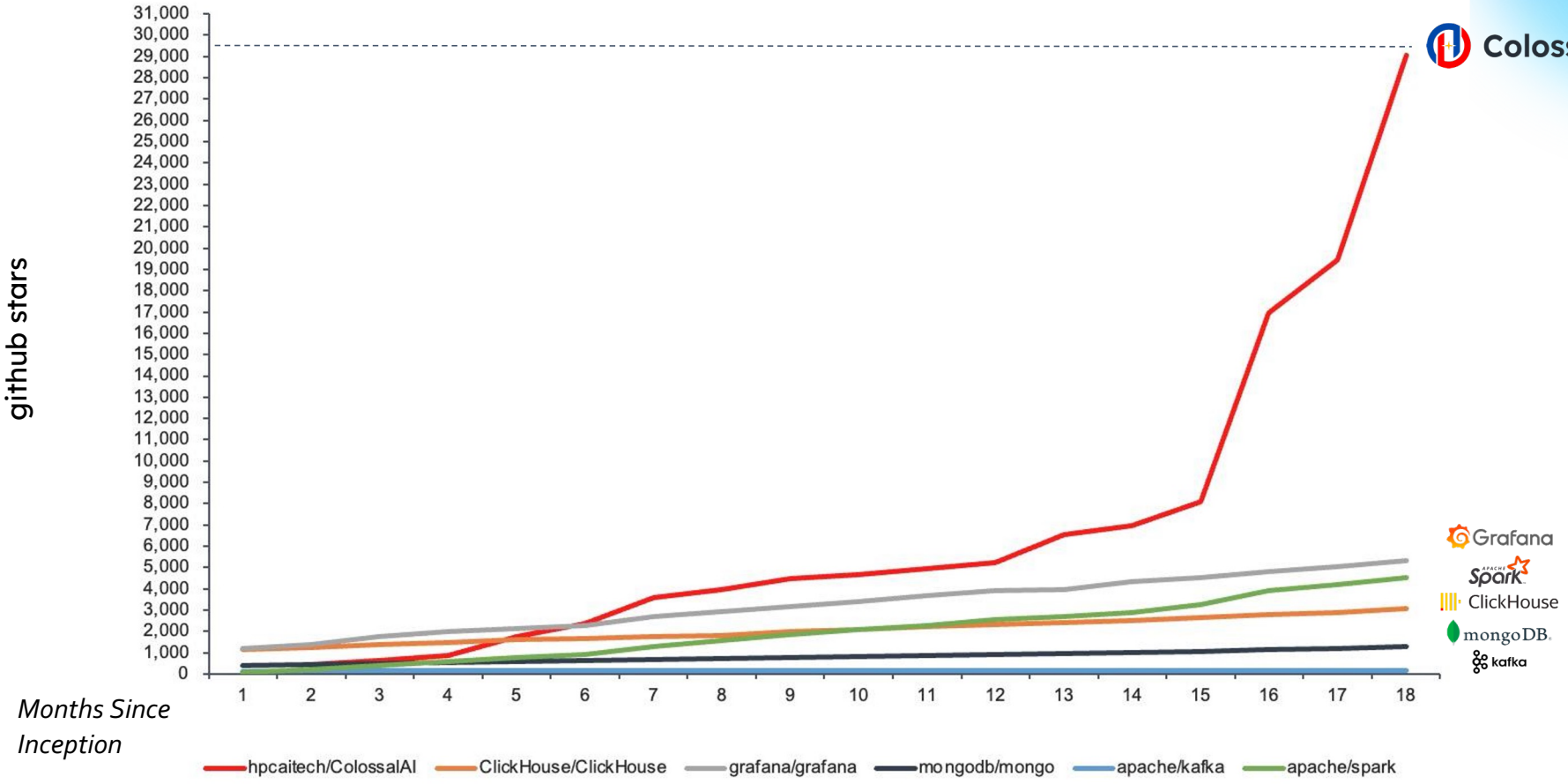
Layer 3: Low Latency Inference System

Layer 2: N-Dim Parallelism System

Layer 1: Efficient Memory System

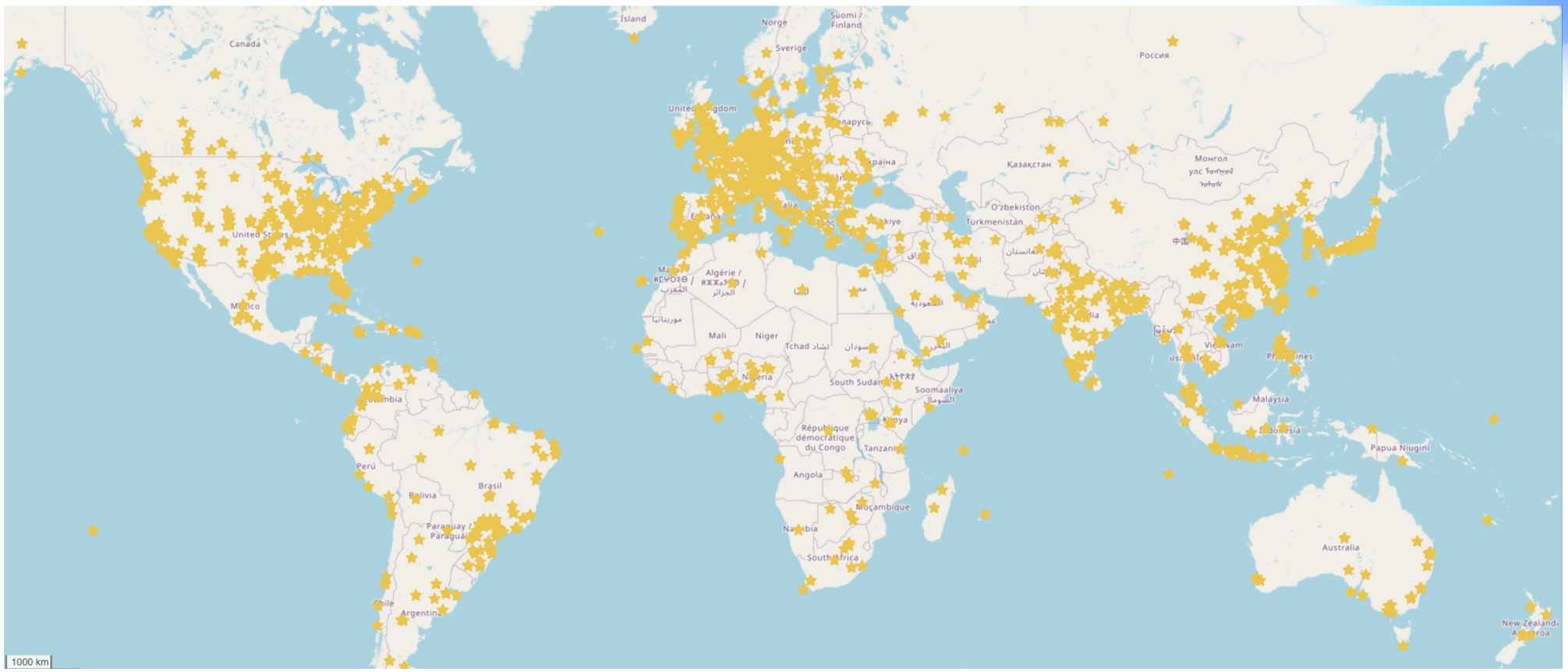
- Maximize computational efficiency
- Minimize system running time
- Minimize communication
- Minimize code refactoring
- Dynamic adaptive scaling
- Reduce memory footprint

Fast Growing Open Source Community

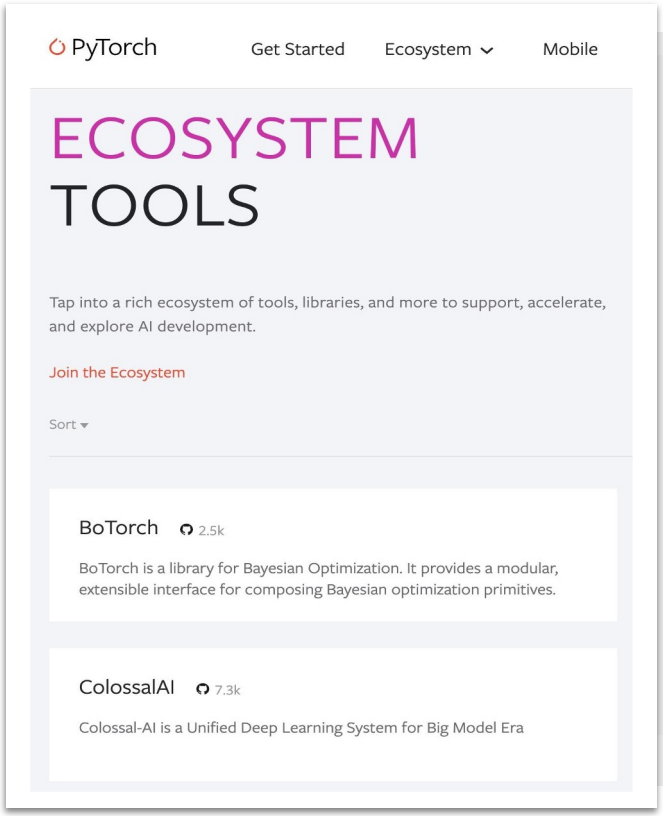


Colossal-AI is public and available at https://github.com/hpcaitech/ColossalAI

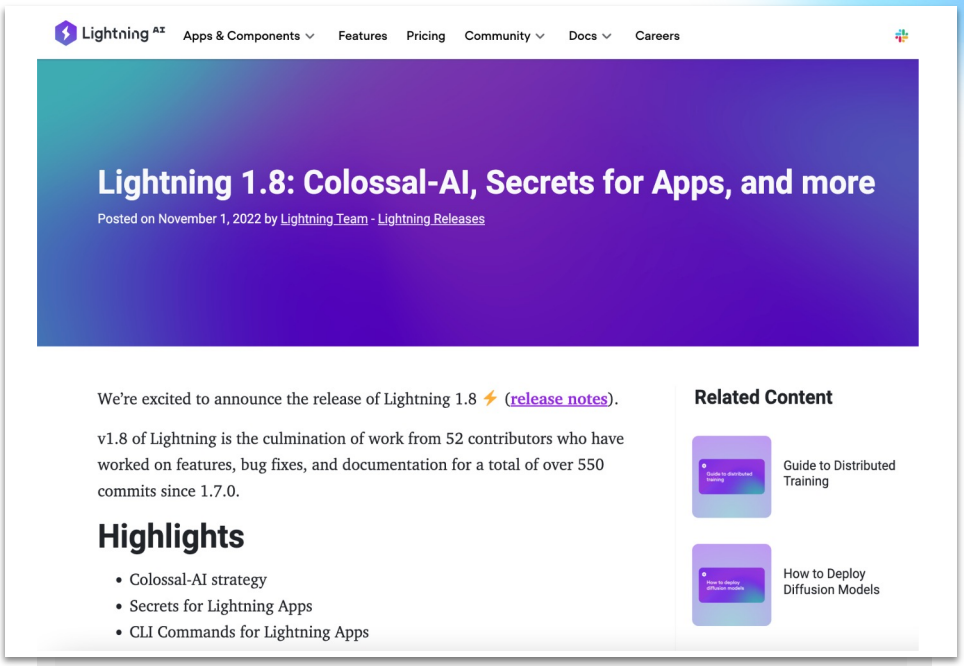
Stargazers of Colossal-AI are All Over the World



Users from Global AI Ecosystems

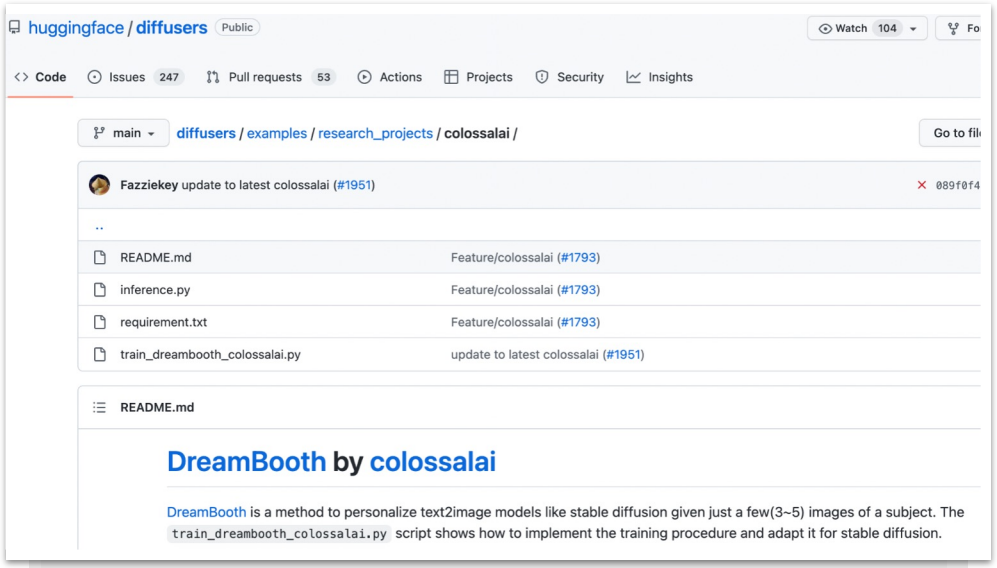


PyTorch Users

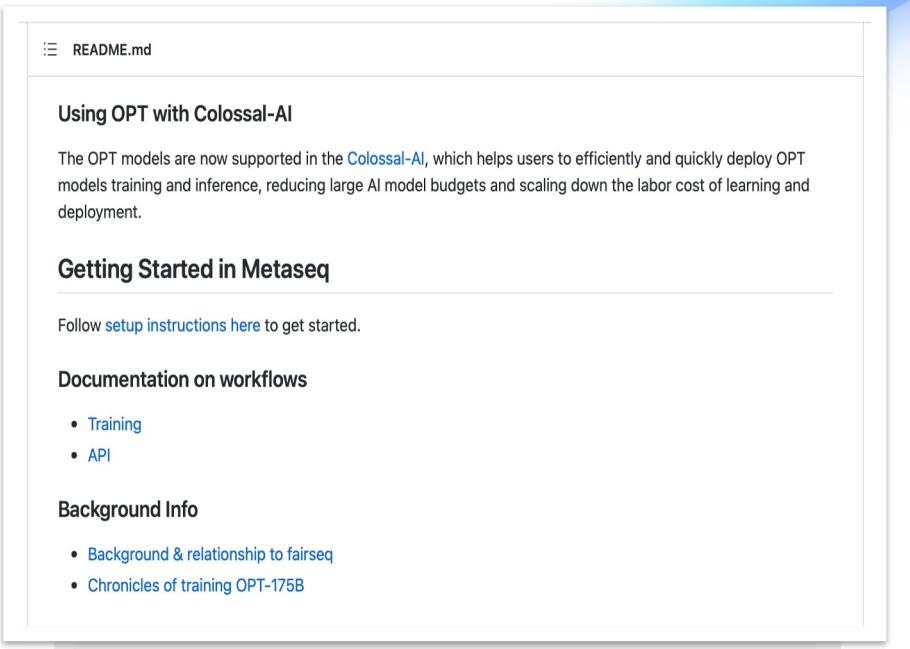


Lightning AI Users

Users from Global AI Ecosystems



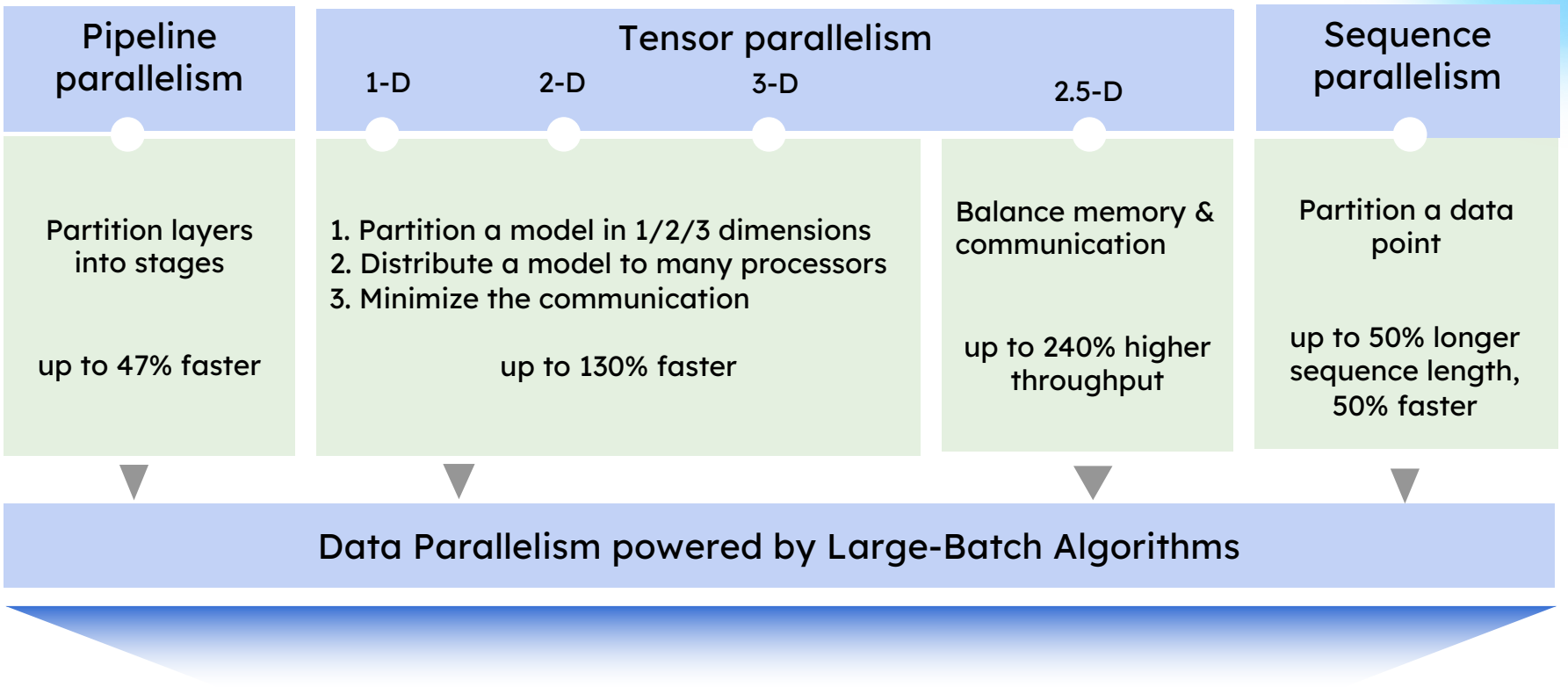
Hugging Face Users



Facebook OPT Users

2 N-Dim Parallelism System

Overview of N-Dim Parallelism System

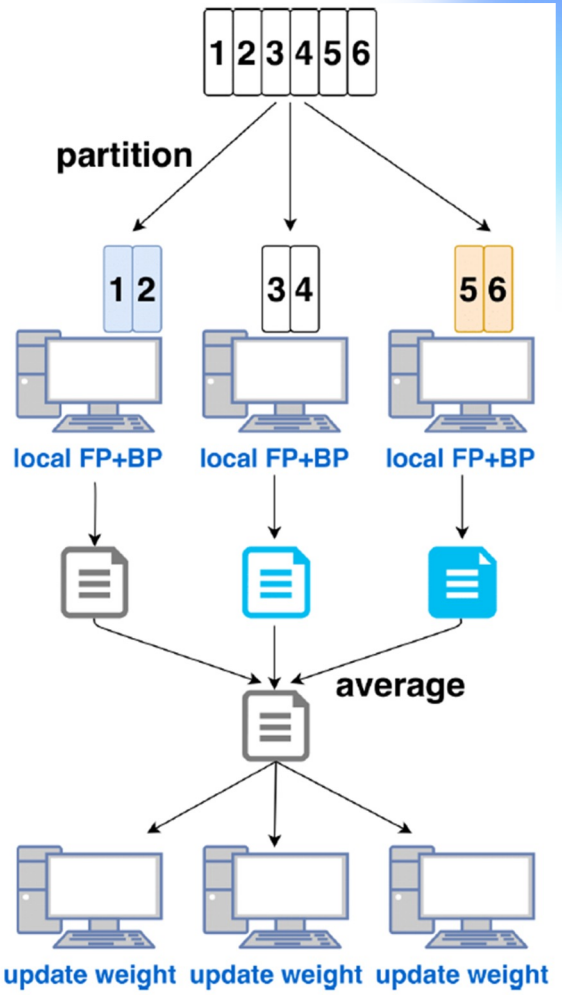


All in N-Dim Parallelism System from Colossal-AI

Data Parallelism

Batch Size	Epochs	Iterations
512	100	250,000
1024	100	125,000
2048	100	62,500
4096	100	31,250
8192	100	15,625
...	...	...
1,280,000	100	100

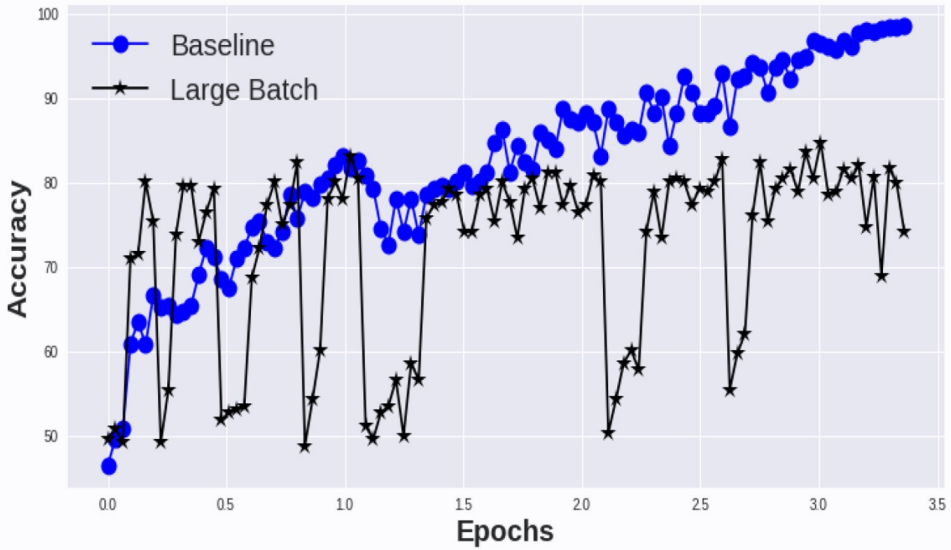
- Larger batch size, faster training



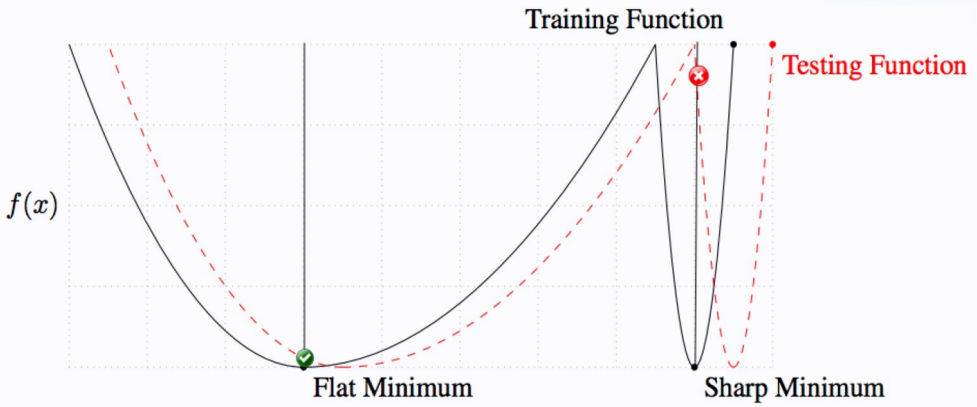
source: NUS CS5260

Challenges of Large Scale Parallel Data Processing

- **Reduced accuracy**



- **sharp minimum problem**



batch size upper limit: 8K

Scalable Large-scale Optimizers

LARS/LAMB

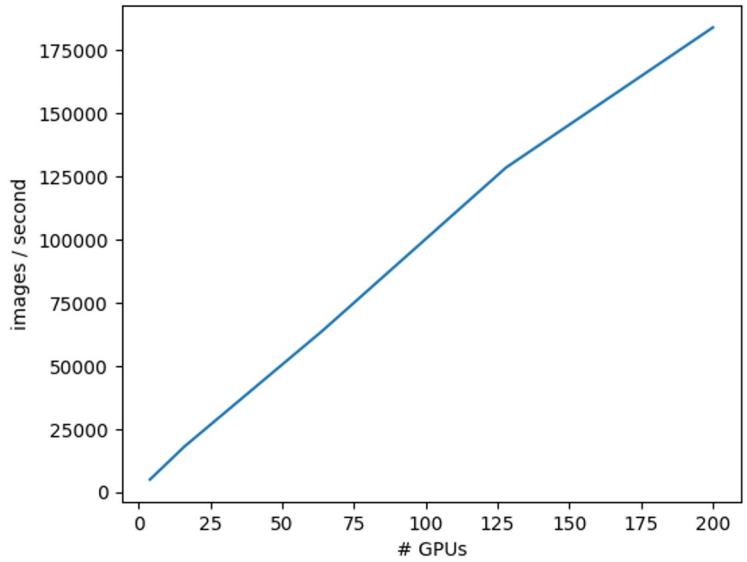
source: NUS CS5260

Scalable Large-scale Optimizers: LARS/LAMB

Benefits

- Maximize the use of GPU resources and achieve near linear acceleration with guaranteed convergence.

# GPU	Batch size	300 epochs (hour)
1	128	73
4	512	21
16	2048	5.88
64	8192	1.67
128	16k	0.83
200	32k	0.68

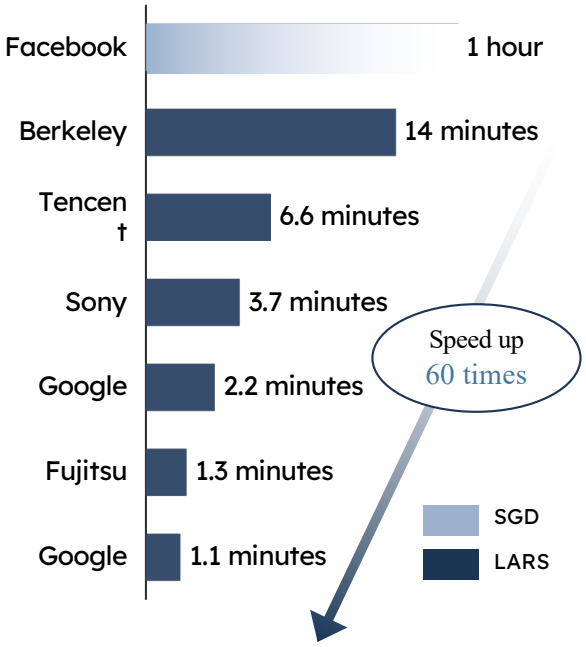


ViT-B/32 ImageNet-1K

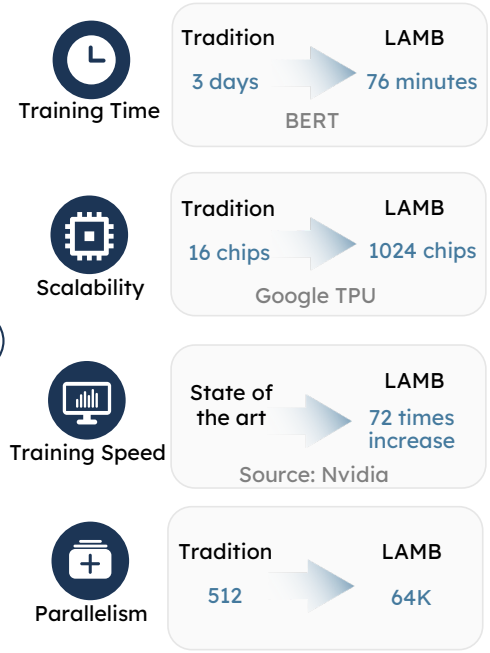
LARS/LAMB's Achievement & Industry Impact

The world's leading deep learning training optimizer

LARS
World record for ImageNet training



LAMB
A leap breakthrough in performance



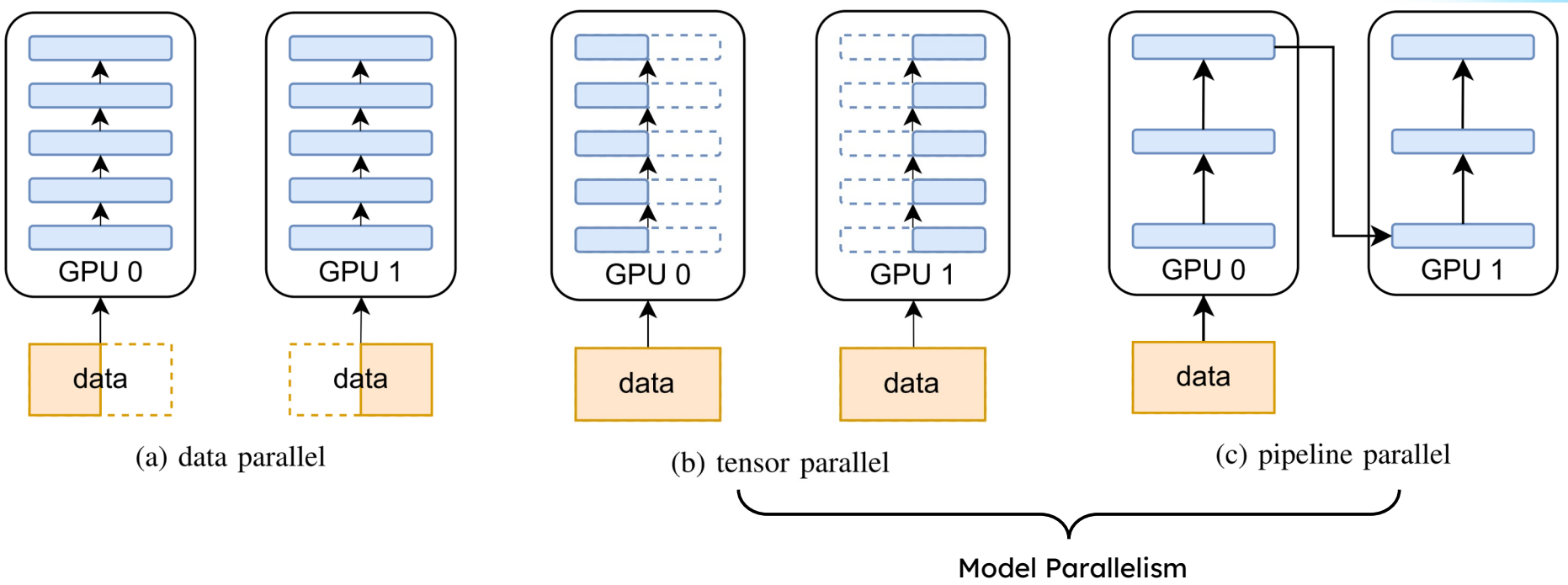
Widely used in the industry

SimCLR | SEER | DeepSpeed

BYOL | Expand AI Scale | Industry AI Training Benchmark

https://github.com/hpcaitech/ColossalAI

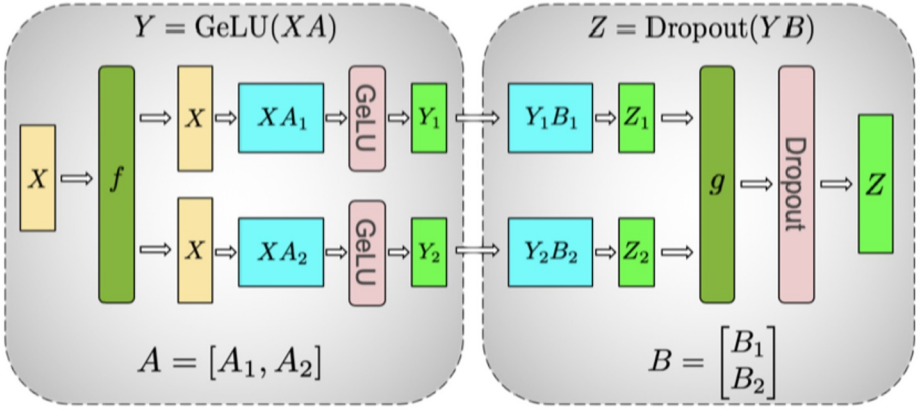
Model Parallelism



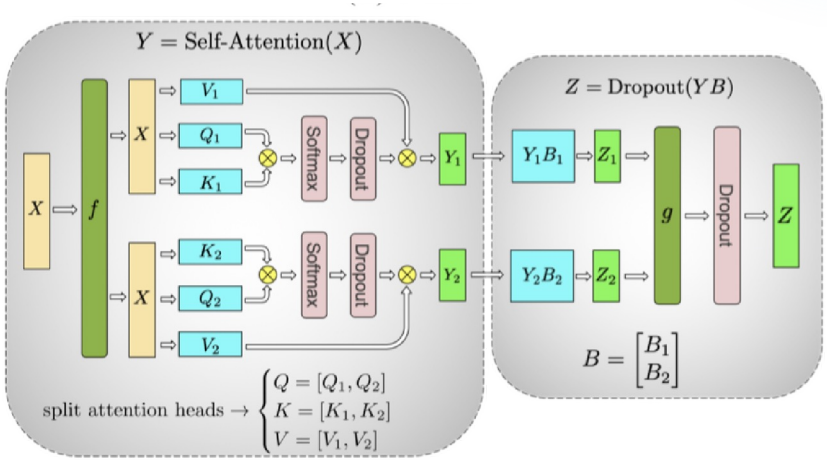
- **Existing parallelism for distributed training**

Existing Solutions Regarding Model Parallelism

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



(a) MLP

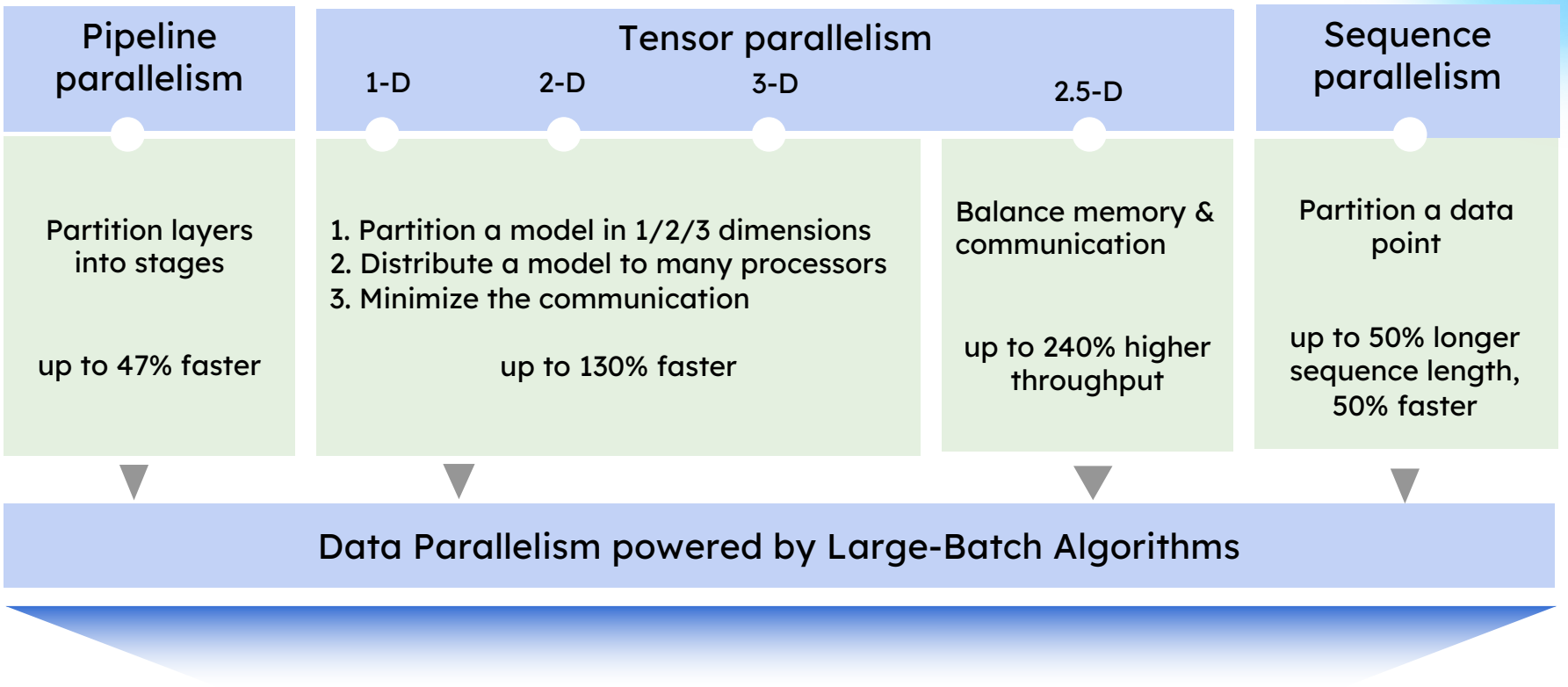


(b) self attention

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

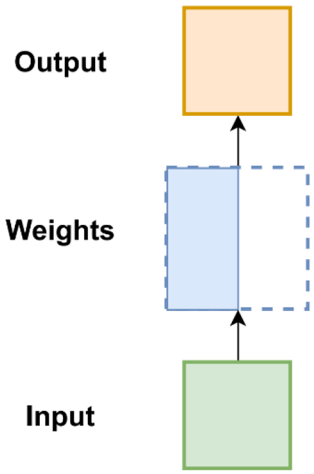
Shoeybi, Mohammad, et al. "Megatron-lm: Training multi-billion parameter language models using model parallelism." arXiv preprint arXiv:1909.08053 (2019).

Overview of N-Dim Parallelism System

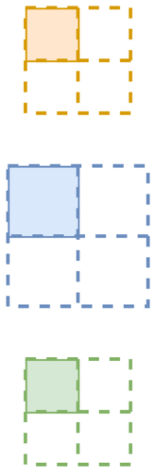


All in N-Dim Parallelism System from Colossal-AI

Tensor Parallelism



(a) 1D tensor parallelism



(b) 2D tensor parallelism

(c) 2.5D tensor parallelism

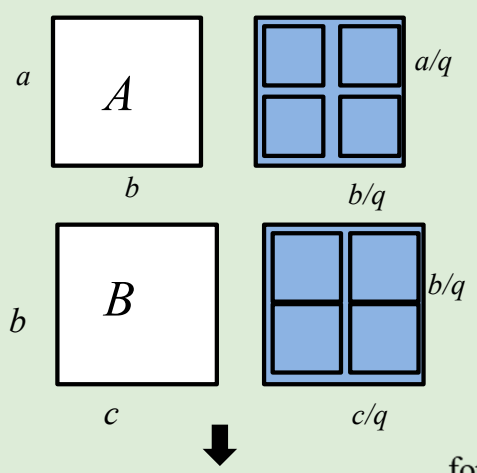
(d) 3D tensor parallelism

- Tensor parallel illustration

2-D/2.5-D Tensor Parallelism

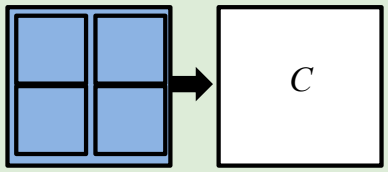
2-D Tensor Parallelism

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

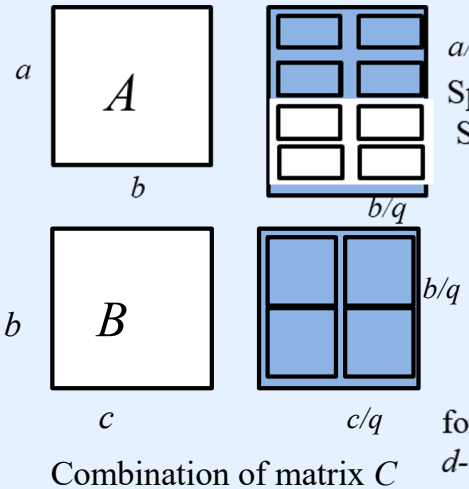
Combination of matrix C



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

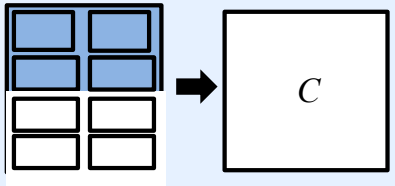
2.5-D Tensor Parallelism

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

Combination of matrix C



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

2-D/2.5-D Tensor Parallelism

Strong scaling setting

(the number of processors is increased while the problem size remains constant)

Parallelization	#GPU	forward time/batch	backward time/batch	throughput	inference
Megatron-LM	4	0.1225	0.4749	1.6739	8.1633
	16	0.1143	0.4293	1.8396	8.7489
	64	0.1195	0.5306	1.5382	8.3682
2-D Tensor Parallelism	4	0.1676	0.5019	1.4937	5.9666
	16	0.2099	0.6159	1.2109	4.7642
	64	0.1329	0.3986	1.8815	7.5245
2.5-D Tensor Parallelism	4	0.1666	0.5014	1.4970	6.0024
	16	0.1444	0.4343	1.7280	6.9252
	64	**0.0869**	**0.2636**	**2.8531**	**11.5075**

2-D/2.5-D Tensor Parallelism

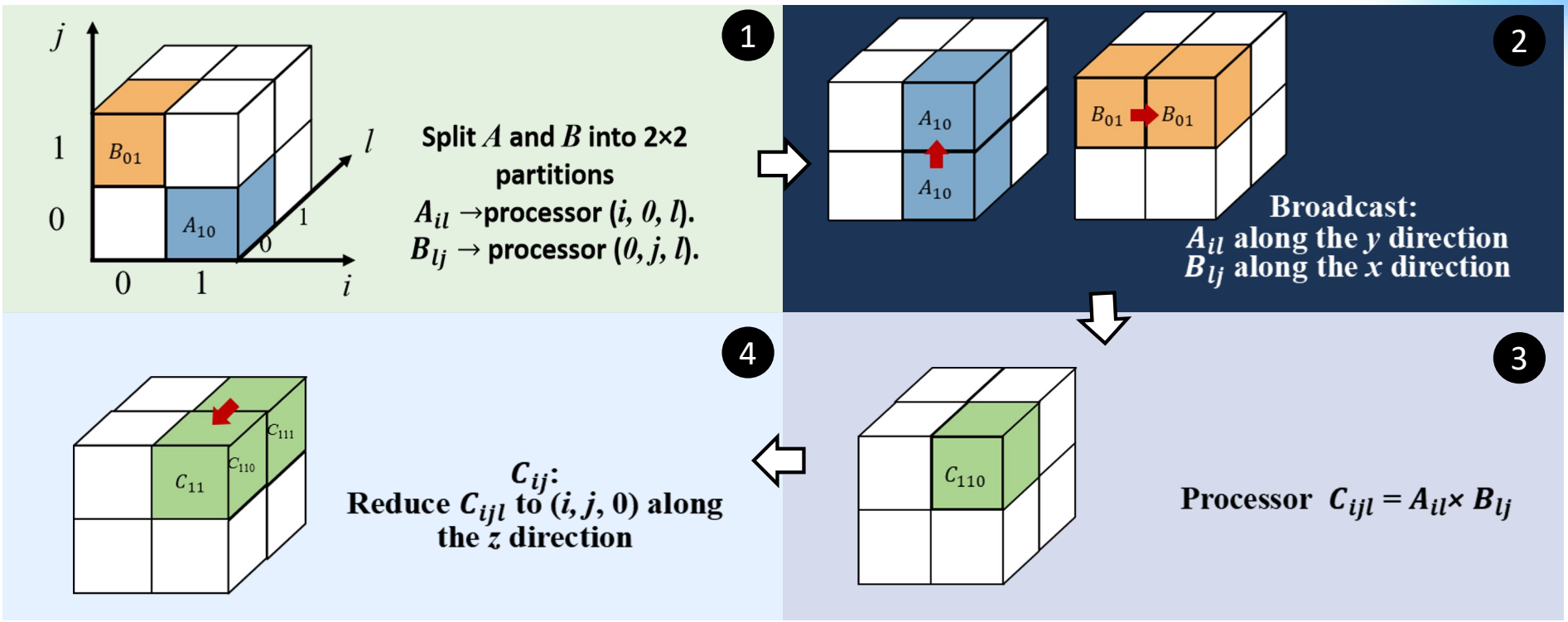
Weak scaling setting

(both the number of processors and the problem size are increased)

Parallelization	#GPU	forward time/batch	backward time/batch	throughput	inference
Megatron-LM	4	0.0793	0.2613	2.9360	12.6103
	16	0.2081	0.5149	1.3831	4.8054
	64	0.4638	1.0963	0.6410	2.1561
2-D Tensor Parallelism	4	0.0827	0.2445	3.0562	12.0919
	16	0.1829	0.5458	1.3723	5.4675
	64	0.1962	0.5964	1.2617	5.0968
2.5-D Tensor Parallelism	4	0.0867	0.2557	2.9206	11.5340
	16	0.1177	0.3553	2.1142	8.4962
	64	**0.1155**	**0.3468**	**2.1631**	**8.6580**

3-D Tensor Parallelism

- 3-D matrix multiplication example： $C = AB$ on a 2 × 2 × 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

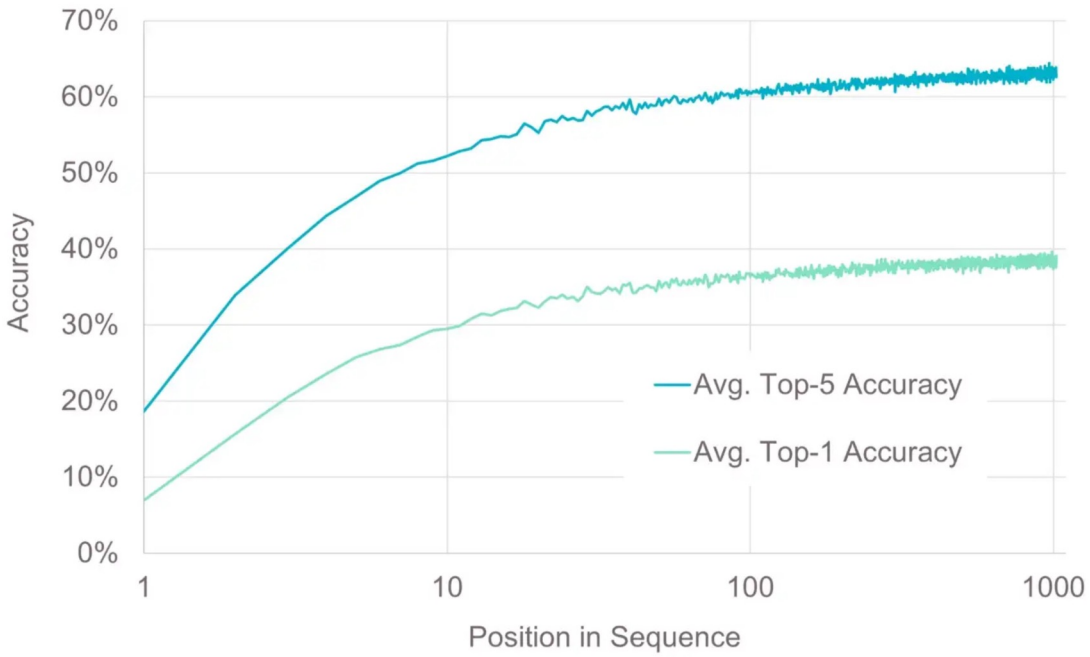
Efficiency:

	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	$O\left(\frac{1}{P}\right)$	$O\left(\frac{1}{P^{\frac{2}{3}}}\right)$	$O\left(\frac{1}{P^{\frac{2}{3}}}\right)$	$O\left(\frac{1}{P^{\frac{2}{3}}}\right)$	O(logP)

- P: number of processors

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

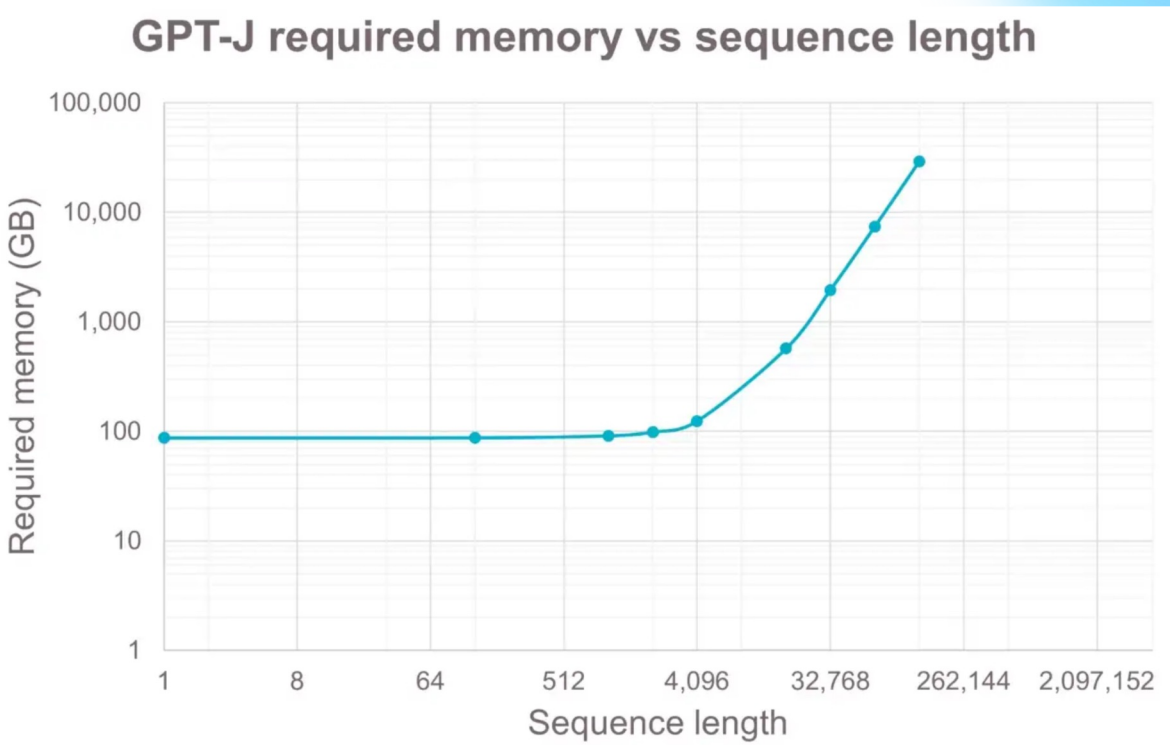
Memory Bottleneck of Sequence Length

- Model
  - Weights
  - Gradients
- Optimizer
  - States

} fixed

- Input Data
  - Activation

} vary



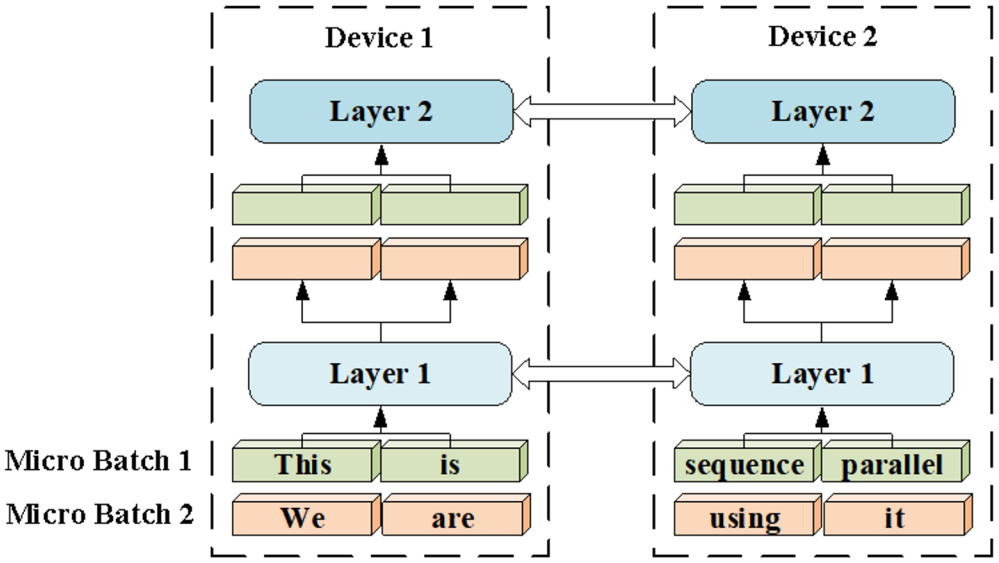
- Transformer (Attention) has quadratic complexity at memory.
- When data dimension is large, it can become the memory bottleneck.

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

Sequence Parallelism

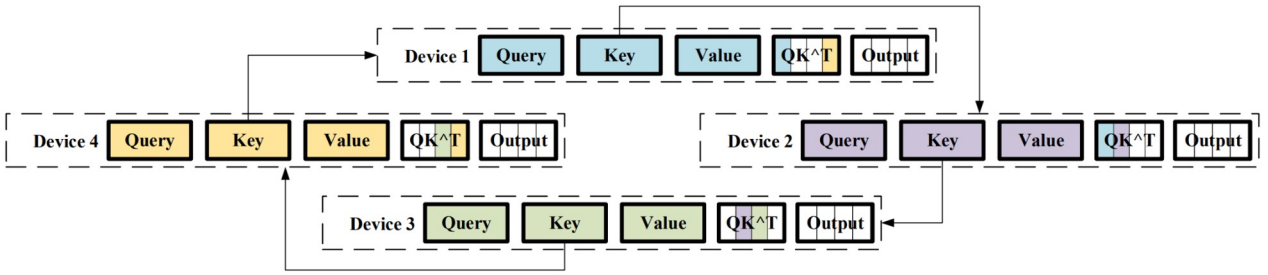
Why Sequence Parallelism?

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

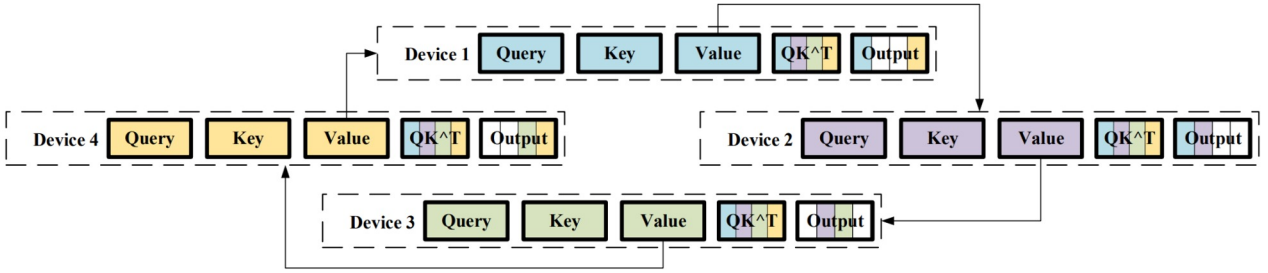


- Parallelize in the sequence dimension -> reduce memory consumption by input data and activation
- Model weights are replicated across devices

Ring Self-Attention



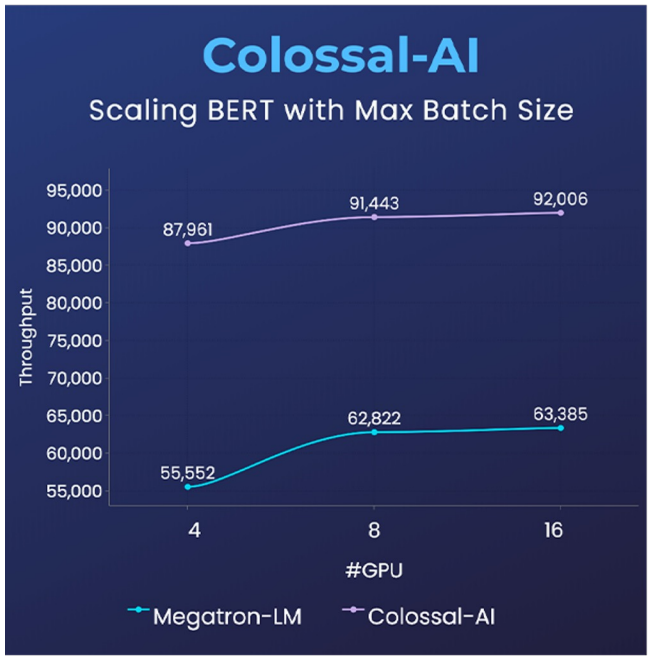
(a) Transmitting key embeddings among devices to calculate attention scores

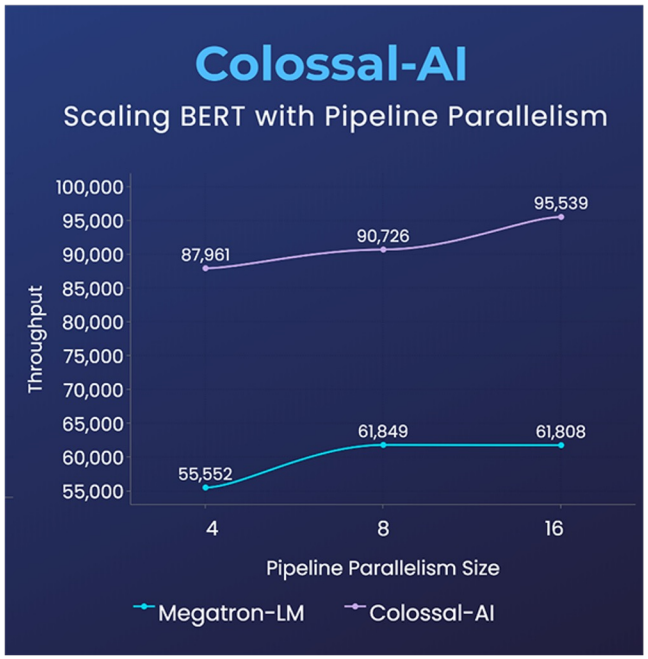


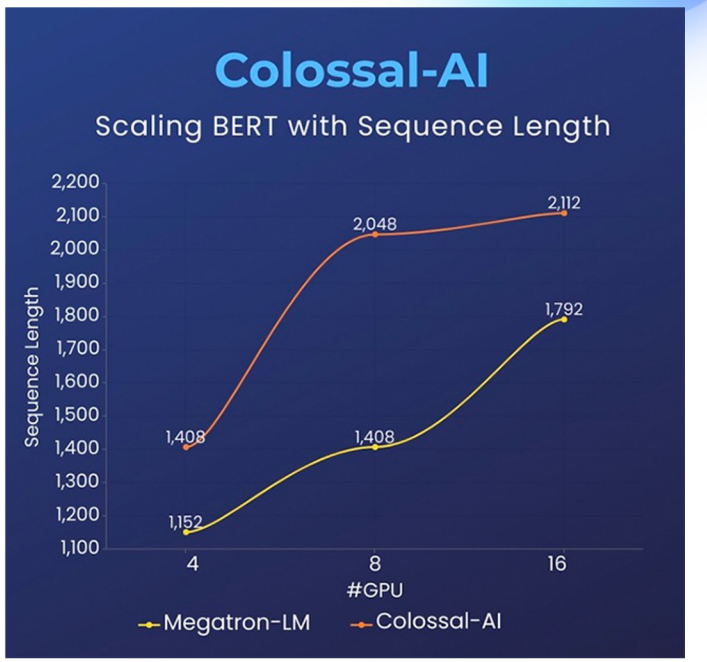
(b) Transmitting value embeddings among devices to calculate the output of attention layers

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

Training Benchmark







- 1.55x faster training, or 50% longer sequence length vs Megatron-LM.

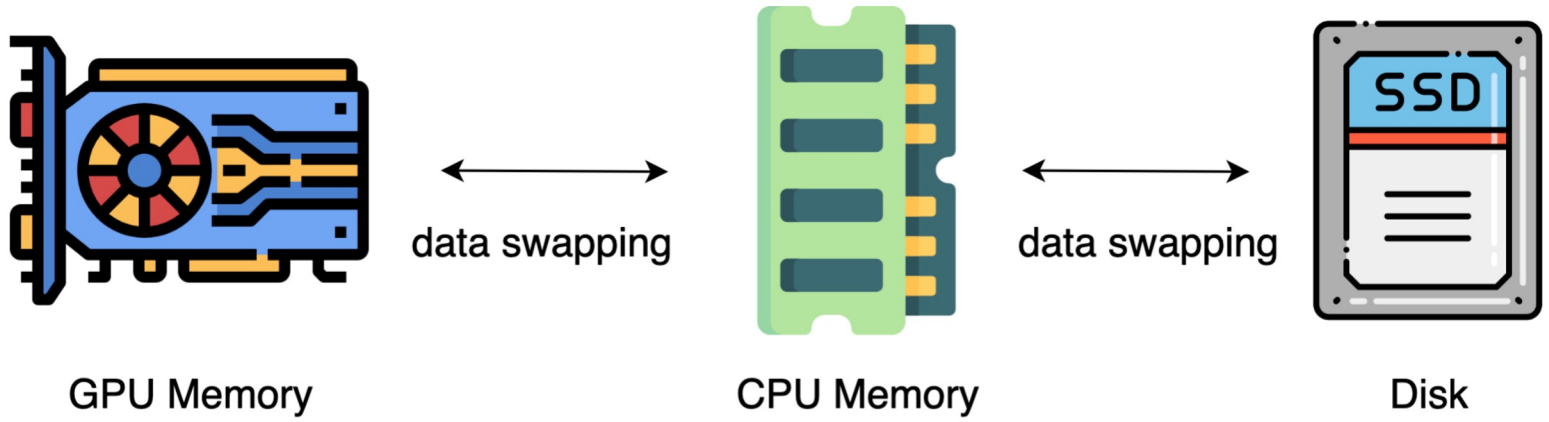
Auto-Parallelism

- 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
...
```

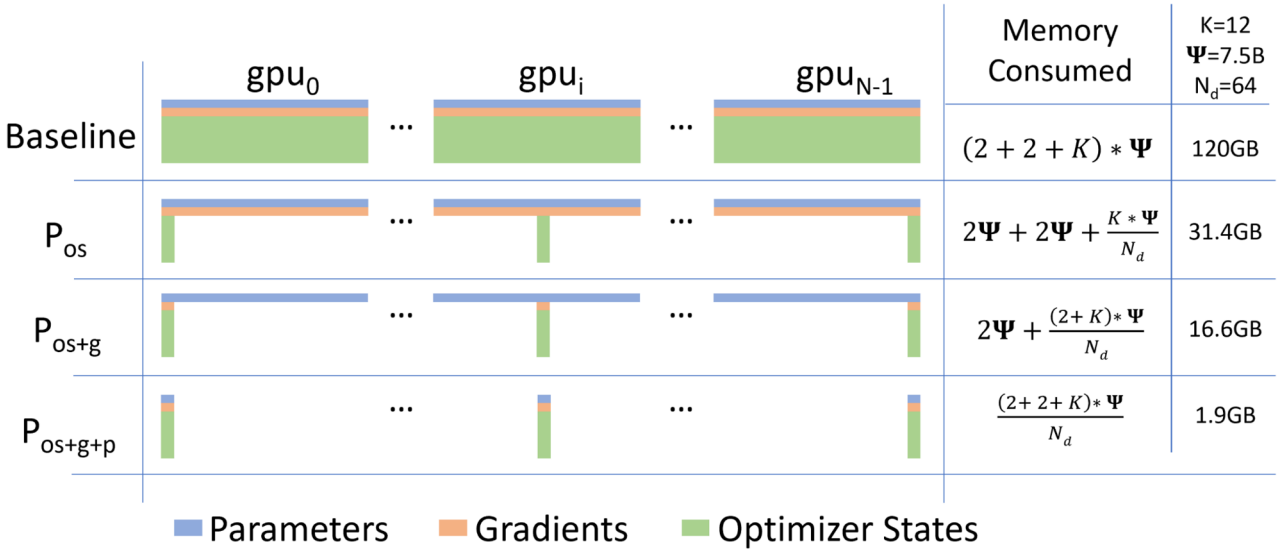
3 Efficient Memory System

Data Movement on a Heterogeneous System



- Heterogeneous system illustration

Existing Solution: ZeRO (Zero Redundancy Optimizer)

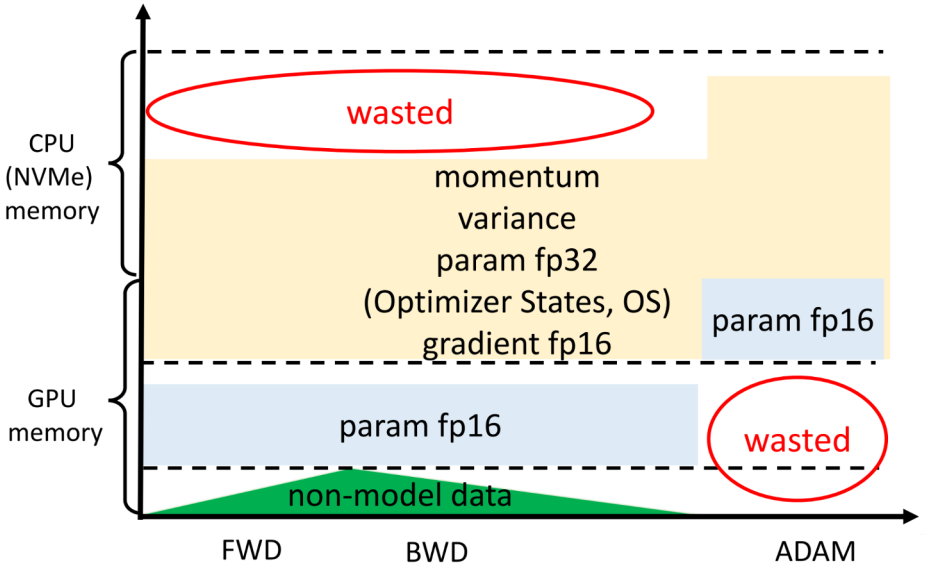


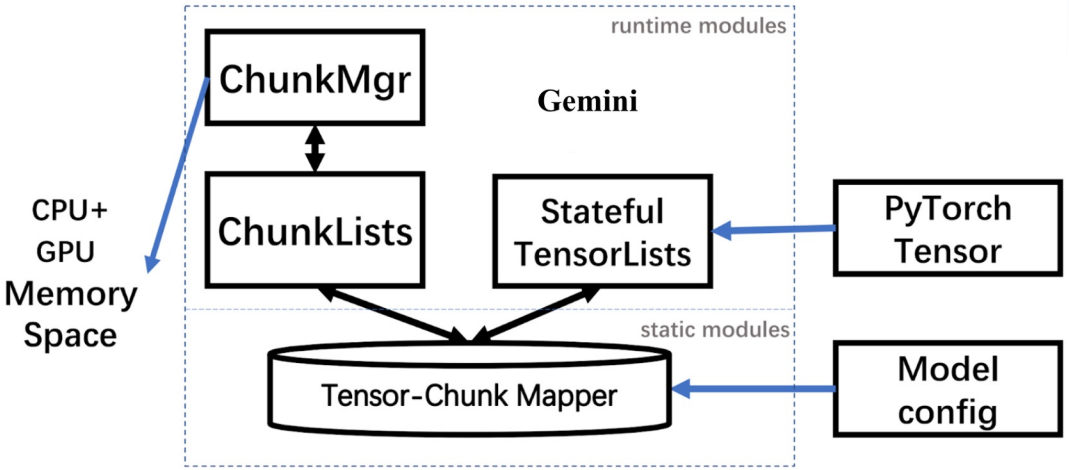
- Partition the model states (weights, gradients, and optimizer states) across available devices.
- Offload GPU memory to both CPU and NVMe memory for huge memory savings.

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.

Our Solution: Heterogeneous Memory Management



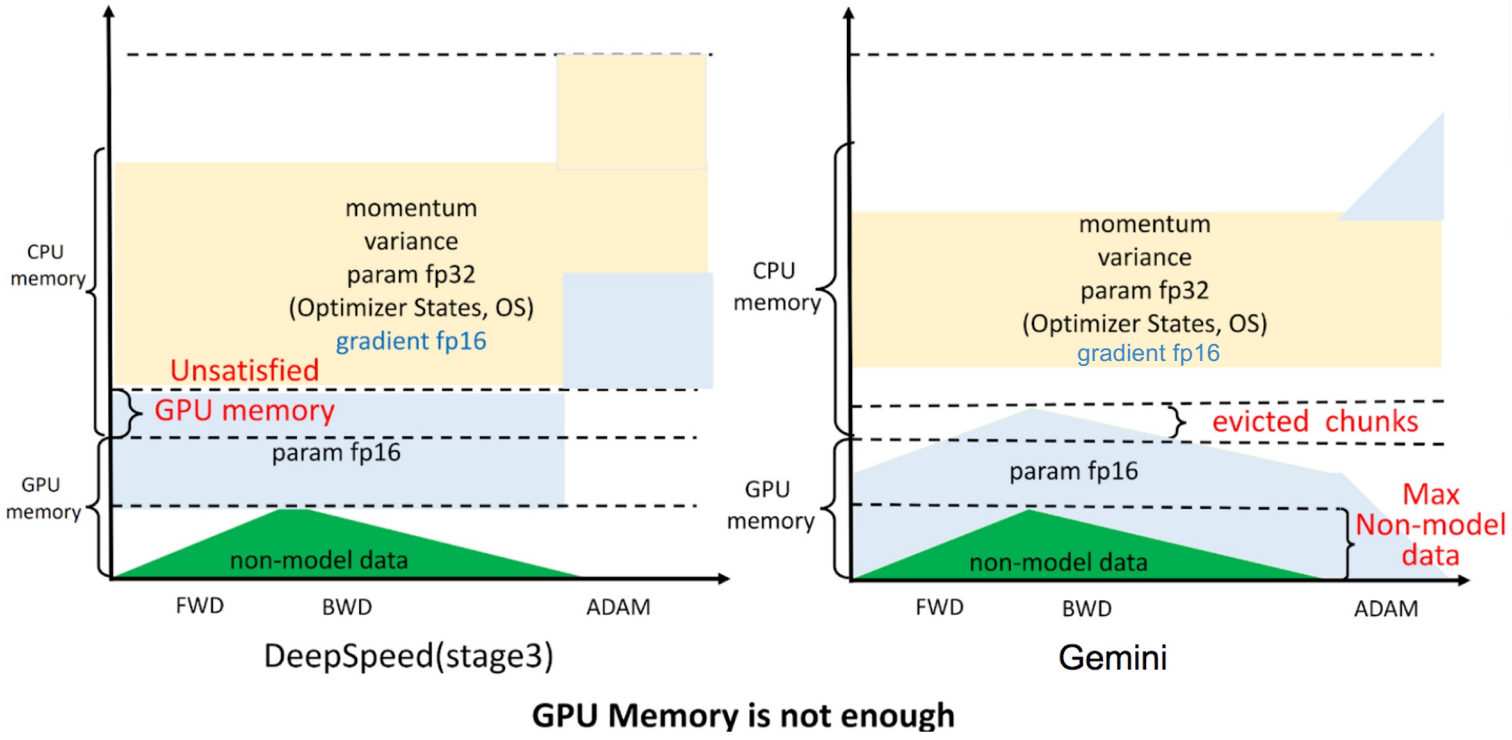


- The static memory partition in DeepSpeed
- The Gemini architecture of Colossal-AI

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.

Fang, Jiarui, et al. "Parallel Training of Pre-Trained Models via Chunk-Based Dynamic Memory Management." IEEE Transactions on Parallel and Distributed Systems 34.1 (2022): 304-315.

Colossal-AI: Heterogeneous Memory Management

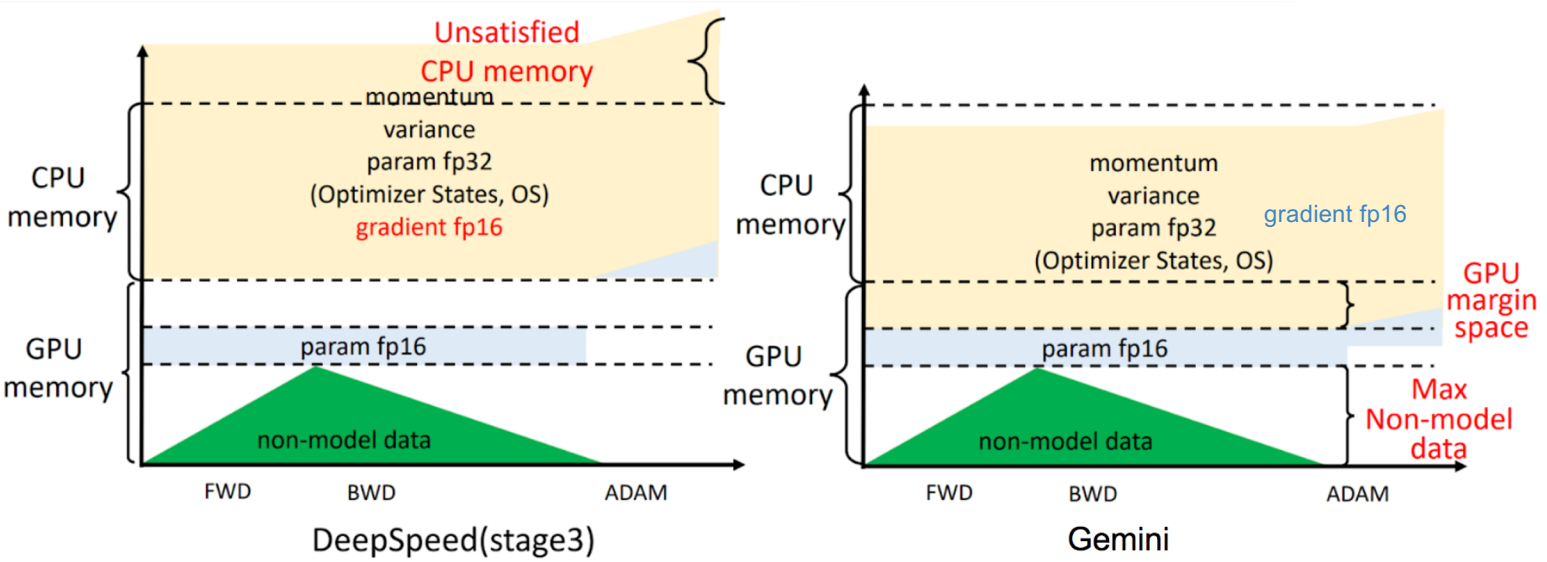


GPU Memory is not enough

- **Colossal-AI can handle situations where DeepSpeed cannot**

Fang, Jiarui, et al. "Parallel Training of Pre-Trained Models via Chunk-Based Dynamic Memory Management." IEEE Transactions on Parallel and Distributed Systems 34.1 (2022): 304-315.

Colossal-AI: Heterogeneous Memory Management

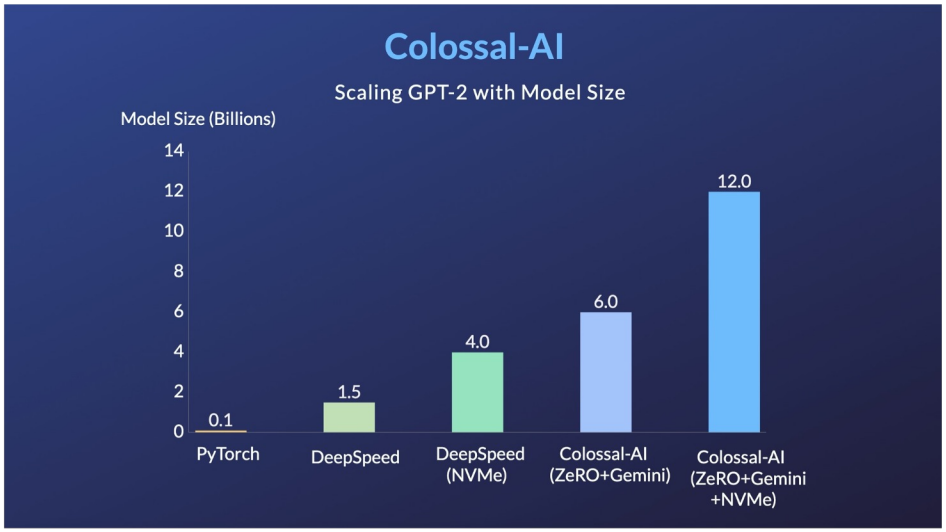


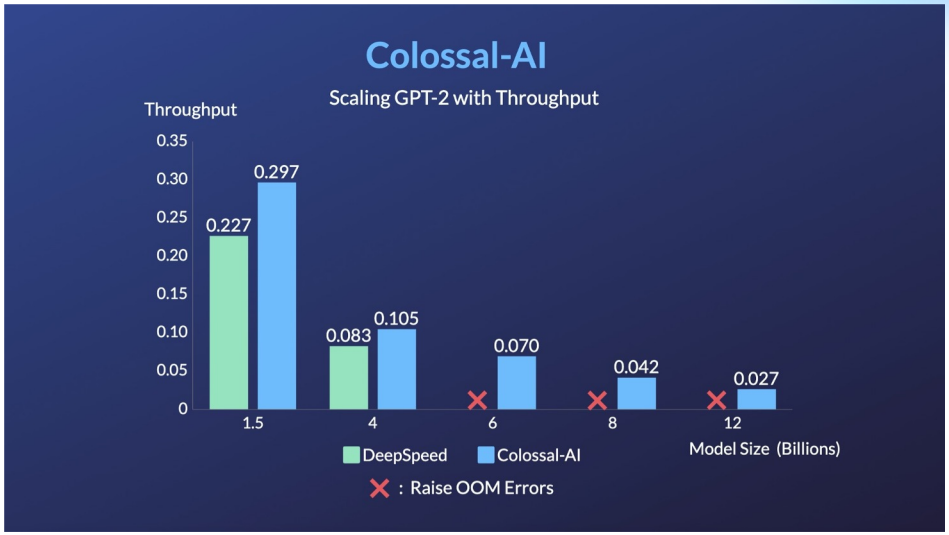
CPU Memory is not enough

- **Colossal-AI can handle situations where DeepSpeed cannot**

Fang, Jiarui, et al. "Parallel Training of Pre-Trained Models via Chunk-Based Dynamic Memory Management." IEEE Transactions on Parallel and Distributed Systems 34.1 (2022): 304-315.

Training Benchmark

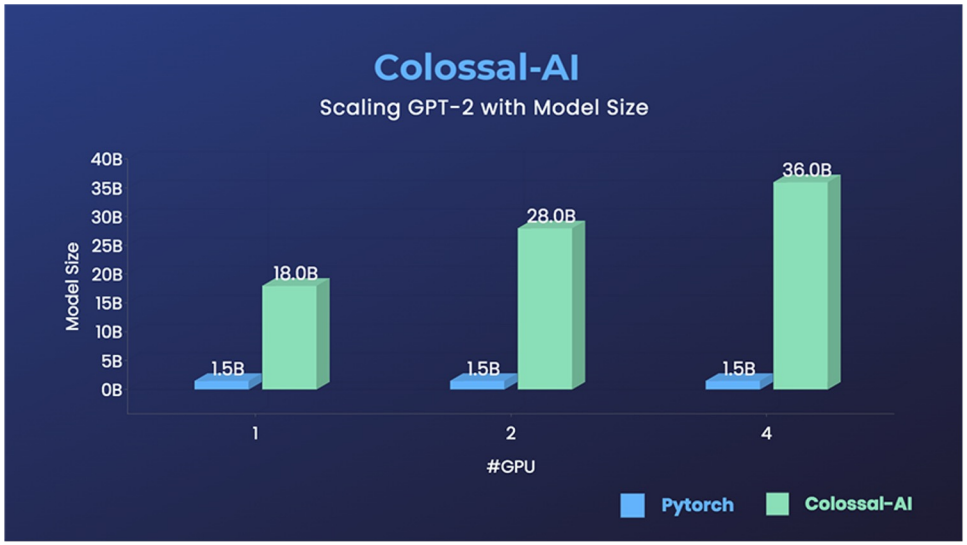


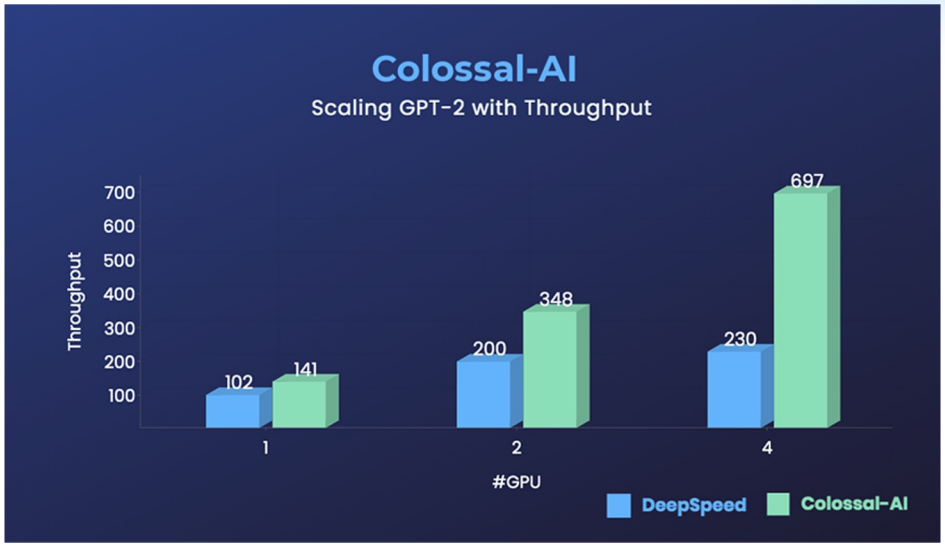


- 120x larger model size on the same hardware, higher acceleration
- One RTX 3080

4 Benchmark & Use Cases

Training Benchmark

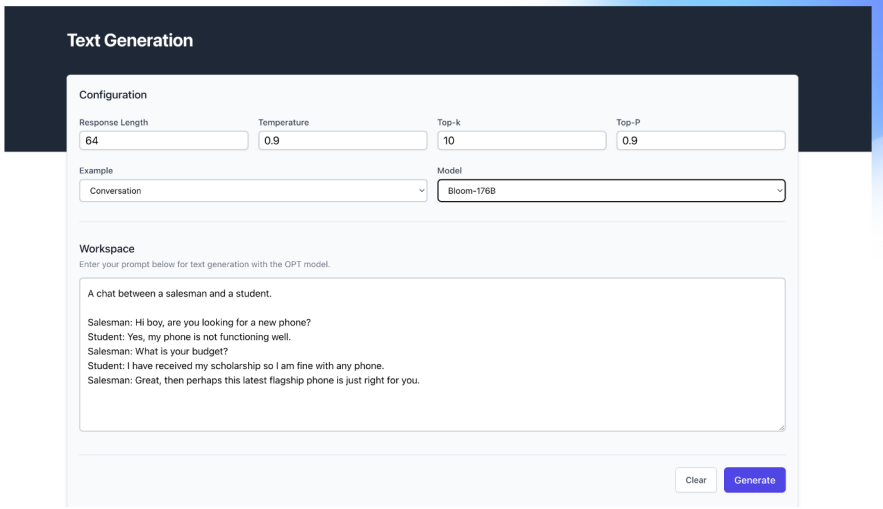


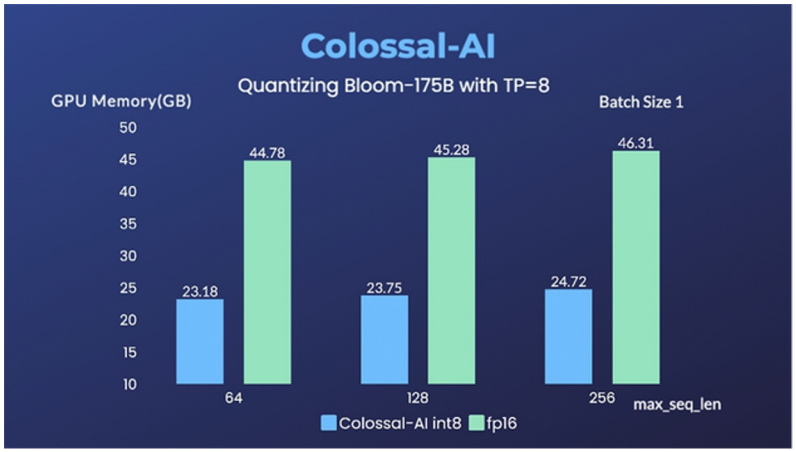


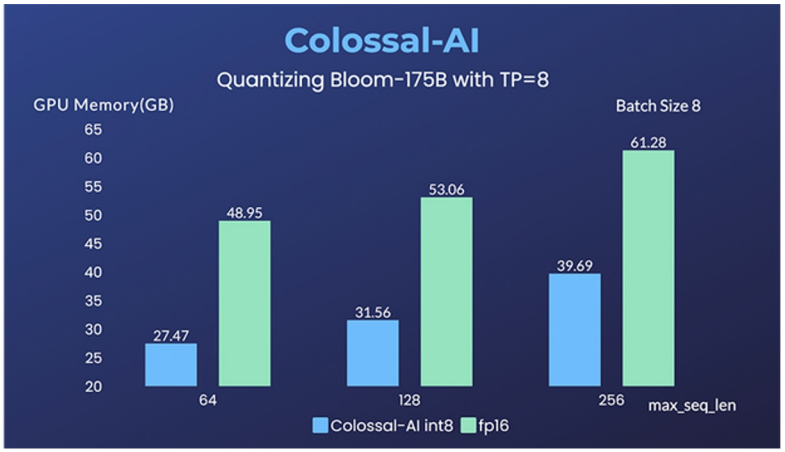
- Up to 24x larger model size on the same hardware vs PyTorch, over 3x acceleration vs DeepSpeed

Inference Benchmark

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







Stable Diffusion - Training Benchmark

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

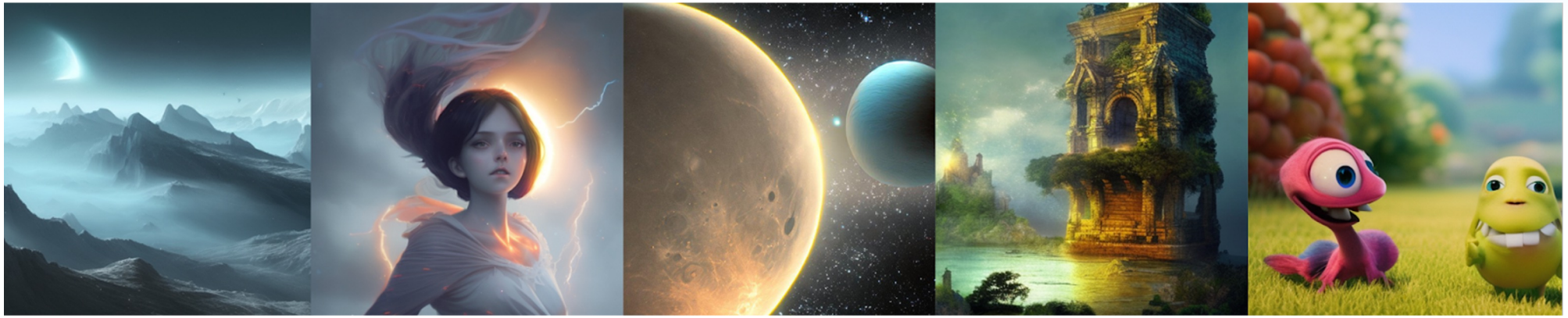
- Reduce GPU memory consumption by up to 5.6x
- Reduce hardware cost by up to 46x (from A100 to RTX3060)
- Can extend to multiple GPUs in parallel

Stable Diffusion - Inference Benchmark

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

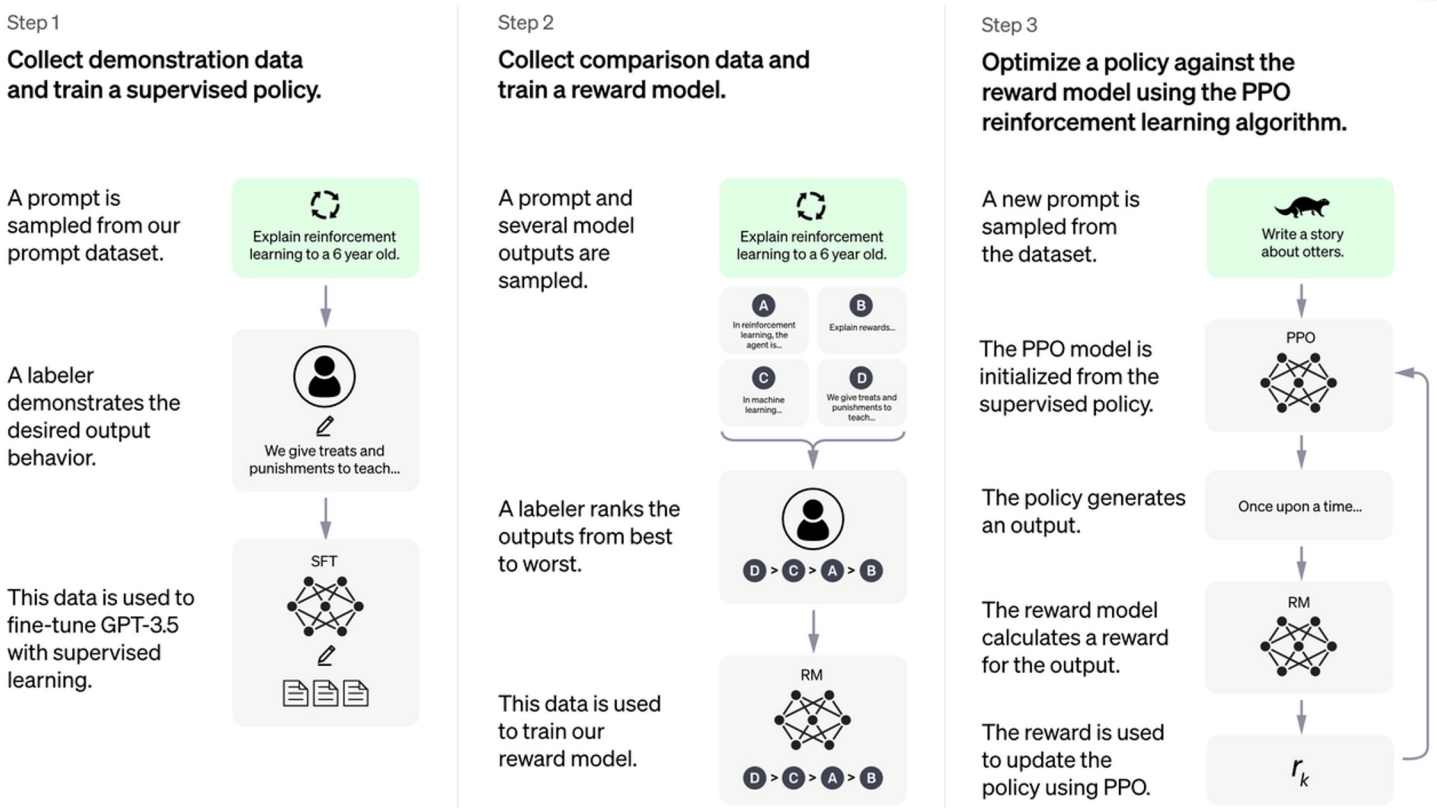
- 2.5 times lower memory consumption (3.1 GB memory required)
- Quantized for inference with a single line of code
  model = replace_module(model)

Stable Diffusion - DreamBooth Fine-tuning with ColosslalAI+LoRA



- Only 3-5 images of a desired subject to personalize text-to-image models
- Reducing GPU memory from 16GB to 4GB for common user

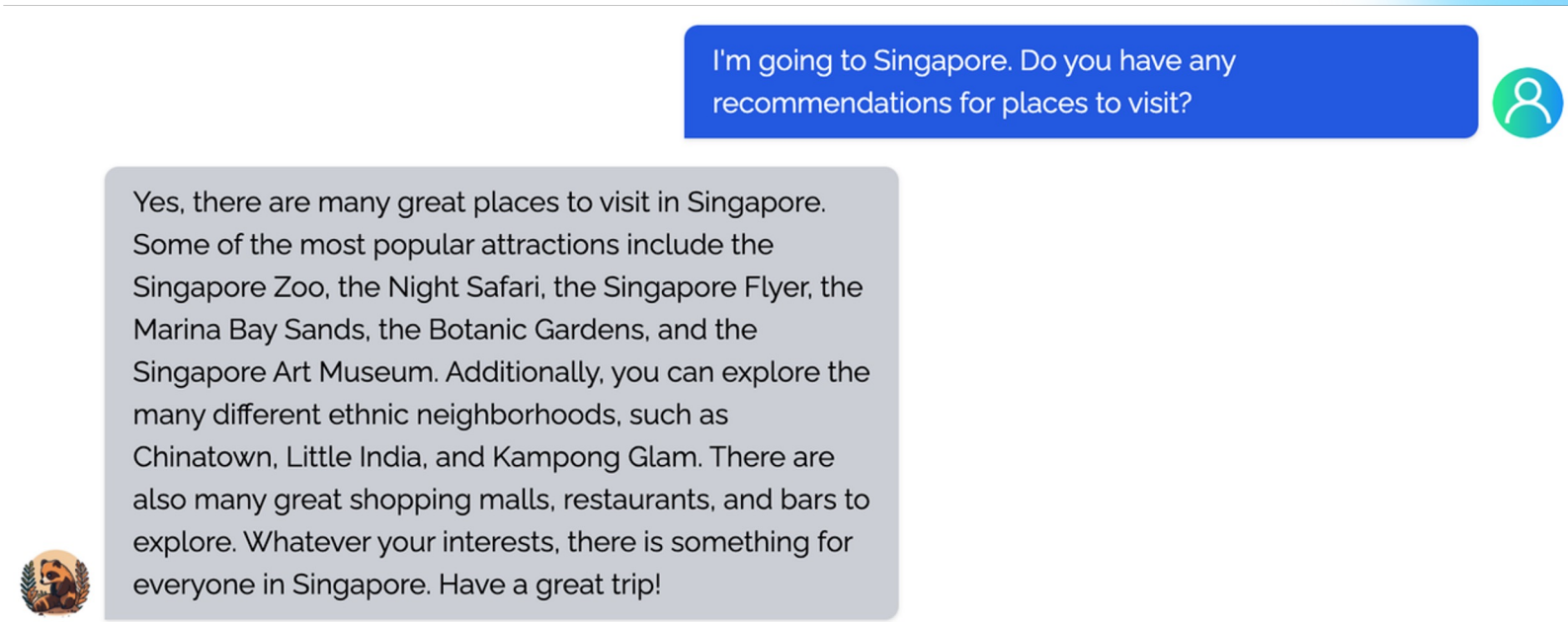
ChatGPT



- ChatGPT

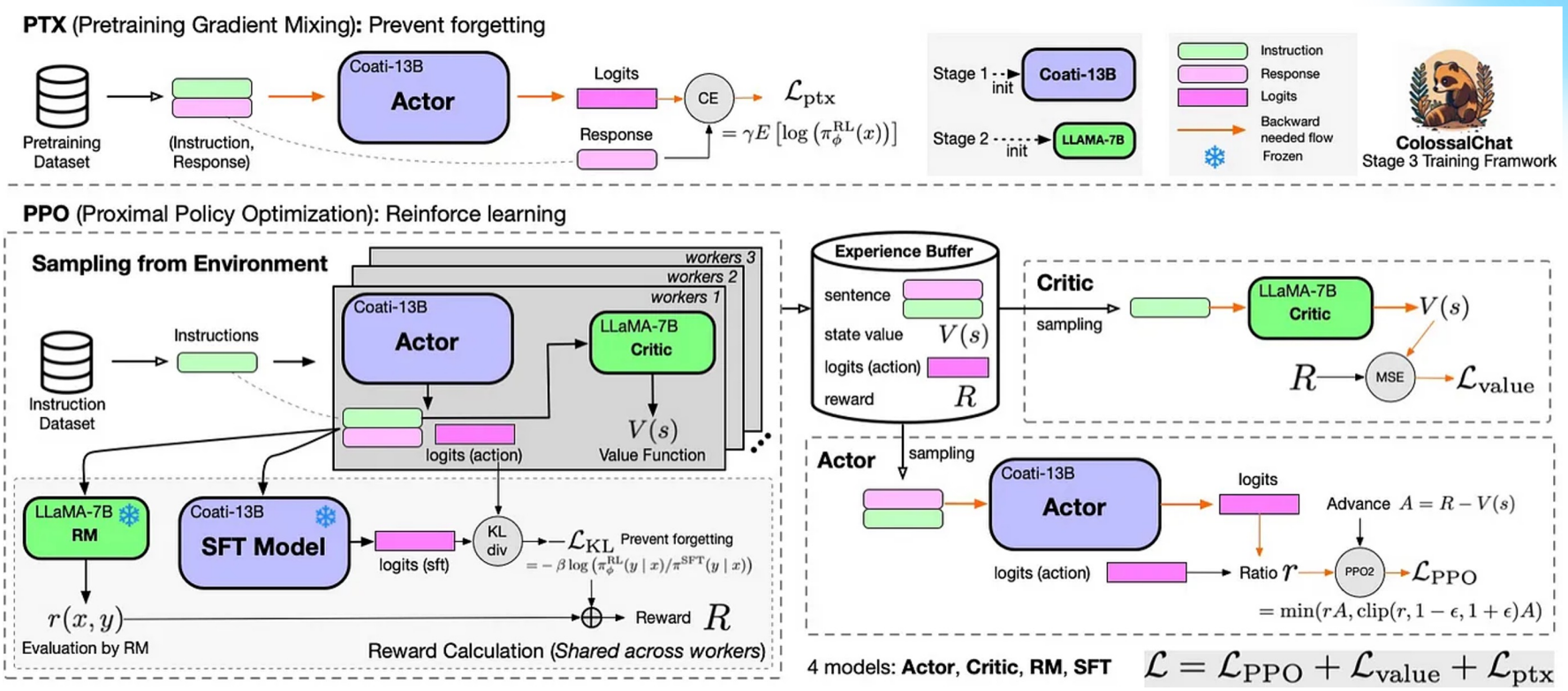
Ouyang, Long, et al. "Training language models to follow instructions with human feedback." arXiv preprint arXiv:2203.02155 (2022).

ColossalChat



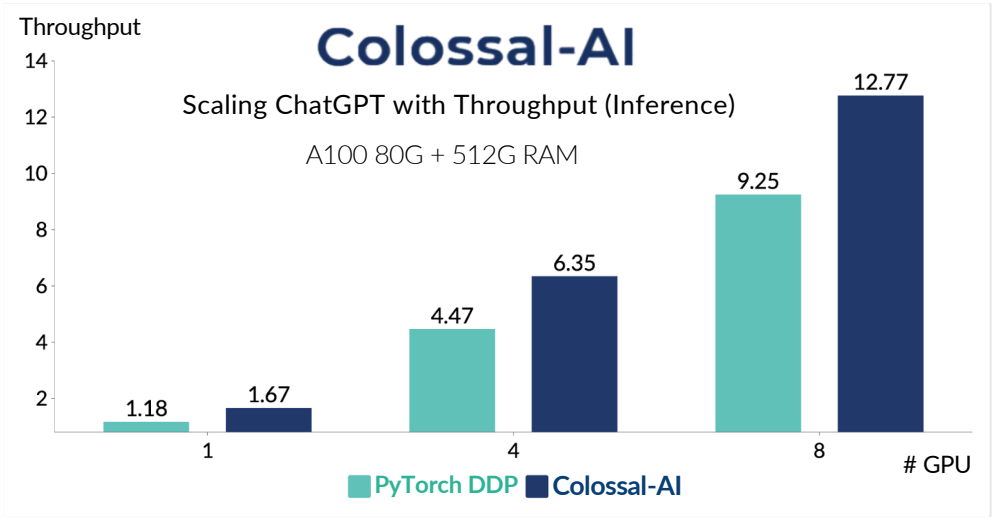
- The first most practical open-source project that closely resembles the original ChatGPT technical solution!
- 10 billion parameters achieving comparable results to ChatGPT and GPT-3.5.

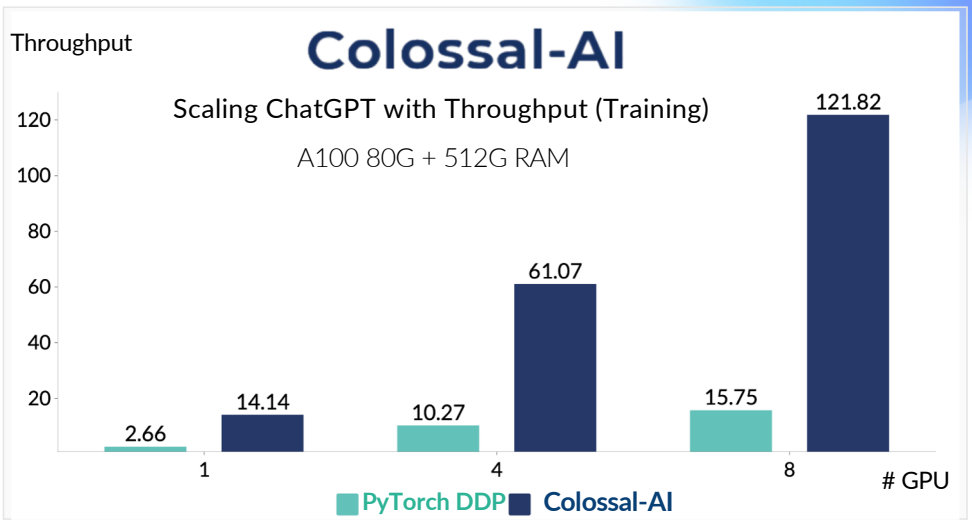
ColossalChat

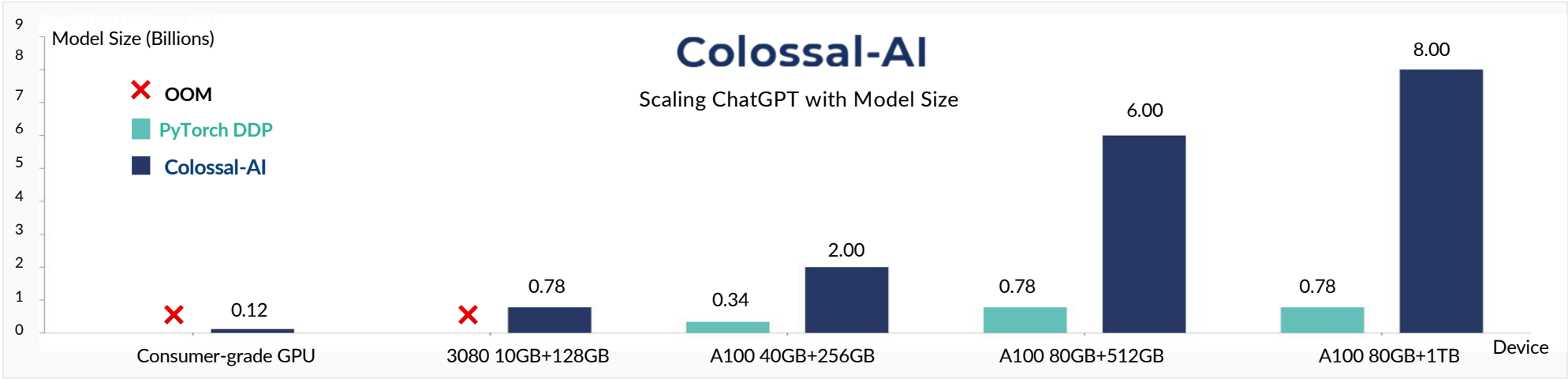


- RLHF-Stage3's Complete Flowchart

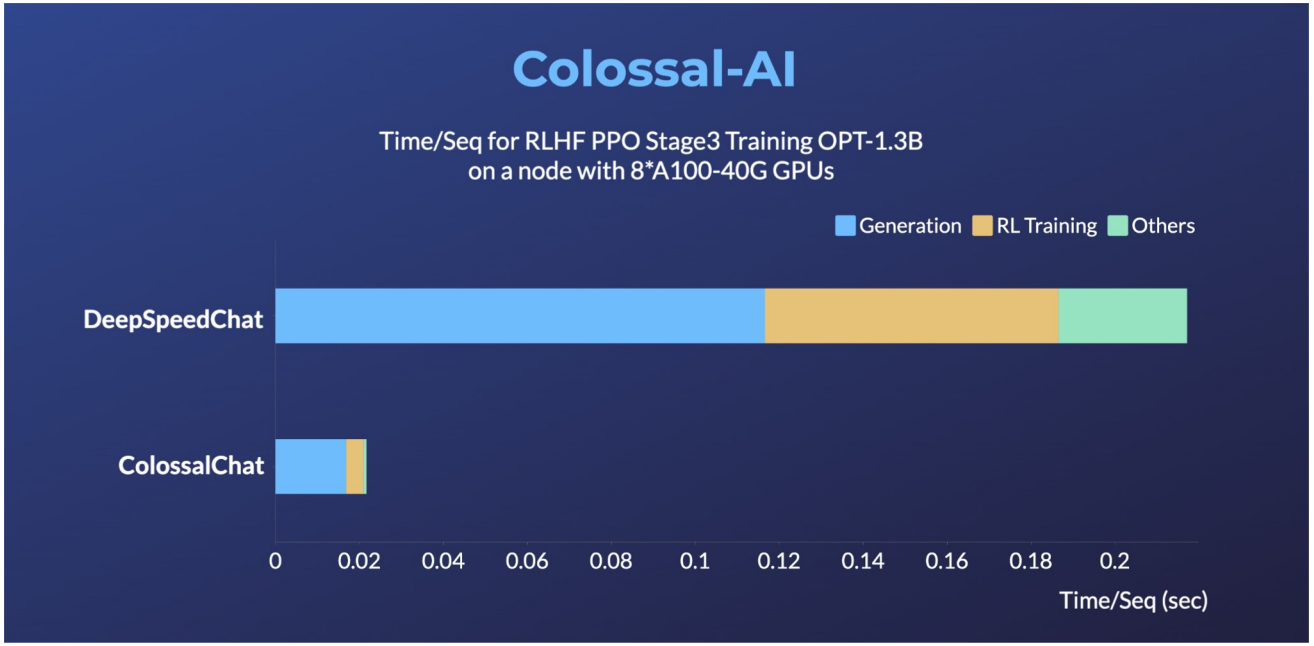
Replicates ChatGPT Training: 7.73x Faster, Lowest at 1.6GB GPU Memory





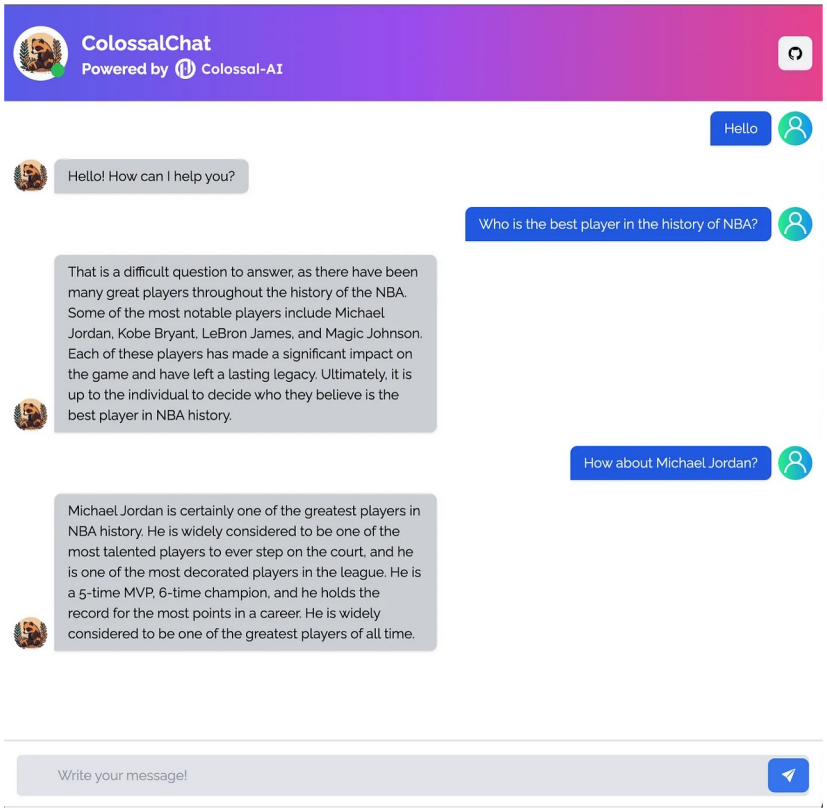


ChatGPT - Training

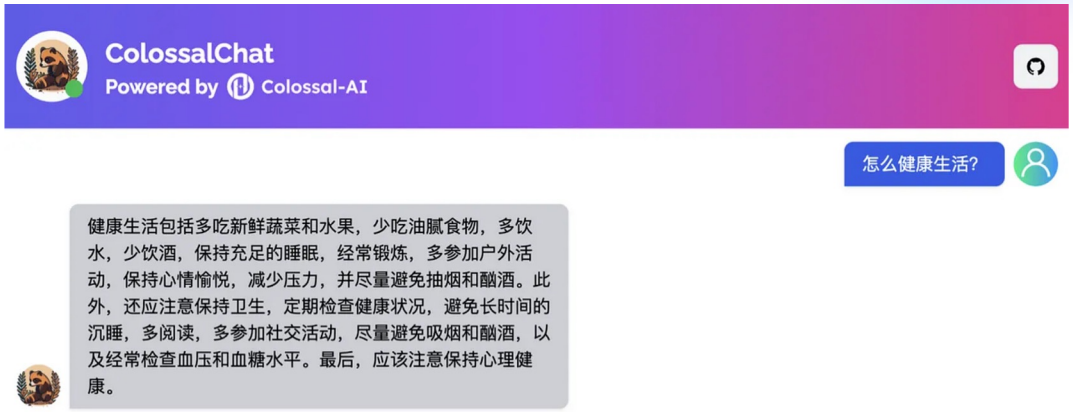


- Up to 10 times faster for RLHF PPO Stage3 Training

ColossalChat

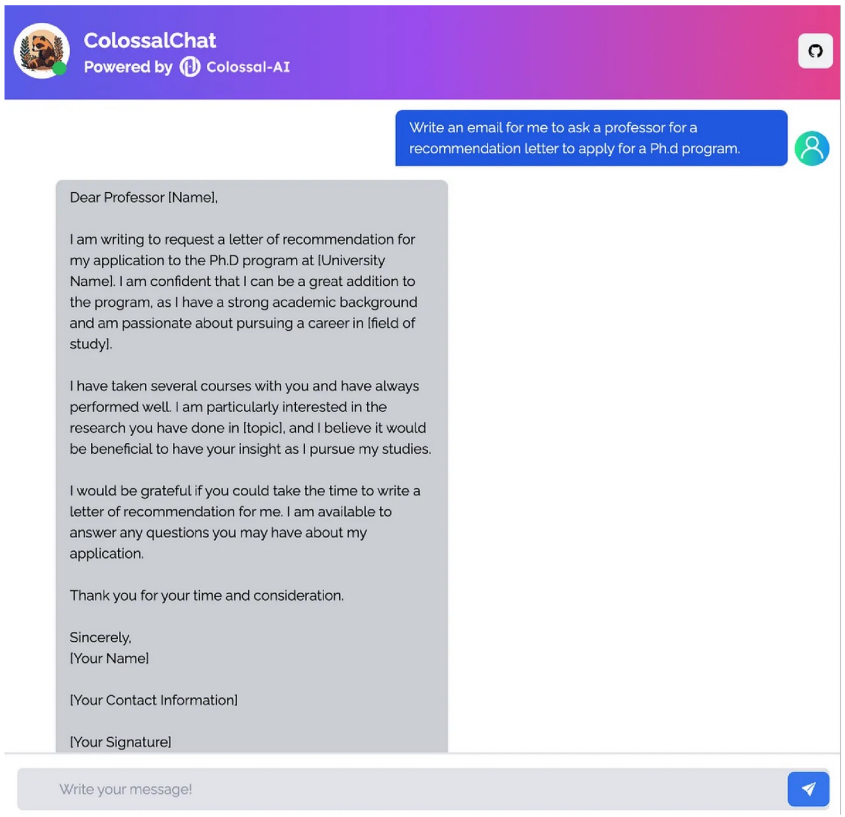


- General knowledge quiz

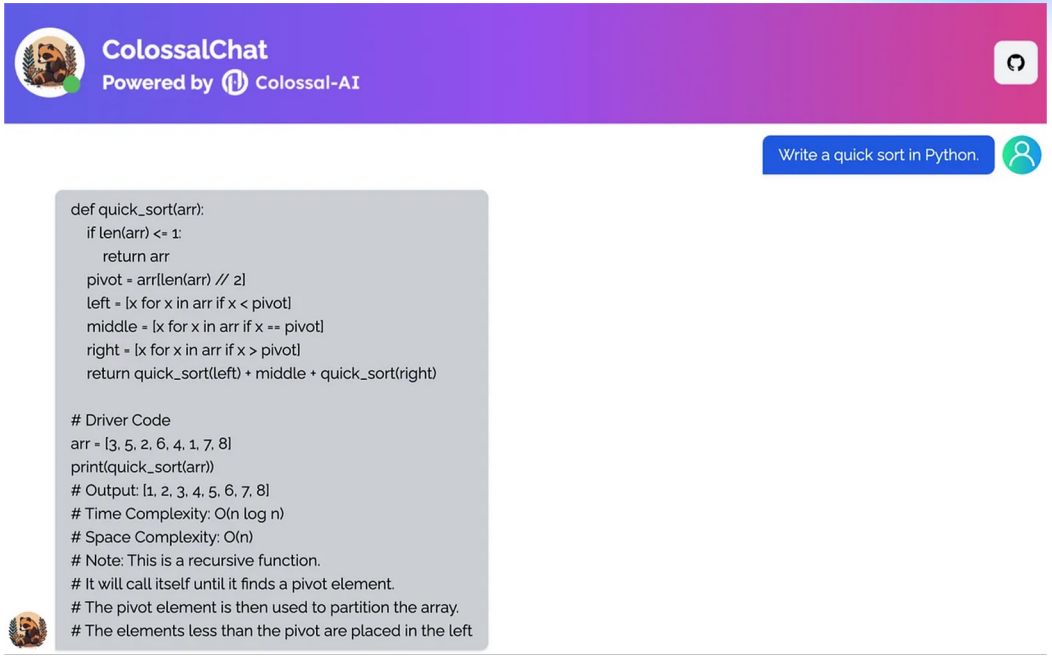


- Answering in Chinese

ColossalChat



- Write an email



- Write an algorithm

ColossalChat vs. Alpaca by Stanford

Write a quick sort in Python.

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# Driver Code
arr = [3, 5, 2, 6, 4, 1, 7, 8]
print(quick_sort(arr))
# Output: [1, 2, 3, 4, 5, 6, 7, 8]
# Time Complexity: O(n log n)
# Space Complexity: O(n)
# Note: This is a recursive function.
# It will call itself until it finds a pivot element.
# The pivot element is then used to partition the array.
# The elements less than the pivot are placed in the left
```

ColossalChat

Alpaca

```
def quickSort(arr):
    n = len(arr)
    if n < 7:
        p = 0
        for i in range(1, n):
            if arr[i - 1] > arr[i]:
                arr[p], arr[i] = arr[i], arr[p]
                p = i + 1
        arr[n - 1], arr[p] = arr[p], arr[n - 1]
    return arr
```

- Write a Quicksort in Python

Write an email for me to ask a professor for a recommendation letter to apply for a Ph.D. program.

Dear Professor [Name],

I am writing to request your help in applying for a Ph.D. program. I am confident that my academic record and research experience make me a strong candidate for the program, and I am hoping that you would be willing to provide a letter of recommendation for me.

I have included my CV and a brief description of my research experience and academic background for your reference. I am available to answer any questions you may have about my qualifications.

I greatly appreciate your help and look forward to hearing from you.

Sincerely,
[Your Name]

[Attached CV and brief description of research experience and academic background]

[Optional: Thank you for your time and consideration.]

[Optional: Please do not hesitate to contact me if you have any questions.]

ColossalChat

Alpaca

```
Dear Professor [Name],

I am writing to request your help in applying for a Ph.D. program. I am very interested
in pursuing a doctoral degree in your field and am confident that I have the necessary q
ualifications and background to be successful in the program.

I have attached my resume for your review and would be grateful if you could provide me
with a letter of recommendation. I am sure that your insight and experience will be inva
luable to me as I pursue this next step in my academic career.

Thank you for your time and consideration.

Sincerely,
```

- Write an email to a professor for a recommendation letter

Rohan, Taori, et al. "Stanford Alpaca: An Instruction-following LLaMA model." arXiv preprint arXiv:2302.13971 (2023).

Evaluation - DataSet

- Carefully selected 1000 test samples in 10 categories
  - Role-playing
  - Multi-round conversation
  - Open-ended Q&A
  - Closed-ended Q&A
  - Brainstorming
  - Language generation
  - Content Rewriting
  - Categorization
  - Information extraction
  - Summarization
- 100 test samples per category

- Open-ended Q&A
  - How do you take the derivative of the sin function?
- Language generation
  - Can you help me write a formal email to a potential business partner proposing a joint venture?
- Content Rewriting
  - Translate the following text into English:
    我最喜欢的季节是春天，因为我可以看到美丽的花朵。
- Summarization
  - What information is provided in the table below? Summarize the core information in it？

    Ranking, Player Name, Team, Position, Salary (in millions of dollars)
    1, LeBron James, Los Angeles Lakers, SF, 45.0
    2, Stephen Curry, Golden State Warriors, PG, 43.5

Evaluation - Standard

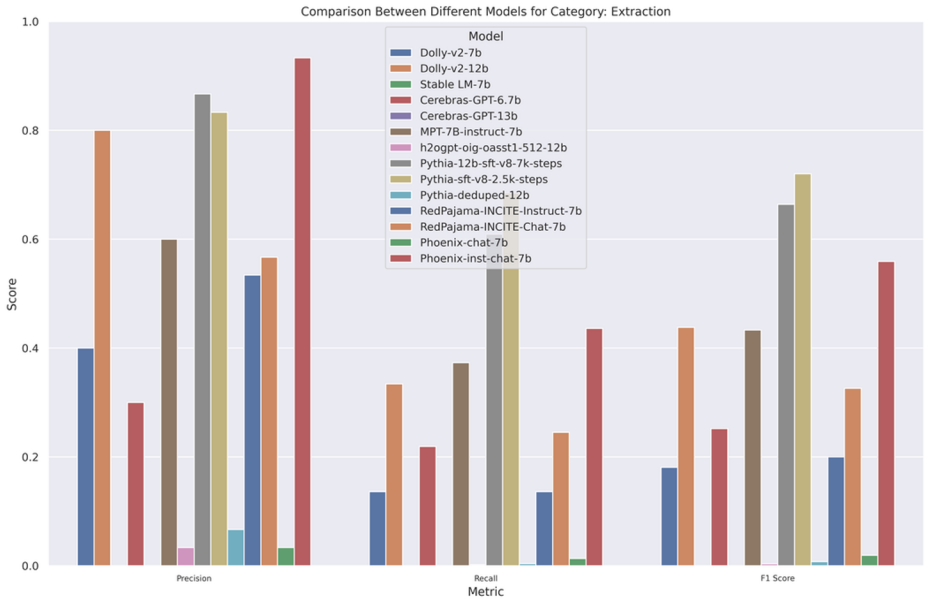
- **GPT-4 API Evaluation with Carefully Designed Prompts**
  - Language Organization
  - Relevance
  - Creativity
  - Practicality
  - Accuracy
  - Naturalness
  - Engagingness
  - Reasonableness
  - Diversity
  - Fidelity
  - Conciseness

- **Human Evaluation**
  - Blind rating of the outputs generated by different models on randomly selected questions

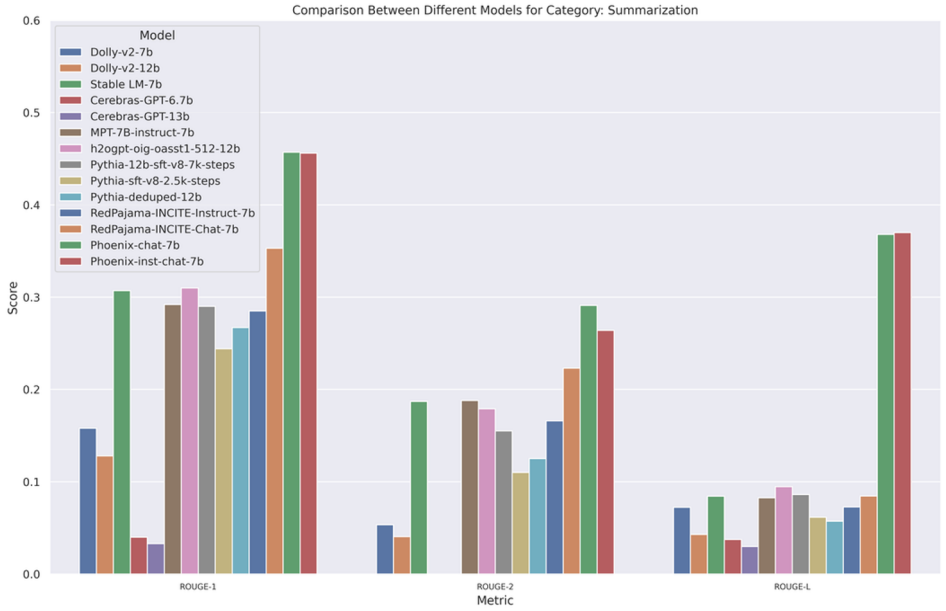
Evaluation Metrics	Target Category
BLEU	Closed-ended Q&A, Language generation, Content Rewriting, Summarization
ROUGE	Closed-ended Q&A, Language generation, Content Rewriting, Summarization
Distinct	Role-playing, Multi-round conversation, Open-ended Q&A, Brainstorming
BERTScore	Closed-ended Q&A, Language generation, Content Rewriting, Summarization
Precision, Recall, F1 Score	Categorization Information extraction

- **Automated Metrics Evaluation**

Evaluation - Benchmark



Information extraction
Precision / Recall / F1 Score



Summarization - ROUGE

- More details about ColossalChat are available on the project homepage https://github.com/hpcaitech/ColossalAI

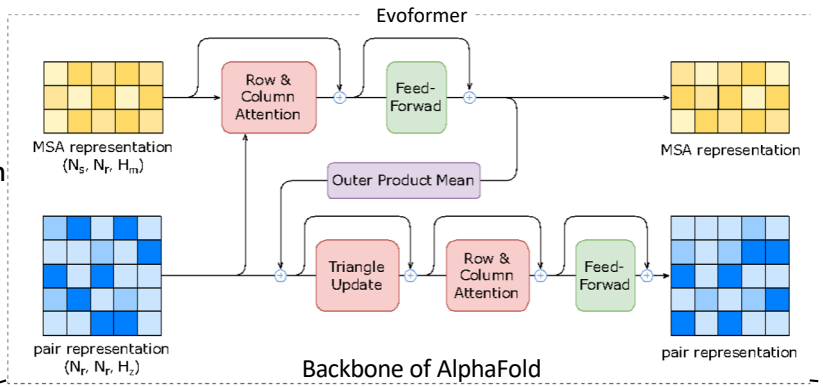
Reducing AlphaFold Training Time (Drug Discovery) from 11 Days to 67 Hours

Challenges:

1) Time and costs

2) Memory Consumption

3) Long Sequence Inference



For Training

1) Overall training time: *11 days -> 67 hours* with significant *economic cost savings*

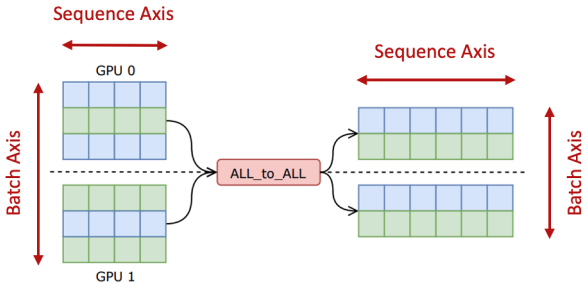
2) Scaling to 512 × A100 with *6.02 PetaFLOPs*

For Inference

1) 7.5 ~ 9.5× speedup for long sequences

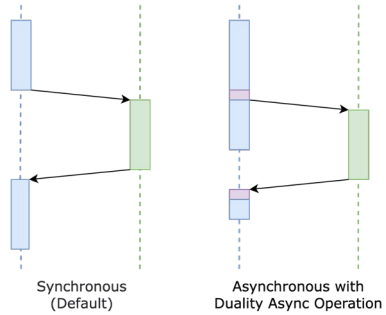
2) makes it possible for inference over extremely long sequences

Parallel Evoformer



Dynamic Axial Parallelism

A novelty approach for efficient model parallelism



Duality Async Operation

A PyTorch extension for communication optimization

Computational Optimizations

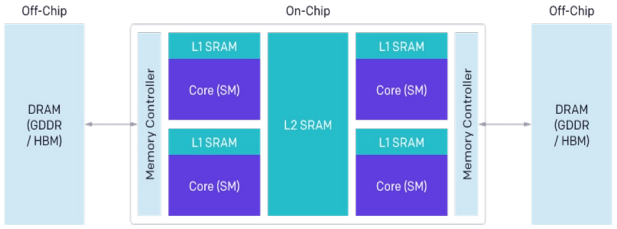
1) Merge GEMM in *Attention* and *Triangle Update Module*

2) JIT Fusion *(bias + dropout + add*, etc*)*

3) Optimized Kernel (LayerNorm, Fused Softmax)

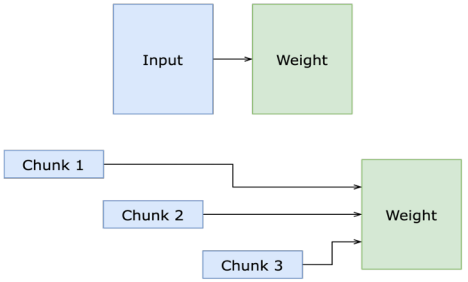
Singular-GPU Inference Sequence Exceeding 10,000, Covering 99.9999% of Proteins

Computation Optimization

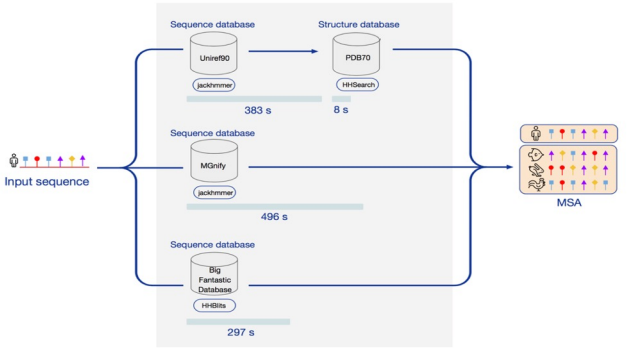


Optimized Kernels Based on Triton

Memory Optimization

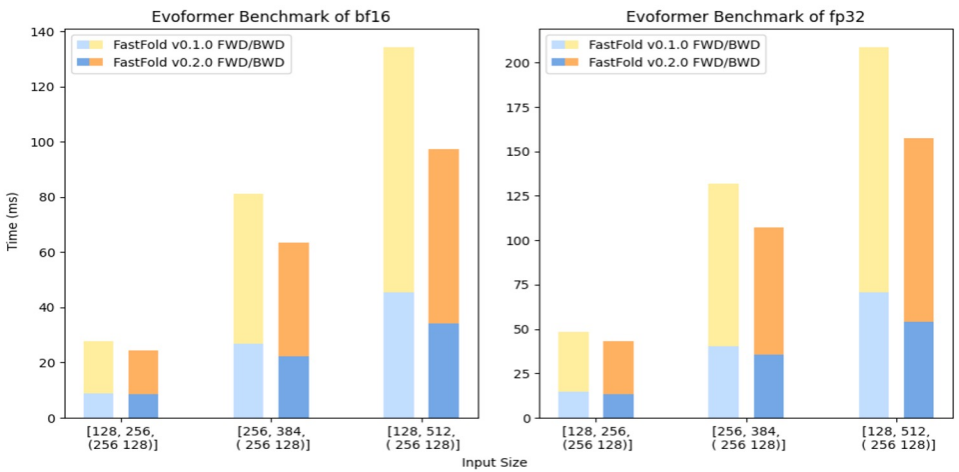


Chunk Optimization

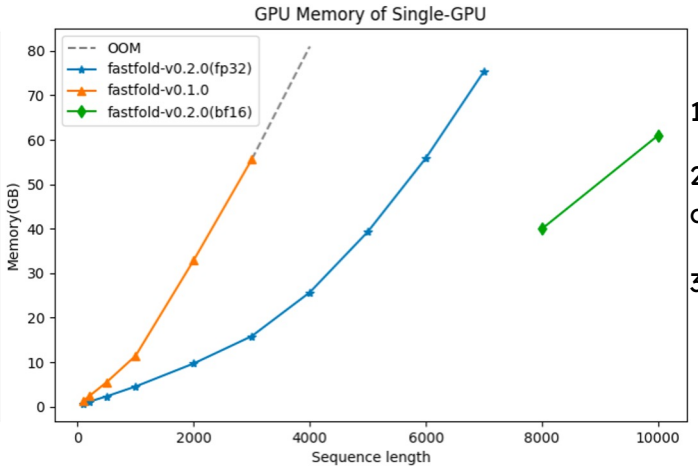


Parallel Acceleration of Pre-processing

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/

Thanks for your time !

https://github.com/hpcaitech/ColossalAI

Join Colossal-AI Slack!

Welcome to contribute !