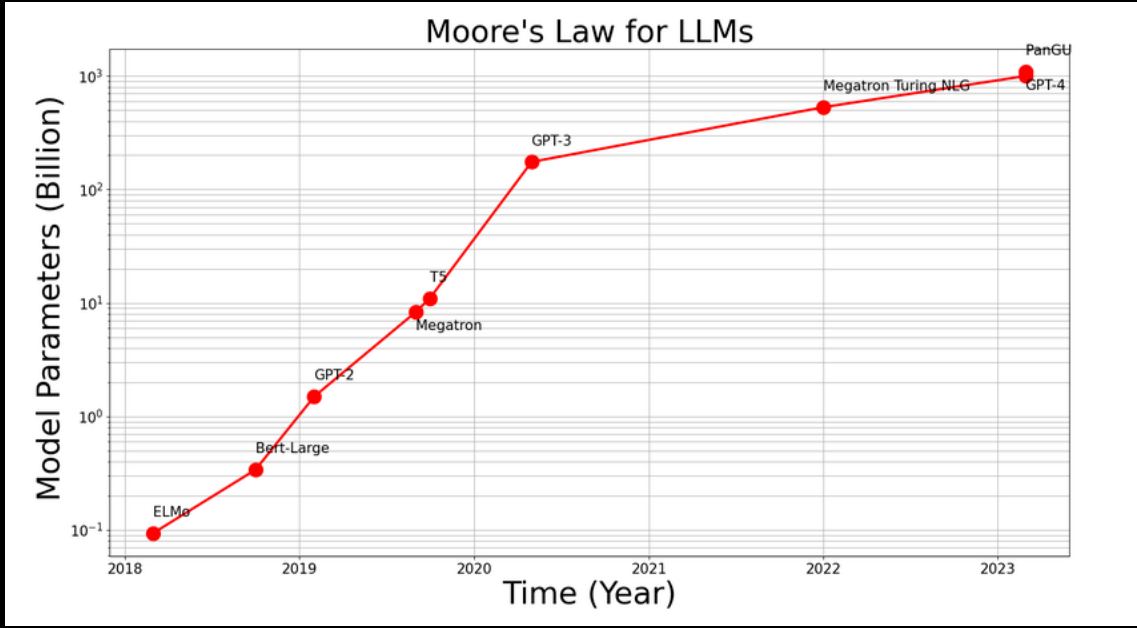
Maximizing LLM Throughput in PyTorch: Optimized Pipelines for Modern Deep Learning Workloads

Davis Wertheimer
IBM Research

Motivation



Source: ChatGPT in the Age of Generative AI and Large Language Models: A Concise Survey (2023)

Motivation: Scope of Pretraining

Some back-of-the-envelope math:

70B parameter LLM

x 10T tokens of training data

x 3 (1 forward pass, 2 backward pass)

= 2.1 * 10^{24} FLOPs

213.3 GPU-years!

IBM

SIG METRICS acm

Motivation: Scope of Inference

Some more back-of-the-envelope math:

70B parameter LLM

x 16-bit precision

= 140 GB streaming through device cache

Vocab size **128K**

Write ~2 bytes by reading 140GB!

IBM

SIG METRICS

Motivation

Nasty synergies between pretraining and inference costs:

- Synthetic data
- Reinforcement Learning from Human Feedback (RLHF)
- Policy Optimization (PPO, DPO, GRPO, etc)
- Multiple generations required for the last two!

Takeaways

- This is a talk about building platforms and codebases
- LLM optimization: multifaceted and open-ended
  - Parallel pipelines run at the speed of the slowest component
  - Juggling many bottlenecks and many constraints
- No one-size-fits-all solutions!
- 80/20 rule: a long tail of possible bottlenecks
- Breadth over depth

IBM

SIG METRICS

Takeaways

1. **Familiarity with general concepts and possible pitfalls**
2. Some useful tools
3. **Comfortably think about, discuss, and operate LLMs at scale**
4. **The right mindset / perspective for approaching the problem**

IBM

SIG METRICS

Who am I?

- Researcher at IBM
- 3 years working on LLM training
- PhD in AI and CV
- Focus on ML under constraints
- Fractal artist, jewelry designer

IBM

SIG METRICS

Who am I?

Our project at IBM:
an open-source, cloud-native platform for foundation model training, fine-tuning, and hosting

IBM Foundation Model Stack

IBM Pretraining Repo (fms-fsdp)

IBM

SIG METRICS

Contents

- Storytelling – what it takes to optimize LLMs at scale
- Know where to start, if you want to do the same
- Background:
  - User-level basic familiarity with PyTorch
  - Familiarity with transformer architecture (attention heads, vocab layers, etc)
  - Familiarity with basic training loop (forward/backward passes, causal LM)

IBM

SIG METRICS acm

Contents

1. INTRO
   - Motivation
   - Takeaways
   - Who am I?
   - Contents
2. BASICS OF TRAINING AT SCALE
3. OPTIMIZING TRAINING AT SCALE
4. BAG OF TRICKS
5. DATA AT SCALE
6. ACCELERATING INFERENCE

IBM

SIG METRICS acm

Contents

IBM

SIG METRICS acm

PART 2: The Basics of LLM PreTraining at Scale

Contents

IBM

SIG METRICS acm

Contents

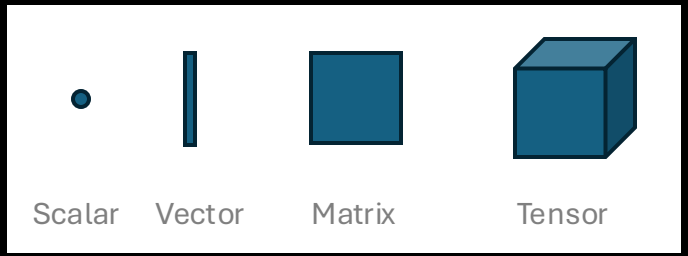
IBM

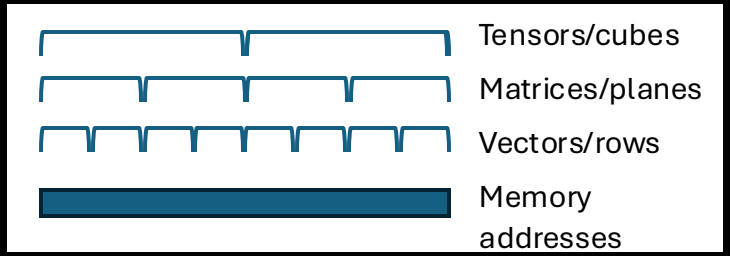
SIG METRICS acm

Revisiting PyTorch: Tensor Basics

PyTorch is a math engine that automatically computes gradients

- Atomic object: tensor
- Linear memory register mapped to higher dimensional grids
- Many operations supported for tensors
  - Algebra: add, subtract, multiply, exponent, log, etc.
  - Matrix math: matmul, triu, diagonal embed, transpose, flip, etc.
  - Data ops: reshape, typecast, transfer to cpu/gpu, slice, concat, broadcast, etc.





IBM

SIG METRICS

Revisiting PyTorch: Tensor Basics

```
# Perform actual attention operation
score = k_.unsqueeze(2).matmul(xq.transpose(-1,-2)).add(affinity.unsqueeze(2))  # b h r c l
denom_ = score.logsumexp(dim=-2)  # b h r l
out_ = score.transpose(-1,-2).softmax(dim=-1).to(dtype=xq.dtype).matmul(v_.unsqueeze(2))  # b h r l d
```

IBM

SIG METRICS acm

Revisiting PyTorch: AutoGrad Basics

- (Most) supported ops implement a “backward” function
- Given inputs, outputs*, and gradient w.r.t. outputs, calculate gradient w.r.t. inputs

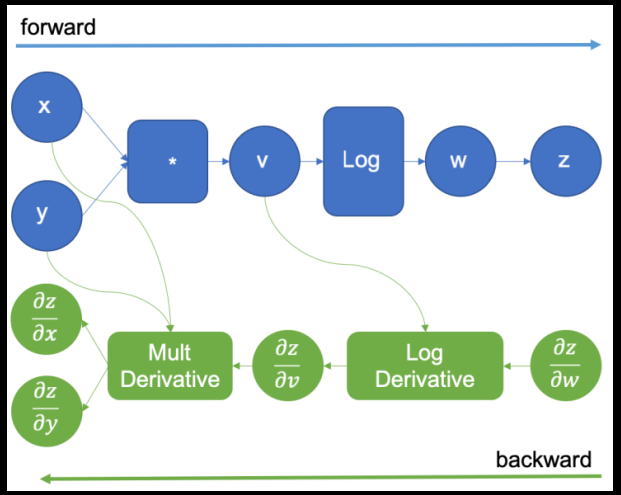
$$WX = Y, \qquad L = f(Y), \qquad \frac{dW}{dL} = X\left(\frac{dY}{dL}\right)^T, \qquad \frac{dX}{dL} = W^T\frac{dY}{dL}$$

Revisiting PyTorch: AutoGrad Basics

- (Most) supported ops implement a “backward” function
- Given inputs, outputs*, and gradient w.r.t. outputs, calculate gradient w.r.t. inputs

$$AB = C, \qquad L = f(C), \qquad \frac{dA}{dL} = B\left(\frac{dC}{dL}\right)^T, \qquad \frac{dB}{dL} = A^T\frac{dC}{dL}$$

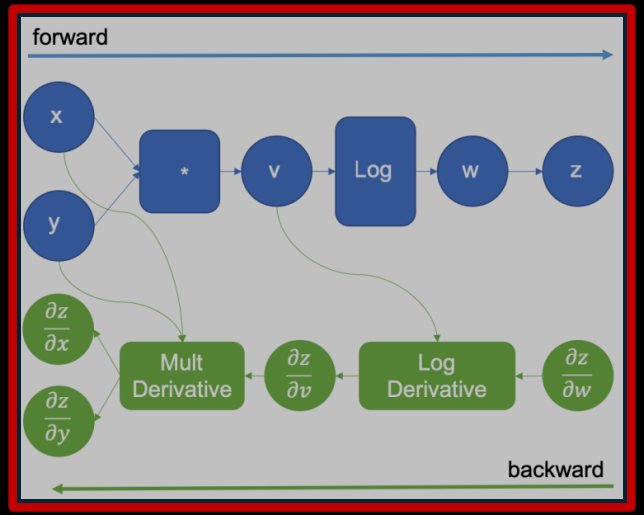
- Composed ops dynamically create a computation graph
- Graph computes gradients via composed backward calls



Source: https://pytorch.org/blog/overview-of-pytorch-autograd-engine/

Revisiting PyTorch: Modules and Networks

- The other “atomic” object: modules
- Contain parameters (persistent tensors)
- Implement a forward function with tensor inputs/outputs
- Implicit backward function stores gradients for parameters
- Essentially wraps function compositions into black boxes
- Optimizers use stored gradient to update parameters
- Blocks are composed into layers and networks



Source: https://pytorch.org/blog/overview-of-pytorch-autograd-engine/

Revisiting PyTorch: Hooks

- Hooks: backend functions triggering before/after forward/backward
- PyTorch implements distributed training through hooks
  - Coordination, synchronization, sharding, etc.
- From the programmer's perspective, it's a model wrapper

```
# FSDP
model = FSDP(
    model,
    auto_wrap_policy=wrapping_policy,
    mixed_precision=mixed_precision_policy,
    sharding_strategy=sharding_strategy_policy,
    use_orig_params=cfg.use_torch_compile,
    device_id=torch.cuda.current_device(),
    limit_all_gathers=True,
    param_init_fn=param_init_fn,
)
```

Contents

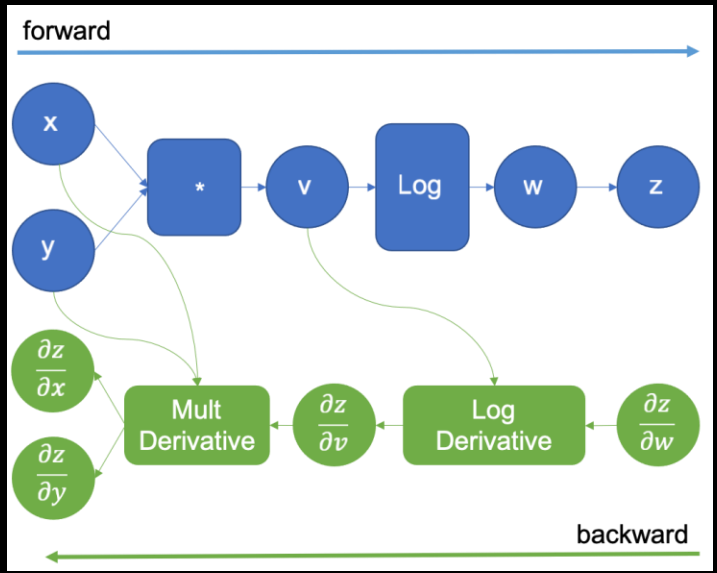
IBM

SIG METRICS acm

Data Parallel

Problem: you have a model on 1 GPU, 3 unused GPUs, and you need to do a forward/backward pass on a batch of data. The model stores too many inputs to fit on 1 GPU. What do you do?

Naïve solution: split batch into 4, run ¼ the data on each GPU, and consolidate (sum) gradients back to master GPU afterwards.

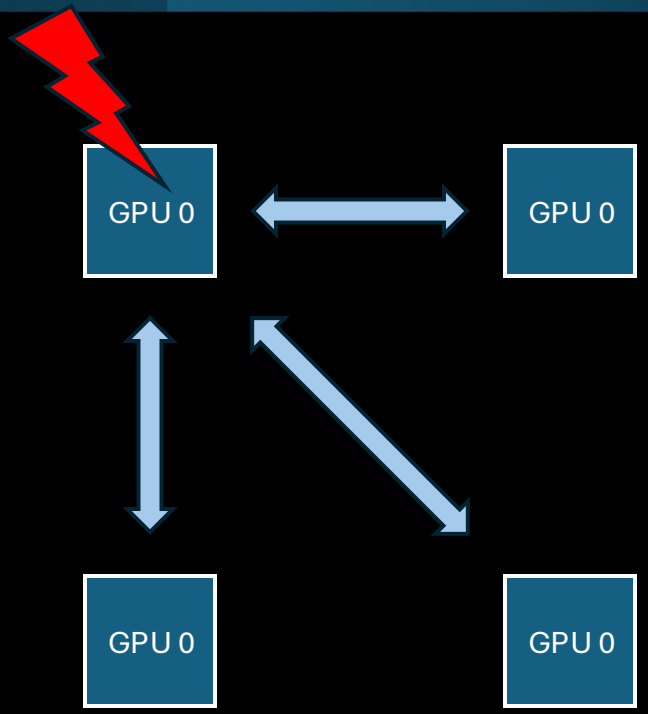


Source: https://pytorch.org/blog/overview-of-pytorch-autograd-engine/

Data Parallel

Problems:

1. Several devices' worth of traffic is all going through 1 GPU
2. Master process is handling multiple workers, but has to deal with GIL, so parallelism is limited



IBM

SIG METRICS

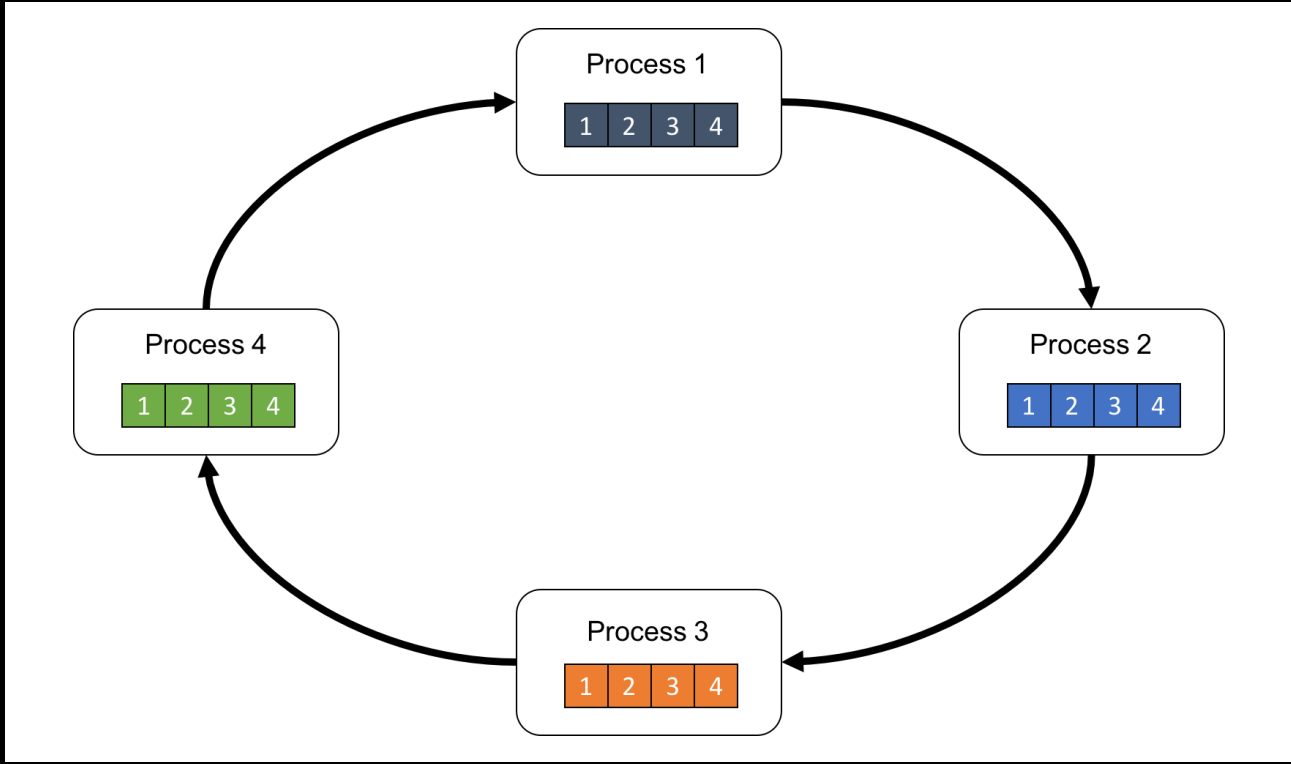
Distributed Data Parallel

- Replicate model across all ranks
  - Assume model is relatively small (<30B params)
- All-reduce gradients across all ranks
- Identical gradients ensure that model replicas update identically
- No extra communication overhead (!)

`DataParallel`	`DistributedDataParallel`
More overhead; model is replicated and destroyed at each forward pass	Model is replicated only once
Only supports single-node parallelism	Supports scaling to multiple machines
Slower; uses multithreading on a single process and runs into Global Interpreter Lock (GIL) contention	Faster (no GIL contention) because it uses multiprocessing

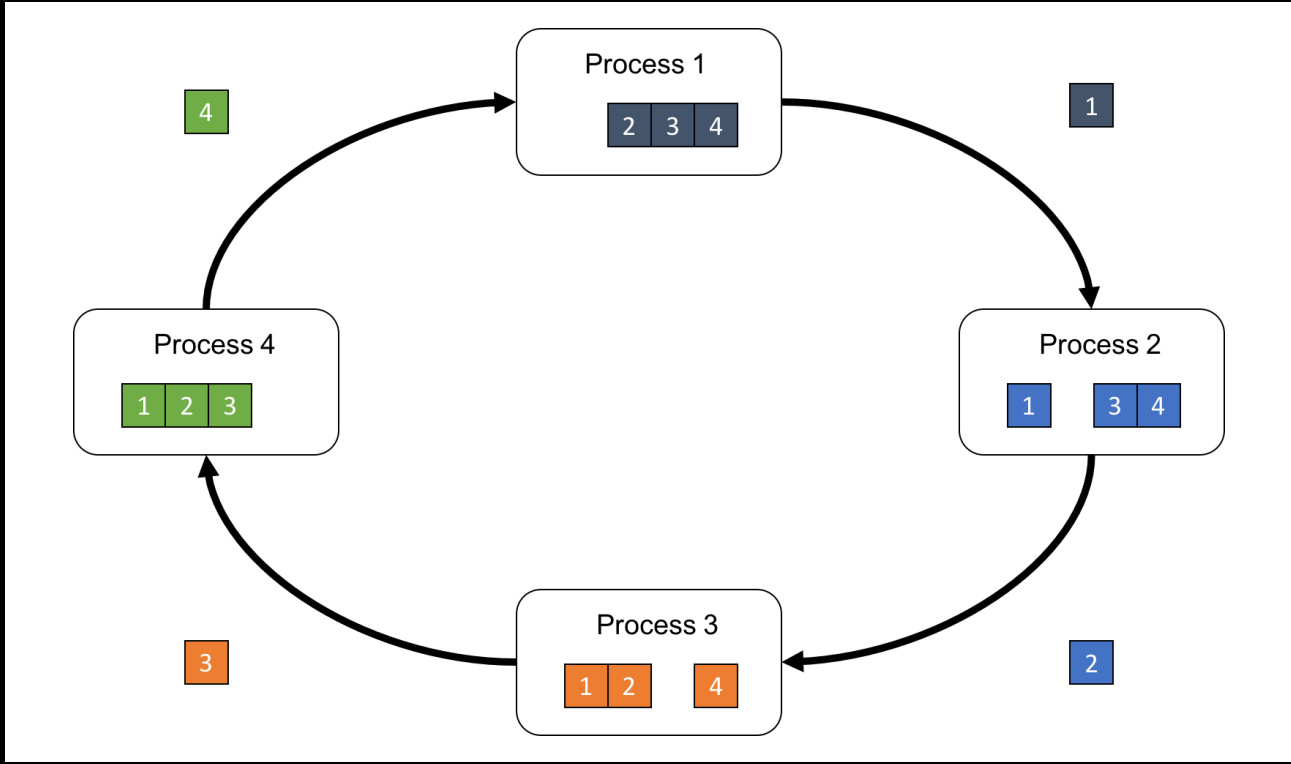
Source: *https://docs.pytorch.org/tutorials/beginner/ddp_series_theory.html*

Distributed Data Parallel



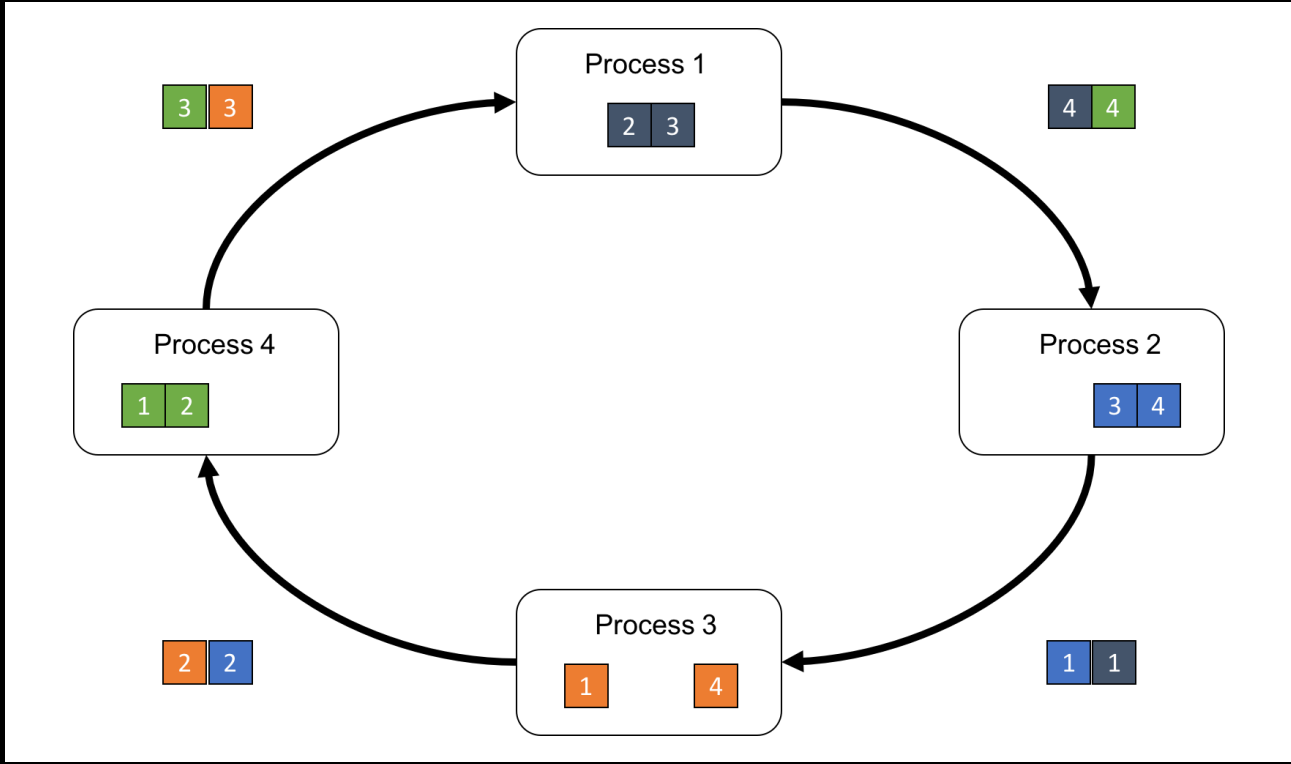
Source: https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/

Distributed Data Parallel



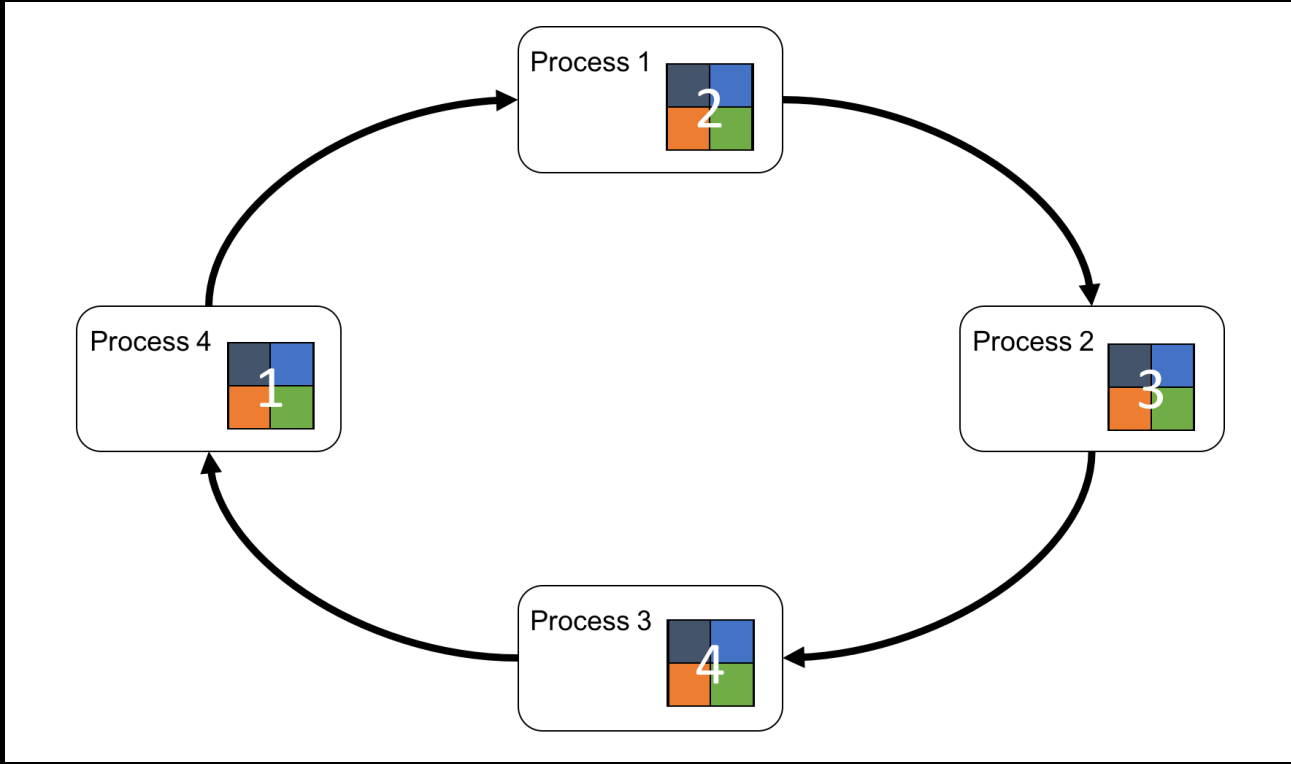
Source: https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/

Distributed Data Parallel



Source: https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/

Distributed Data Parallel



Source: *https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/*

Distributed Data Parallel

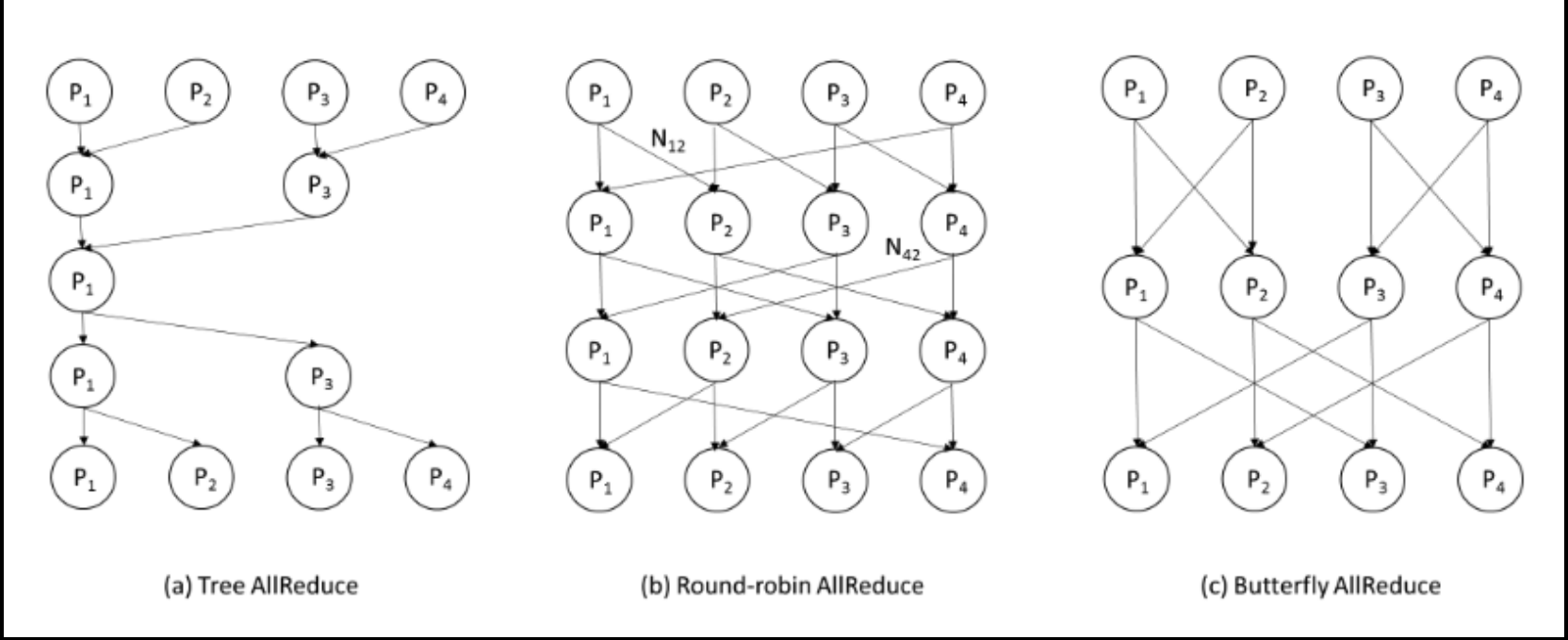
- DP overhead:

  X model size
  $\times\ (R-1)$ ranks
  $\times\ 2$ (model forward + gradient backward)
  $= 2X(R-1)$ total

- DDP overhead:

  X/R packet size
  $\times\ (R-1)$ steps
  $\times\ 2$ (sum step + propagate step)
  $\times\ R$ ranks
  $= 2X(R-1)$ total

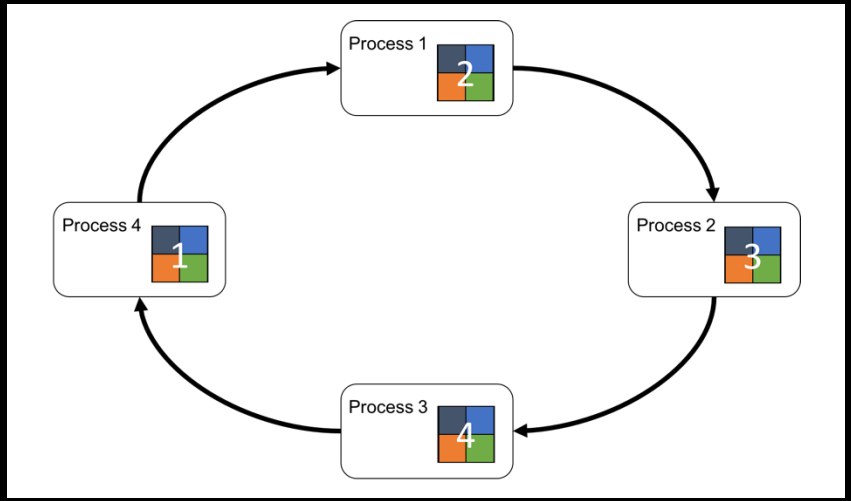
Distributed Data Parallel



Source: *Sparse Allreduce: Efficient Scalable Communication for Power-Law Data* (2013)

Fully Sharded Data Parallel

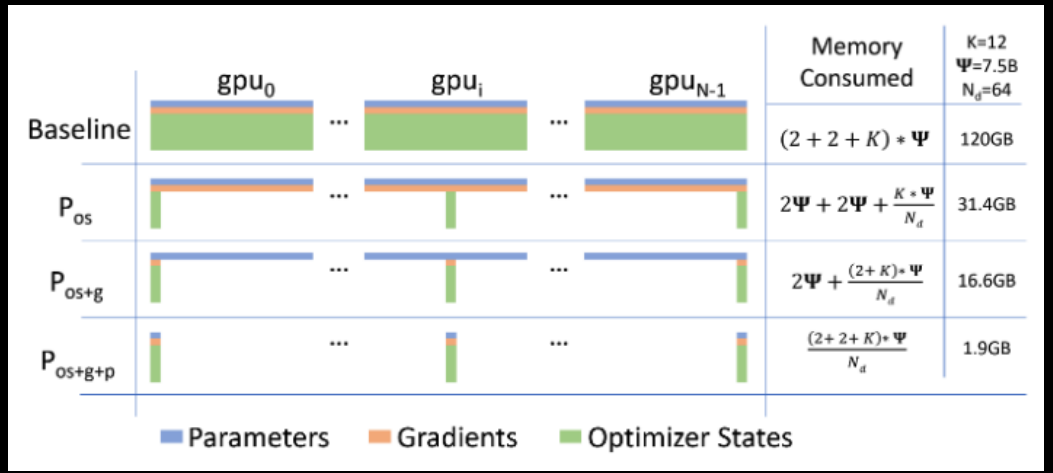
- DDP assumes your entire model fits on each GPU x
  - (and the gradient) $+x$
  - (and the optimizer) $+2x$
- Limit for an A100 is ~30B parameters
- We already split up the gradient for Ring-AllReduce, so do the model too!



Source: *https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/*

Fully Sharded Data Parallel

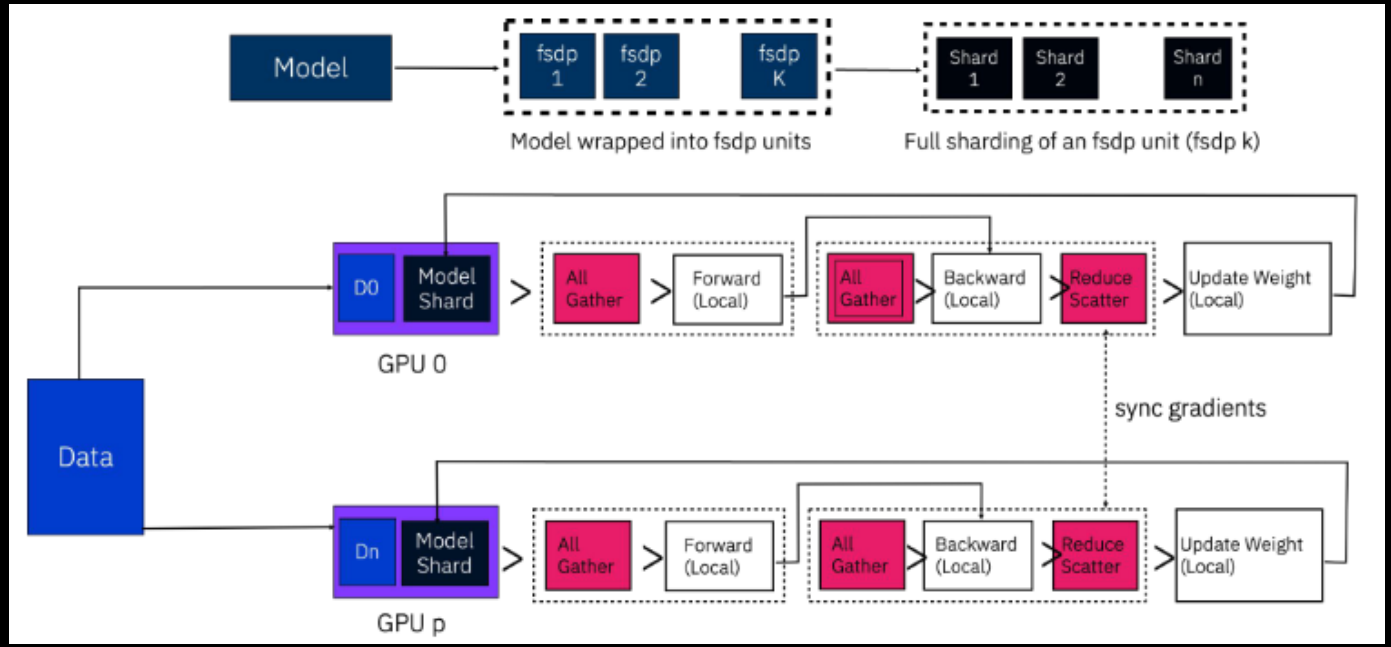
- DDP assumes your entire model fits on each GPU x
  - (and the gradient) $+x$
  - (and the optimizer) $+2x$
- Limit for an A100 is ~30B parameters
- We already split up the gradient for Ring-AllReduce, so do the model too! (and the optimizer)
- A.K.A. “Zero”-DP



Source: *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models* (2019)

Fully Sharded Data Parallel

Every rank holds 1/n of each layer, and layers/gradients are gathered and materialized on demand

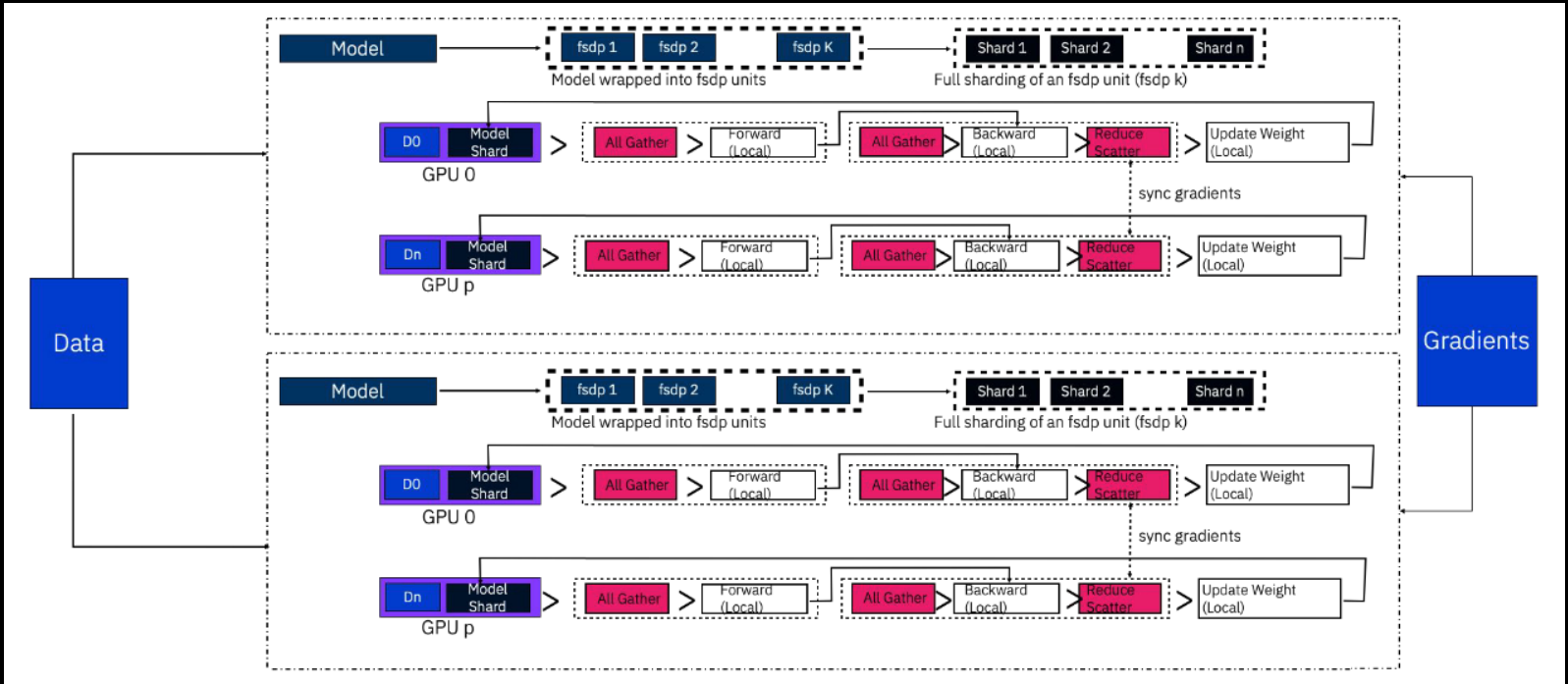


Fully Sharded Data Parallel

- 50% total increase in communication traffic over DDP (via Ring)
- MUCH more frequent communication of smaller packets
  - Tighter synchronization of devices
  - Latency over throughput
- Can overlap (most) communication with computation from the previous layer
- If training on multiple nodes, latency can become a problem

Hybrid Sharded Data Parallel

- FSDP within nodes, DDP between nodes

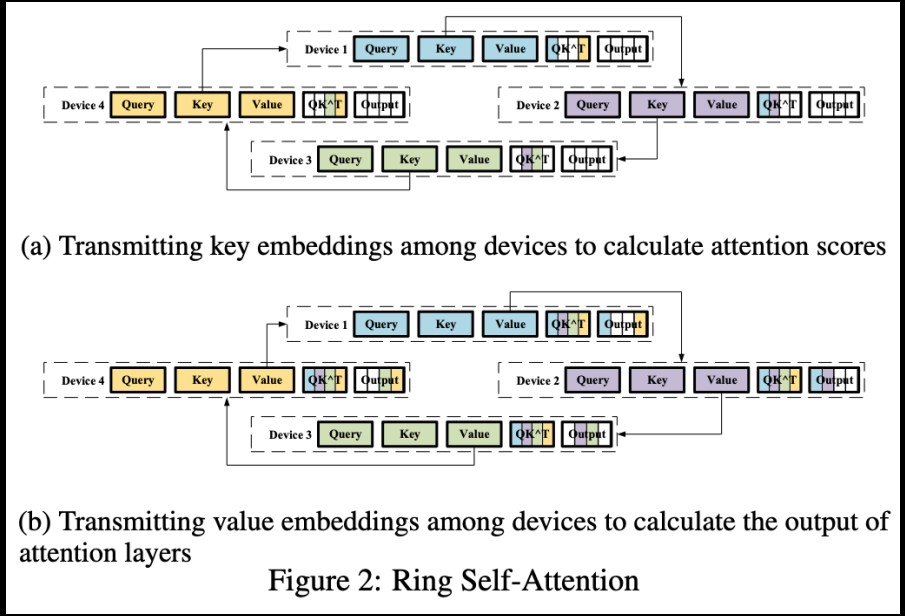


Hybrid Sharded Data Parallel

- A good compromise
- Memory footprint stays small
  - X → X/8 is a lot of freed memory (DDP → HSDP)
  - X/8 → X/64 is only a few GB (HSDP → FSDP)
- Improves communication costs by up to 3x in the best case
- More complex rank management (but PyTorch makes this invisible)

Sequence Parallel

- For long sequence lengths (>64k): shard sequence over devices
- For attention, stream keys/values in chunks through devices (e.g. ring-allreduce)
- Can still overlap comms and compute



Source: *Sequence Parallelism: Long Sequence Training from System Perspective* (2021)

Model Parallel

Pass around the inputs, not the model

- Not actually that big a difference
- Activations and model weights scale together

Model weights:

4096 (input dim)

$\times$ 4096 (output dim)

$\times$ 2 (attn + mlp layer)

$\times$ $[4,8]$ (inner expansion)

Activations:

4096 (input dim)

$\times$ 4096 (sequence length)

$\times$ $[64,512]$ (batch size)

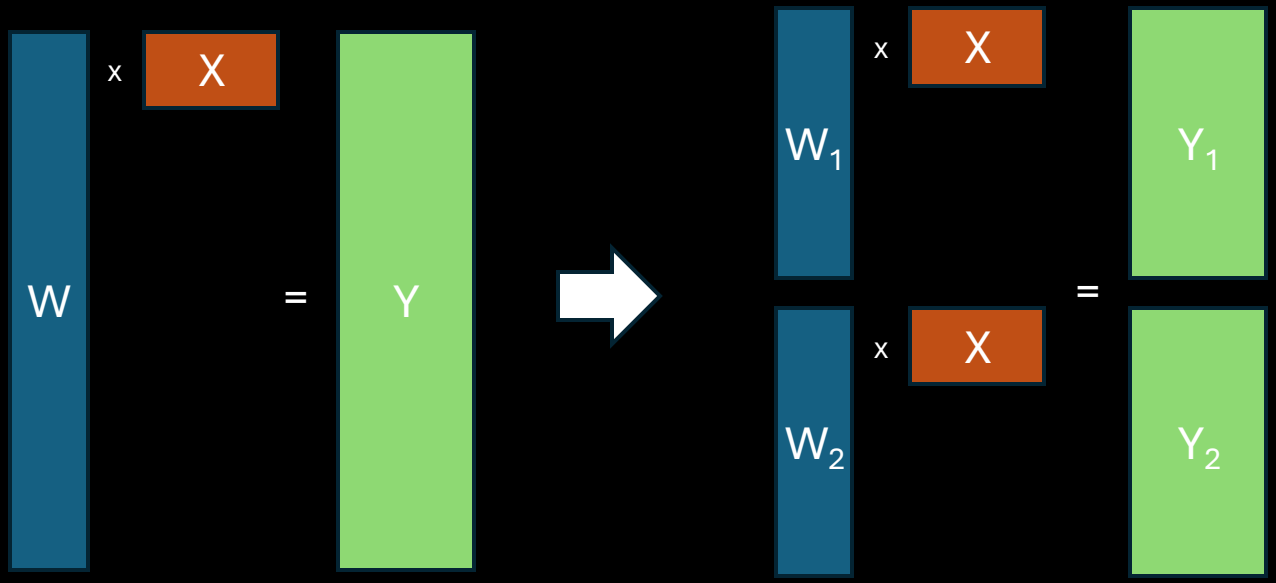
No more overlapping comms and compute!

IBM

SIG METRICS acm

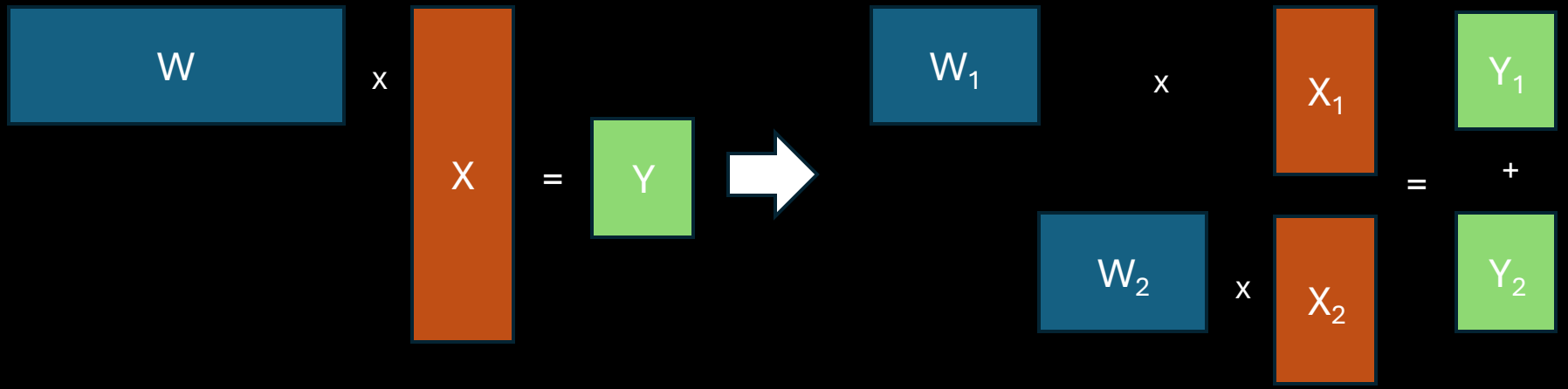
Tensor Parallel

Row-wise: every row (and row group) of matmul is independent



Tensor Parallel

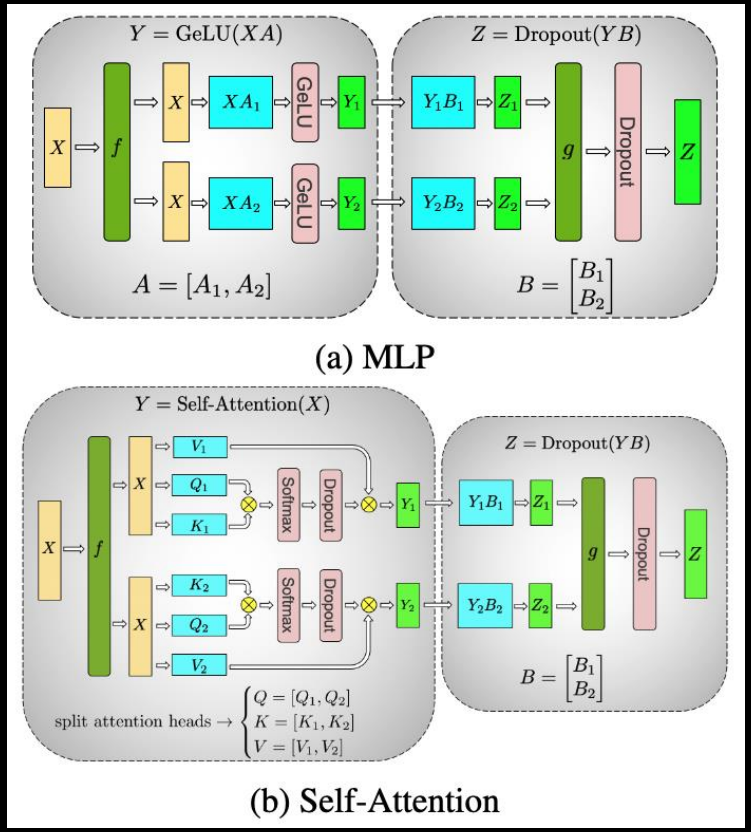
Column-wise: compute partial sums and reduce later



Tensor Parallel

Transformer layers project outward, perform some element-wise or group-wise transformation, then project back...

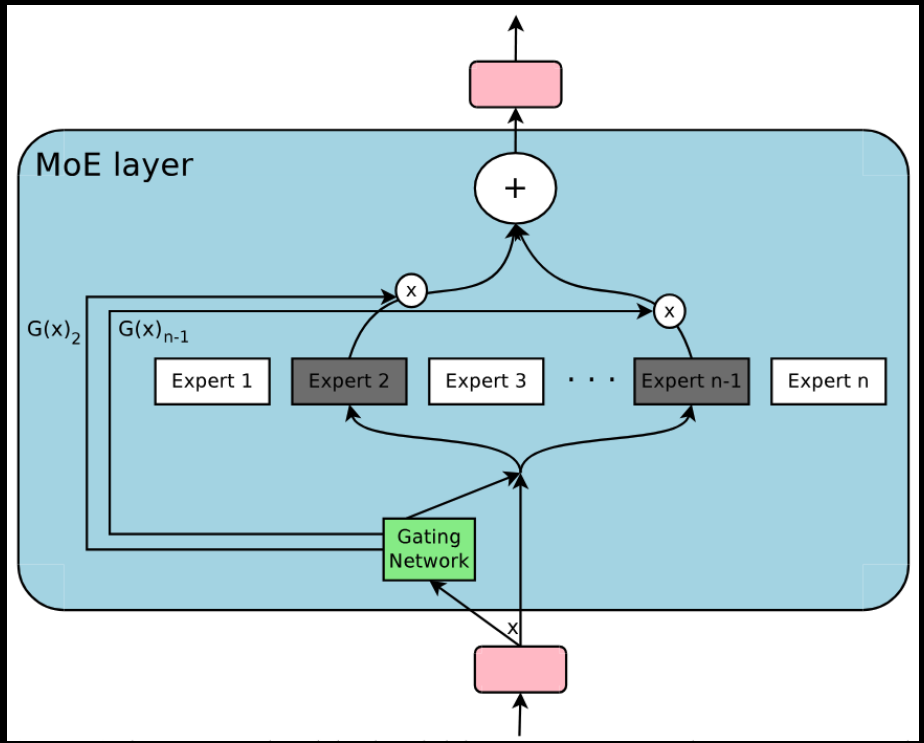
...which is perfect for row-wise then column-wise TP



Source: *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism* (2019)

Expert Parallel

- Blow up MLP width by ~20x
- Partition into heads/experts
- Route tokens to top-k experts (on different devices)
- Manually balance expert/device loads



Source: *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer* (2017)

Expert Parallel

Model weights:

4096 (input dim)

$\times\ 2048$ (expert dim)

$\times\ 256$ (n_experts)

$\times\ 2$ (in + out proj)

$= 4B$ (weights)

Activations:

4096 (input dim)

$\times\ 4096$ (sequence length)

$\times\ [64, 512]$ (batch size)

$\div\ (\frac{256}{4})$ (n_experts / top_k)

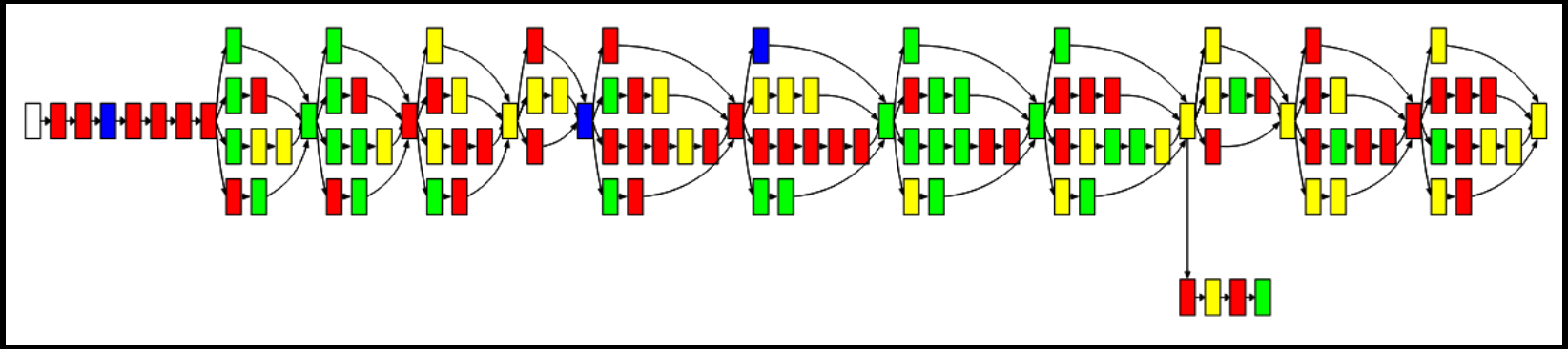
$=\ [16M, 134M]$ (activations)

IBM

SIG METRICS acm

Pipeline Parallel

- Parallelize over depth: shard layers over devices
- Communications schedule VERY non-homogenous

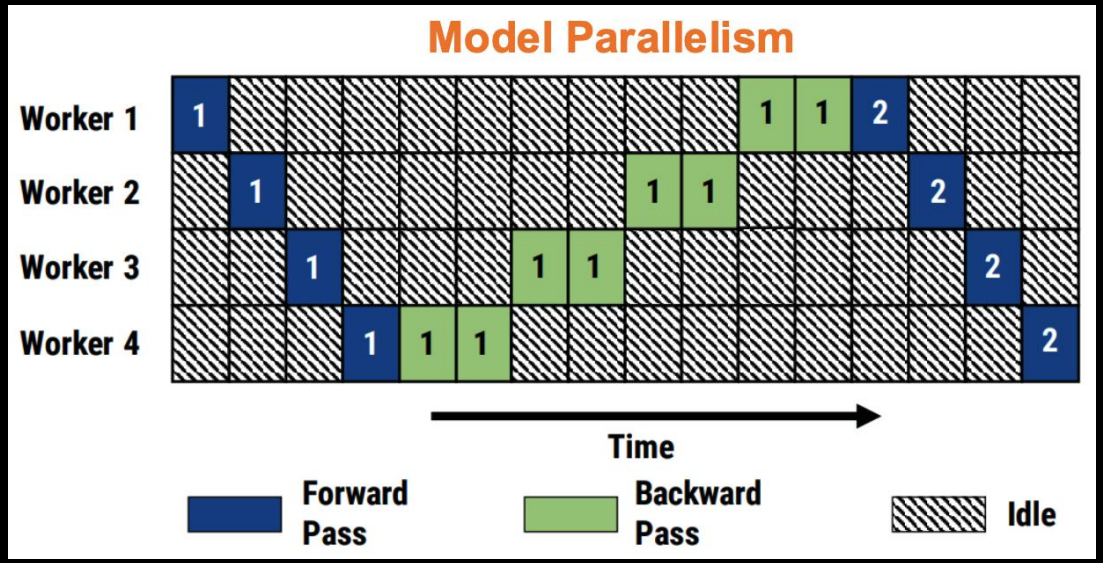


Source: *Device Placement Optimization with Reinforcement Learning* (2017)

Pipeline Parallel

Naïve implementation is very inefficient

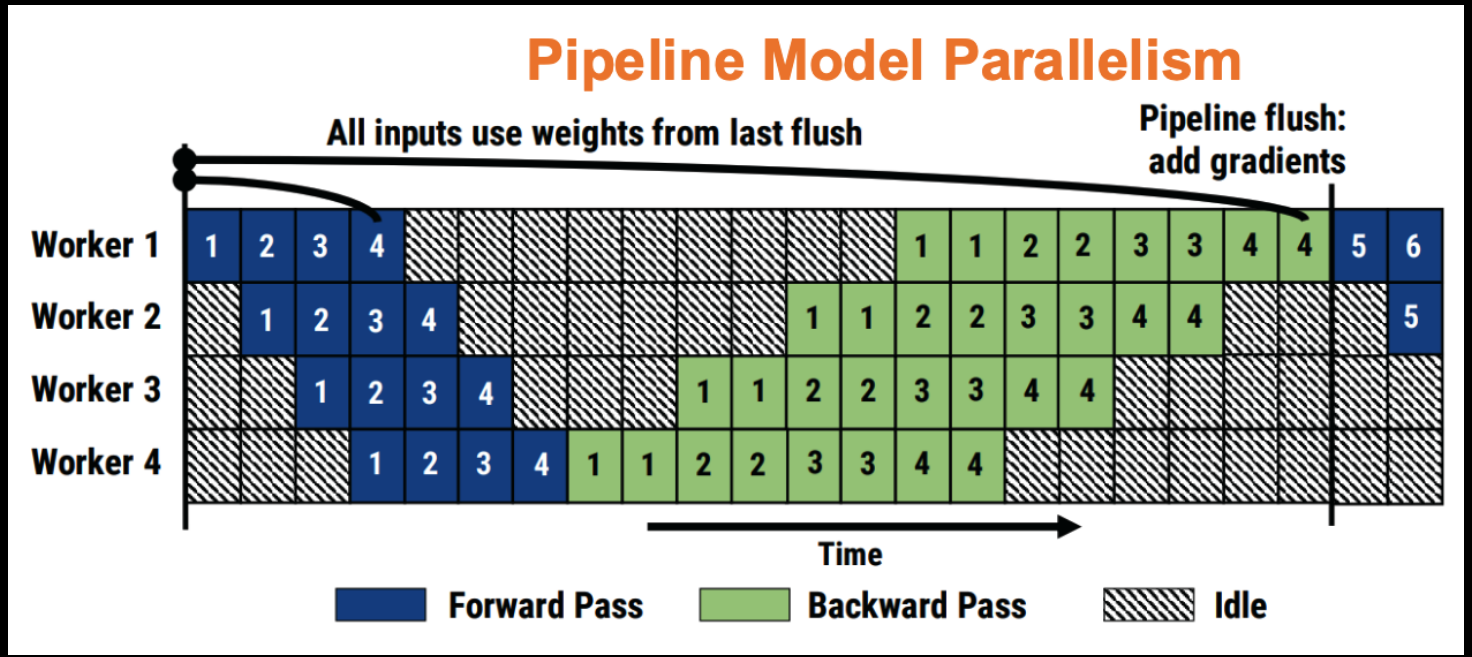
$$WX = Y, \qquad L = f(Y), \qquad \frac{dW}{dL} = X(\frac{dY}{dL})^T, \qquad \frac{dX}{dL} = W^T \frac{dY}{dL}$$



Source: https://www.cs.cmu.edu/~15418/lectures/25-parallel_deep_learning_model_pipeline_parallel.pdf

Pipeline Parallel

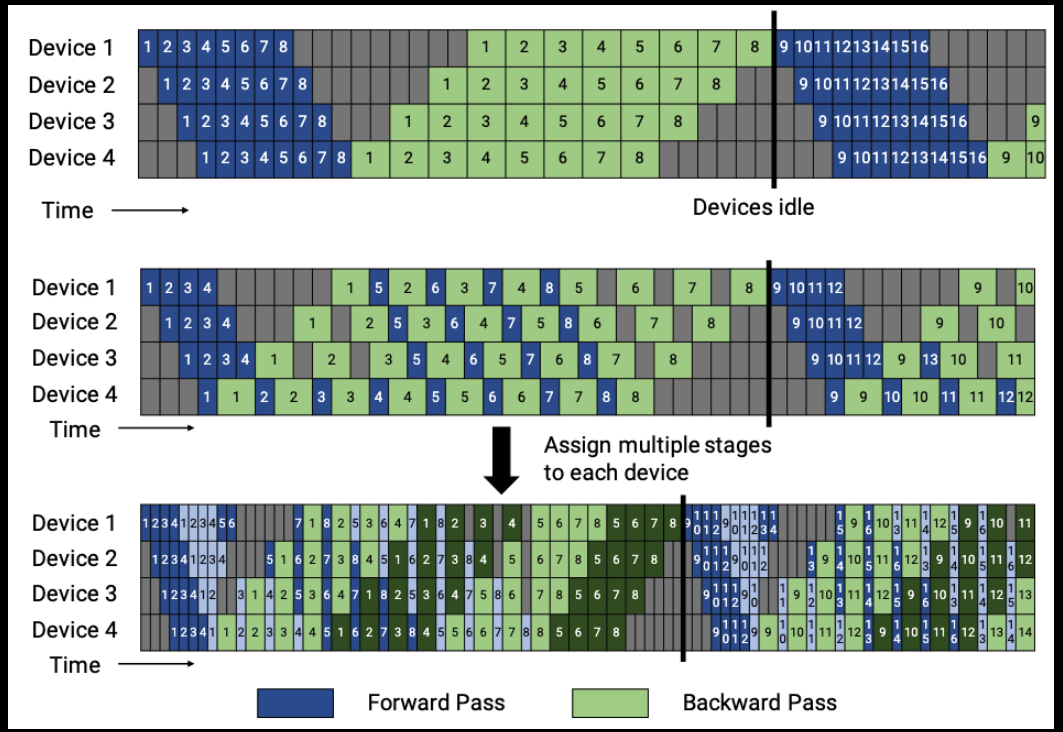
Aggregating mini-batches shrinks, but doesn't eliminate, bubbles



Source: https://www.cs.cmu.edu/~15418/lectures/25-parallel_deep_learning_model_pipeline_parallel.pdf

Pipeline Parallel

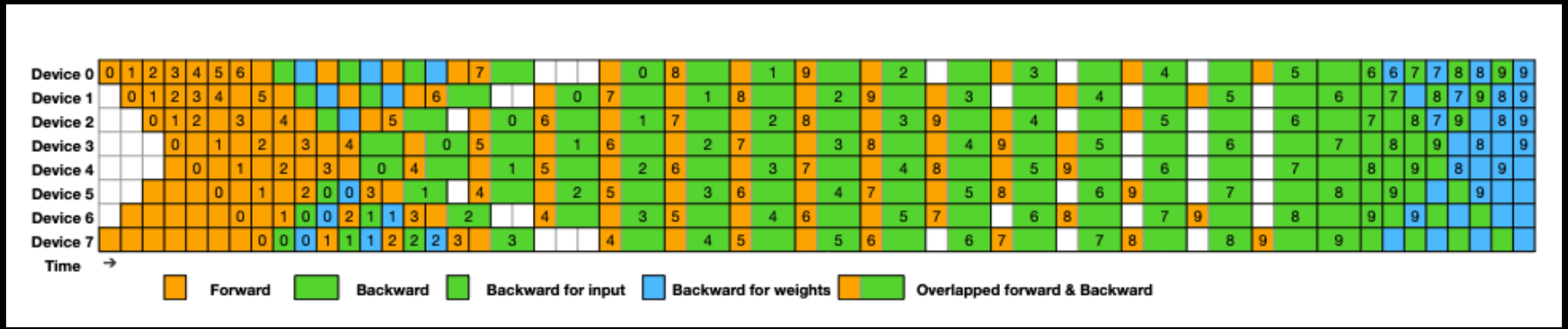
Lots of work on optimizing schedules



Source: *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism* (2019)

Pipeline Parallel

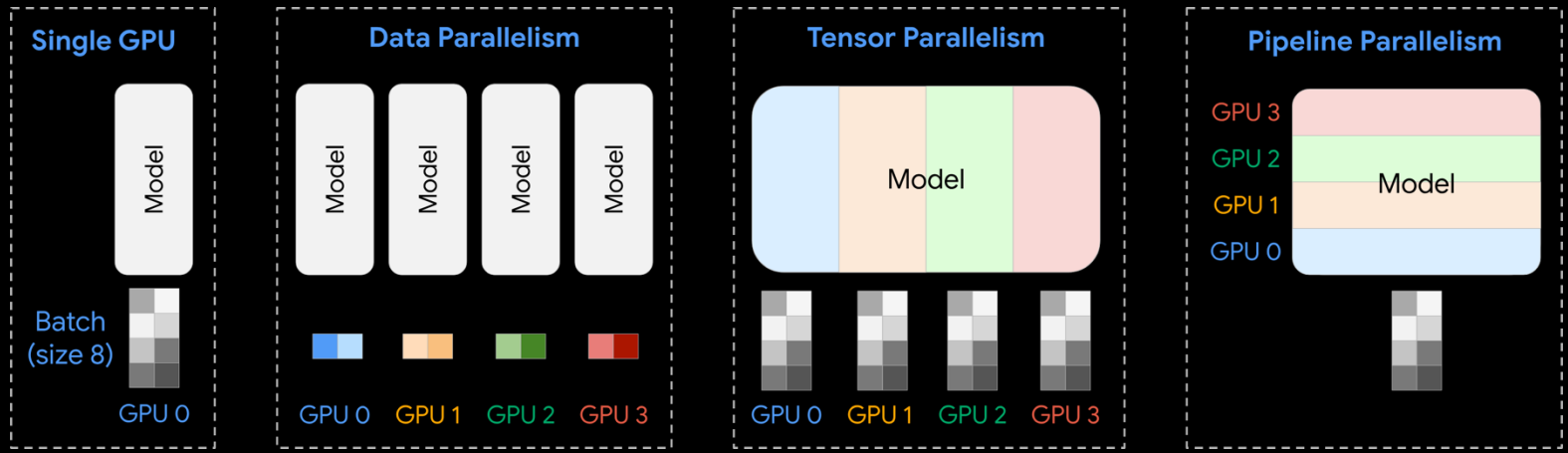
Lots of work on optimizing schedules



Source: *DeepSeek V3 Technical Report* (2025)

N-Dimensional Parallelism

- Data parallel (DDP, HSDP/FSDP, SP)
- Model parallel (TP/EP, PP)
- A safe bet – avoid scaling any one axis too far
- But good luck debugging it!



Source: *The Road to El Dorado* (2000)
Source: *https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/scaling/JAX/tensor_parallel_simple.html*

Contents

IBM

SIG METRICS acm

Handling Data at Scale

>2TB datasets require careful handling…

…even before multiple-device parallelism gets involved

Handling Data at Scale

>2TB datasets require careful handling…

…even before multiple-device parallelism gets involved

- Data parallel:
  - Sharding/partitioning (probably dynamic!)
  - Fractional file ownership?
  - How to handle shuffling?

Handling Data at Scale

>2TB datasets require careful handling…

…even before multiple-device parallelism gets involved

- Data parallel:
  - Sharding/partitioning (probably dynamic!)
  - Fractional file ownership?
  - How to handle shuffling?
- Tensor parallel:
  - Synchronized retrieval across devices
  - Shared-memory storage of several TB

Handling Data at Scale

>2TB datasets require careful handling…

…even before multiple-device parallelism gets involved

- Data parallel:
  - Sharding/partitioning (probably dynamic!)
  - Fractional file ownership?
  - How to handle shuffling?
- Tensor parallel:
  - Synchronized retrieval across devices
  - Shared-memory storage of several TB
- Sequence parallel:
  - Partitioning AND synchronization

IBM

SIG METRICS

Handling Data at Scale

>2TB datasets require careful handling…

…even before multiple-device parallelism gets involved

- Data parallel:
  - Sharding/partitioning (probably dynamic!)
  - Fractional file ownership?
  - How to handle shuffling?
- Tensor parallel:
  - Synchronized retrieval across devices
  - Shared-memory storage of several TB
- Sequence parallel:
  - Partitioning AND synchronization
- Pipeline parallel:
  - Only particular devices (top/bottom) need data
  - Single-device storage of several TB

IBM

SIG METRICS

Handling Data at Scale

>2TB datasets require careful handling…

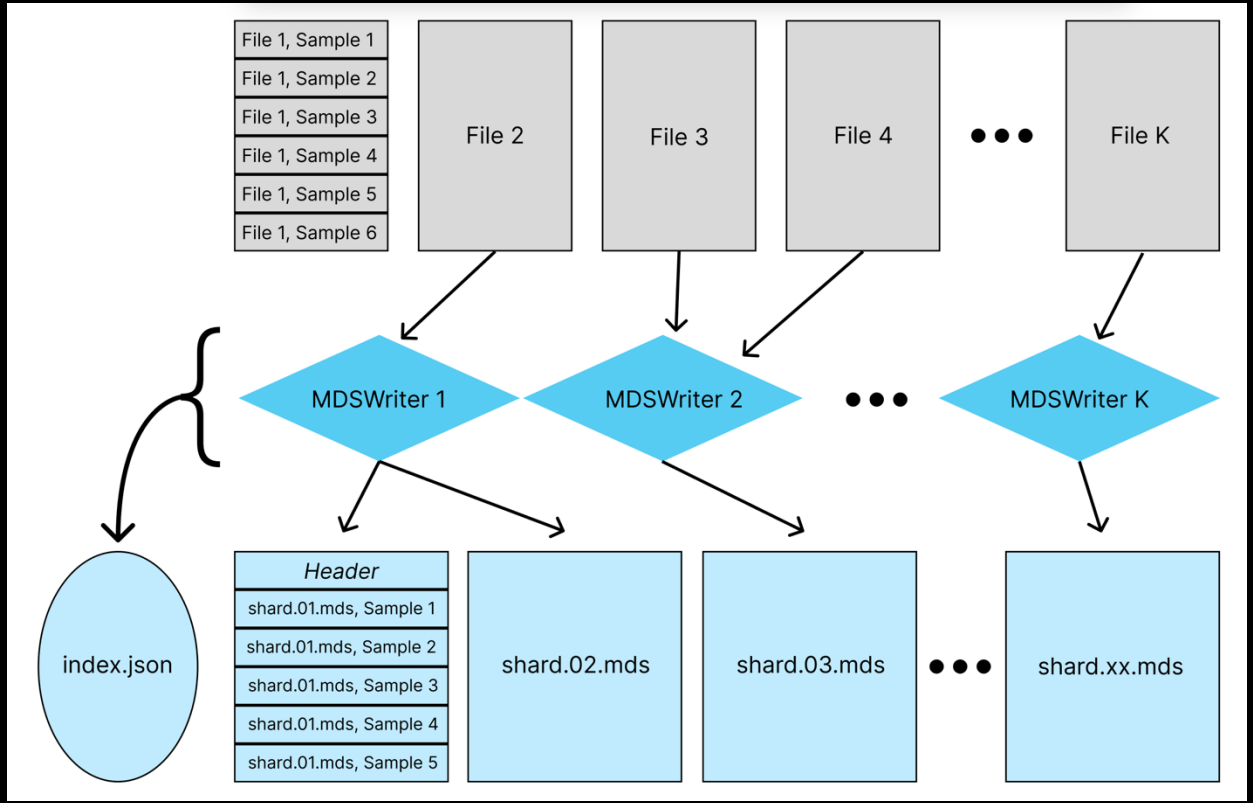
…even before multiple-device parallelism gets involved

- Data parallel:
  - Sharding/partitioning (probably dynamic!)
  - Fractional file ownership?
  - How to handle shuffling?
- Tensor parallel:
  - Synchronized retrieval across devices
  - Shared-memory storage of several TB
- Sequence parallel:
  - Partitioning AND synchronization
- Pipeline parallel:
  - Only particular devices (top/bottom) need data
  - Single-device storage of several TB
- N-dim parallel:
  - Have mercy

IBM

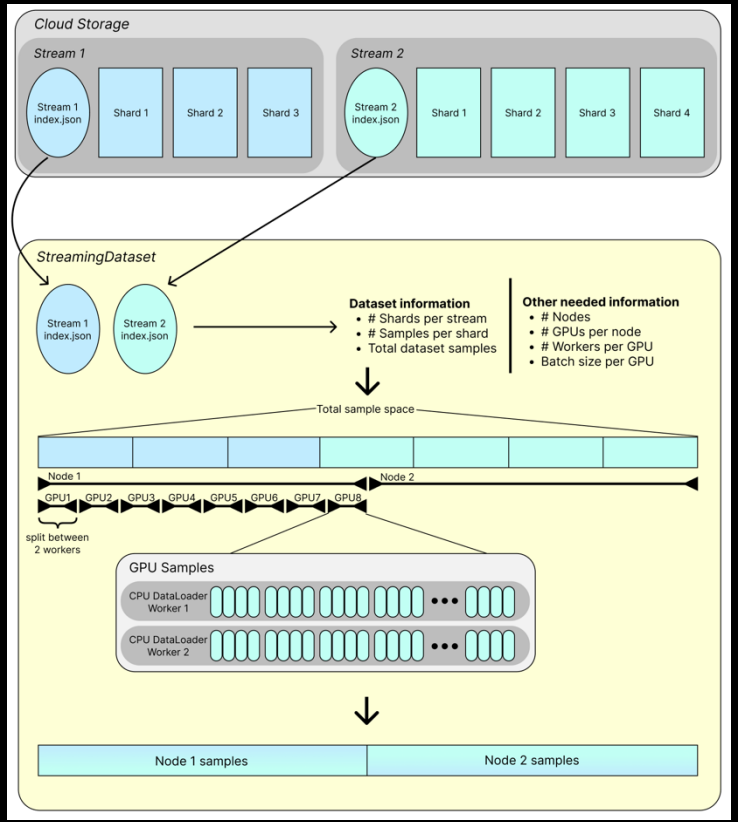
SIG METRICS

Handling Data at Scale



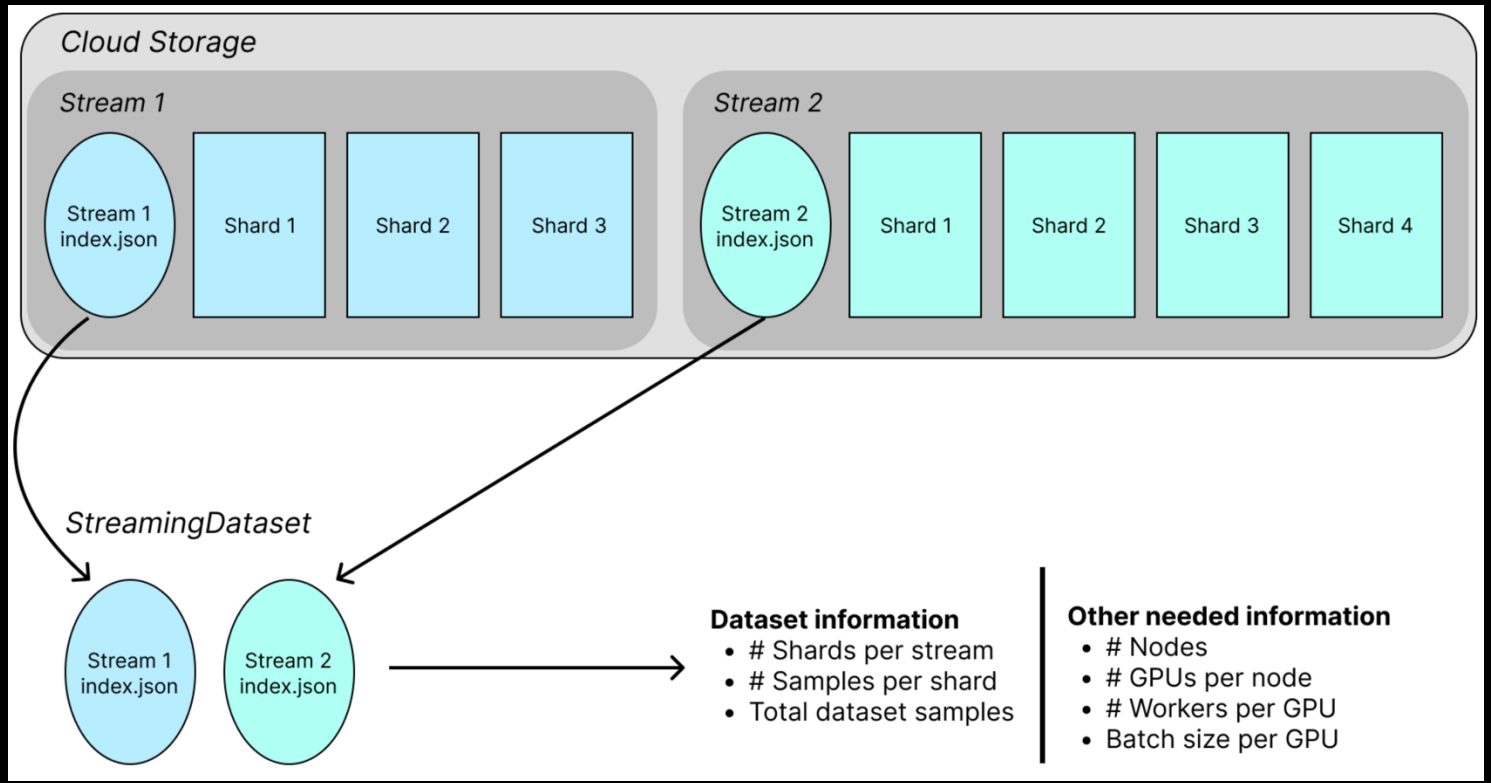
Source: *https://docs.mosaicml.com/projects/streaming/en/stable/getting_started/main_concepts.html*

Handling Data at Scale



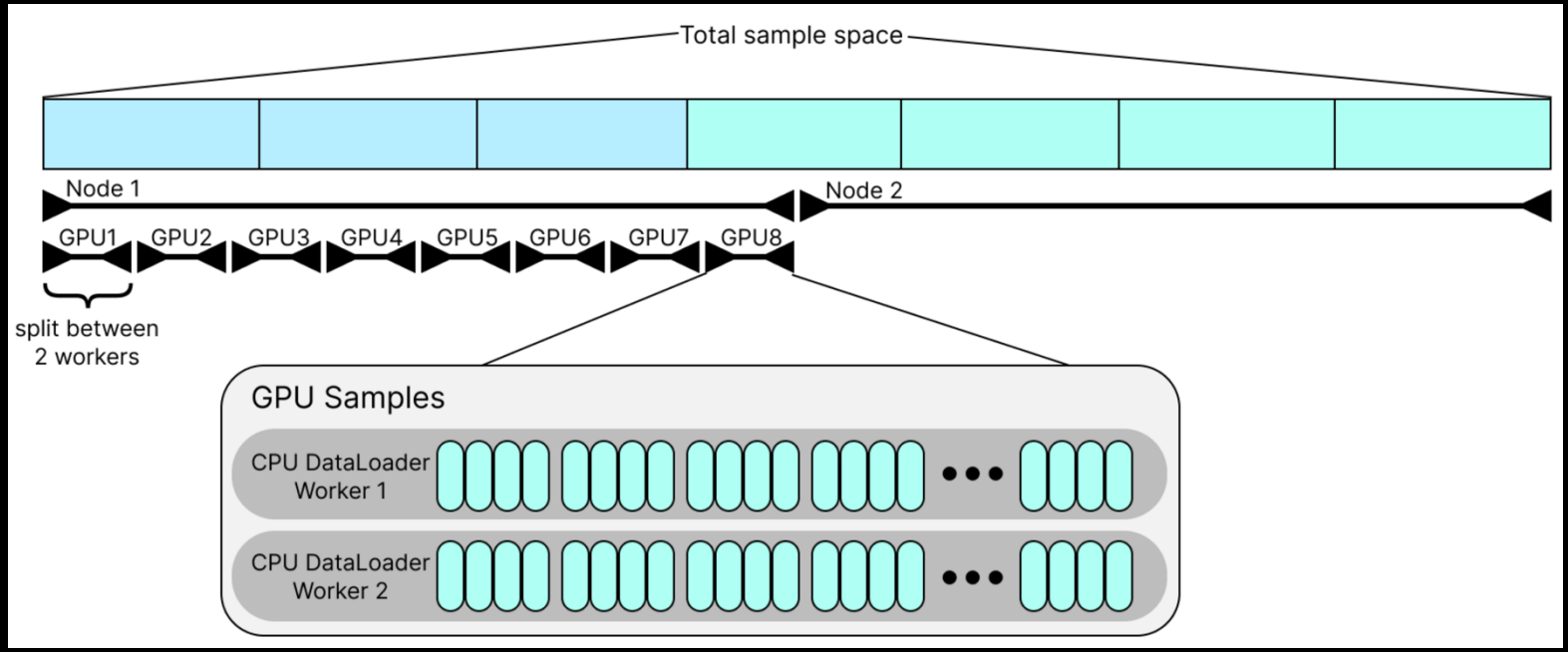
Source: *https://docs.mosaicml.com/projects/streaming/en/stable/getting_started/main_concepts.html*

Handling Data at Scale



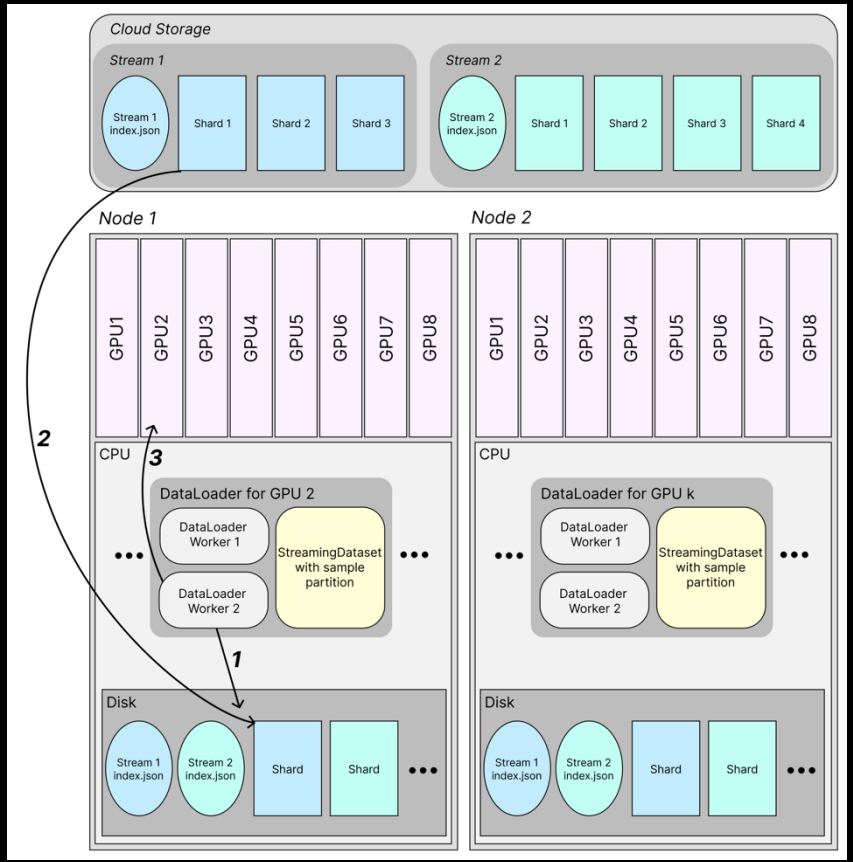
Source: *https://docs.mosaicml.com/projects/streaming/en/stable/getting_started/main_concepts.html*

Handling Data at Scale



Source: *https://docs.mosaicml.com/projects/streaming/en/stable/getting_started/main_concepts.html*

Handling Data at Scale



Source: *https://docs.mosaicml.com/projects/streaming/en/stable/getting_started/main_concepts.html*

Handling Data at Scale

- A whole separate problem behind-the-scenes
- General solution: mounted cloud storage folder
- A single master process is too slow
- Gets much worse for image data
- *(We'll revisit this in session 2)*

Contents

IBM

SIG METRICS acm

PyTorch's Approach

- Every device runs the same training script
- Model wrapper uses hooks to handle distributed compute
- Primarily data-parallel (FSDP good enough for most cases)
- TorchRun: auto-handling of necessary env vars

```
# FSDP
model = FSDP(
    model,
    auto_wrap_policy=wrapping_policy,
    mixed_precision=mixed_precision_policy,
    sharding_strategy=sharding_strategy_policy,
    use_orig_params=cfg.use_torch_compile,
    device_id=torch.cuda.current_device(),
    limit_all_gathers=True,
    param_init_fn=param_init_fn,
)
```

PyTorch's Approach

- Torchrun reference: https://docs.pytorch.org/docs/stable/elastic/run.html
- Torch.distributed and backend info: https://docs.pytorch.org/docs/stable/distributed.html
- FSDP (and caveats): https://docs.pytorch.org/docs/2.7/fsdp.html
- Getting started with FSDP: https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html
- FMS-FSDP (our codebase): https://github.com/foundation-model-stack/fms-fsdp

https://tinyurl.com/davis-sigm25

IBM

SIG METRICS

PART 3: Building an Optimized LLM Pretraining Platform

The Secret to Optimized Pretraining

What kind of work have you guys done? XXX is sending us extra bill for extra air conditioning… the work you're doing is seriously heating up the GPUs 🙂?

— ETE Team

(after constant 70°C+ and constant peak power usage for a couple months)

Source: *Kung Fu Panda* (2008)

The Secret to Optimized Pretraining

Last year, fms-fsdp broke the record (to our knowledge) for highest throughput open-source LLM pretraining platform. How?

- HSDP
- Compile
- Activation checkpointing
- Mixed precision
- Custom dataloader

None of these were particularly unique or novel!

IBM

SIG METRICS

The Secret to Optimized Pretraining

Last year, fms-fsdp broke the record (to our knowledge) for highest throughput open-source LLM pretraining platform. How?

- HSDP
- Compile
- Activation checkpointing
- Mixed precision
- Custom dataloader

None of these were particularly unique or novel!

Source: *Kung Fu Panda* (2008)

The Secret to Optimized Pretraining

Looking for the secret is counterproductive

because

You don't really take a scaled-up platform and optimize it

but instead

Optimize a small platform, then scale it up

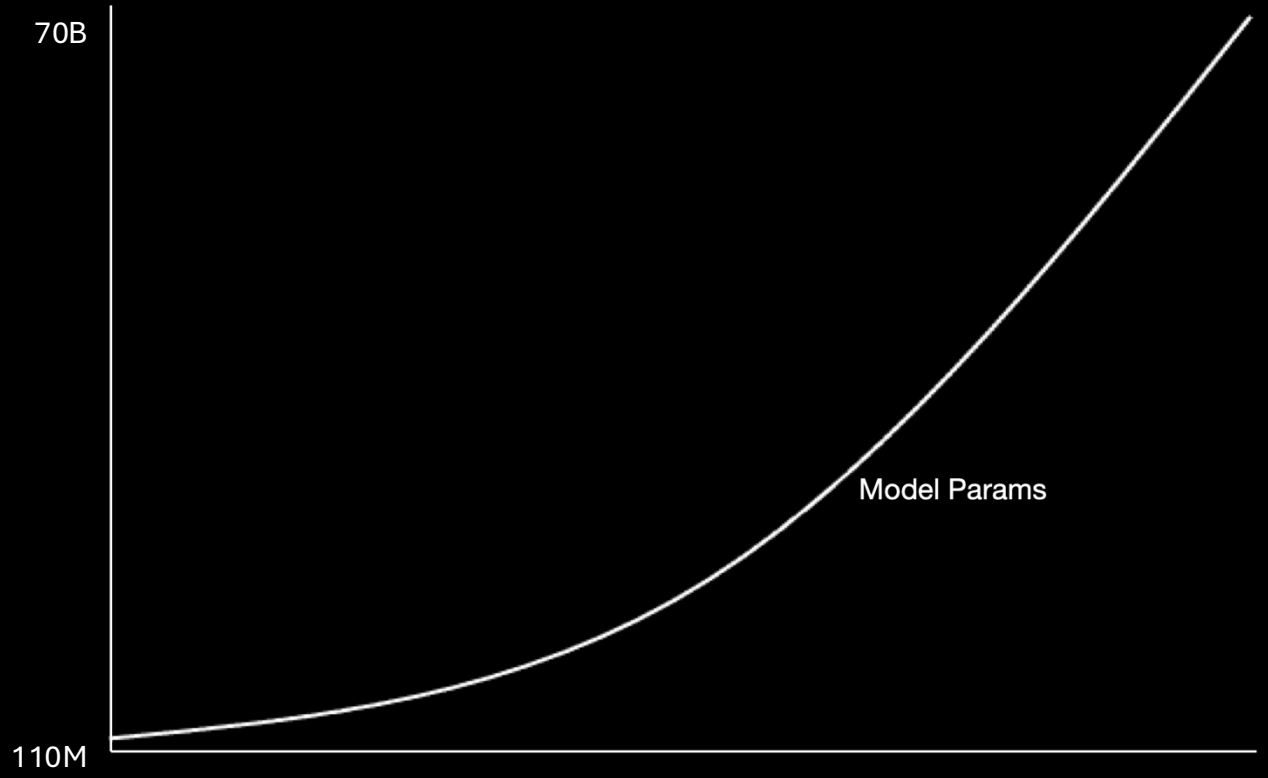
which is

A fundamentally multidimensional problem

IBM

SIG METRICS

Scaling: Number of Model Parameters

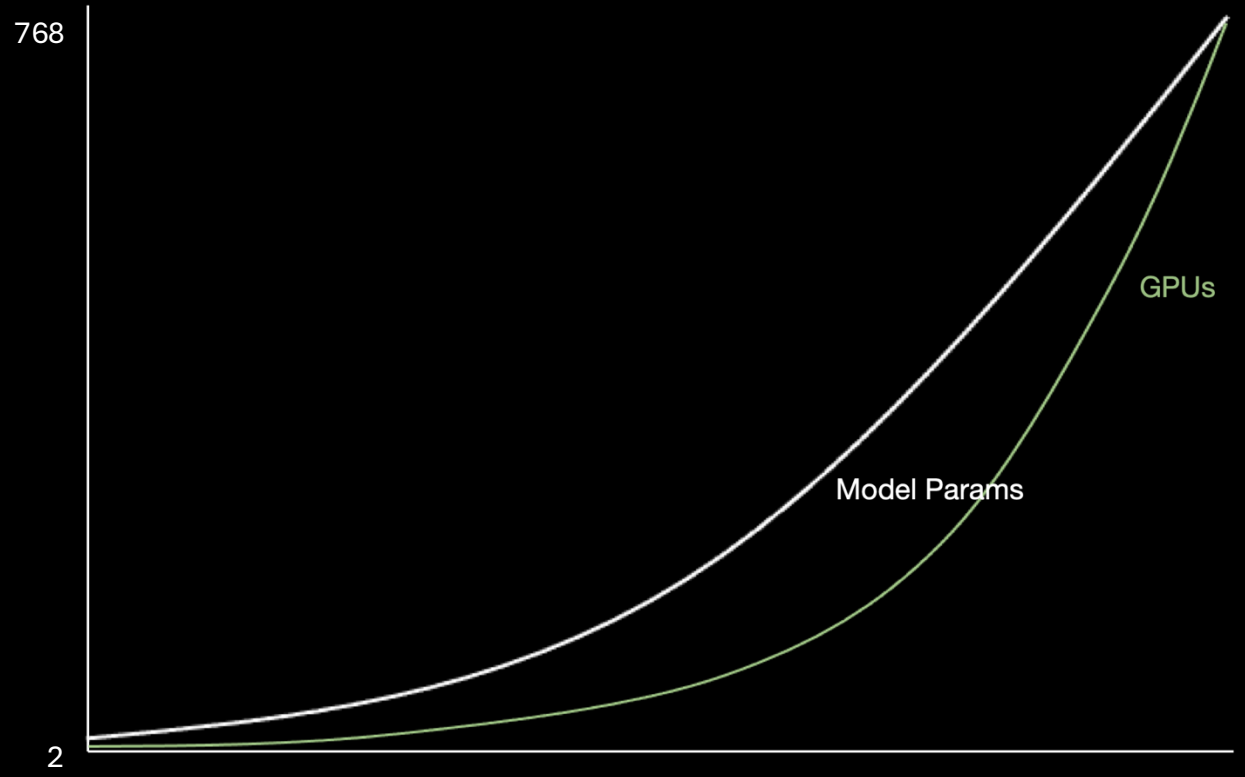


(y-axis not to scale)

IBM

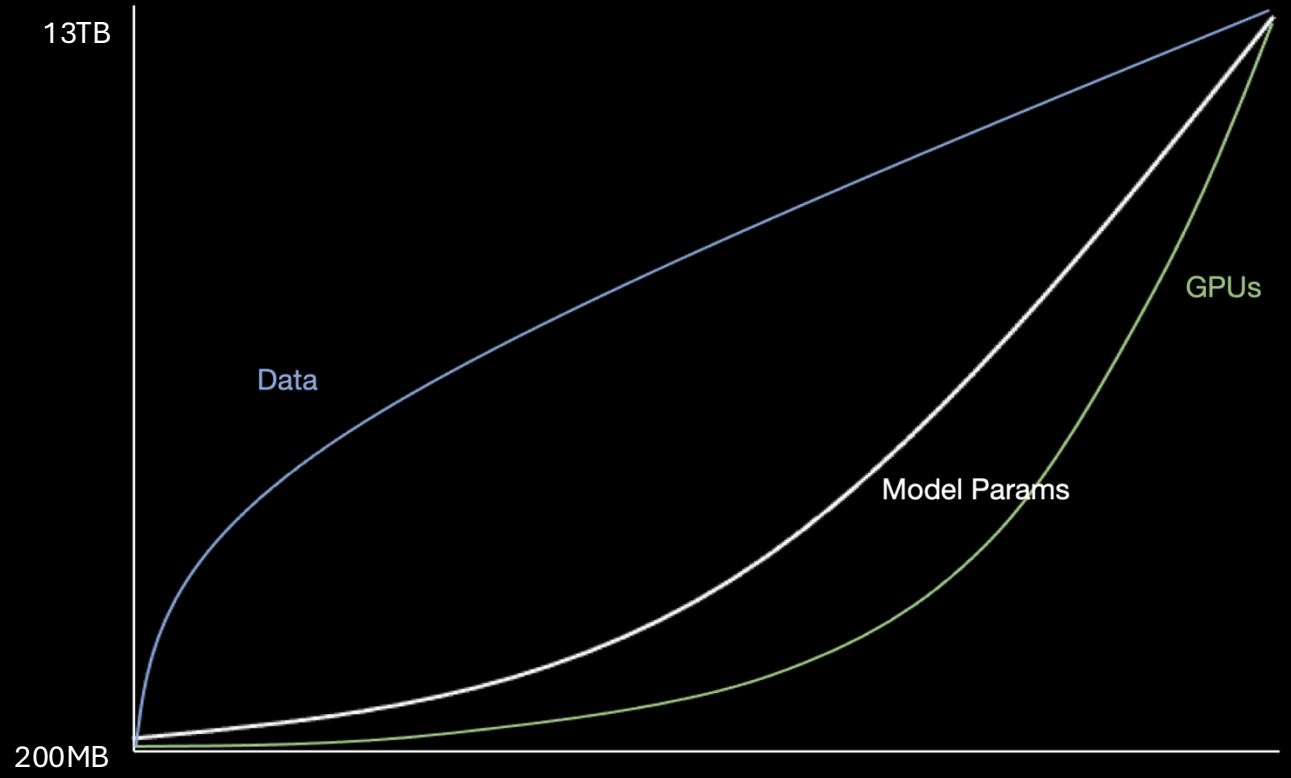
SIG METRICS

Scaling: Number of GPUs



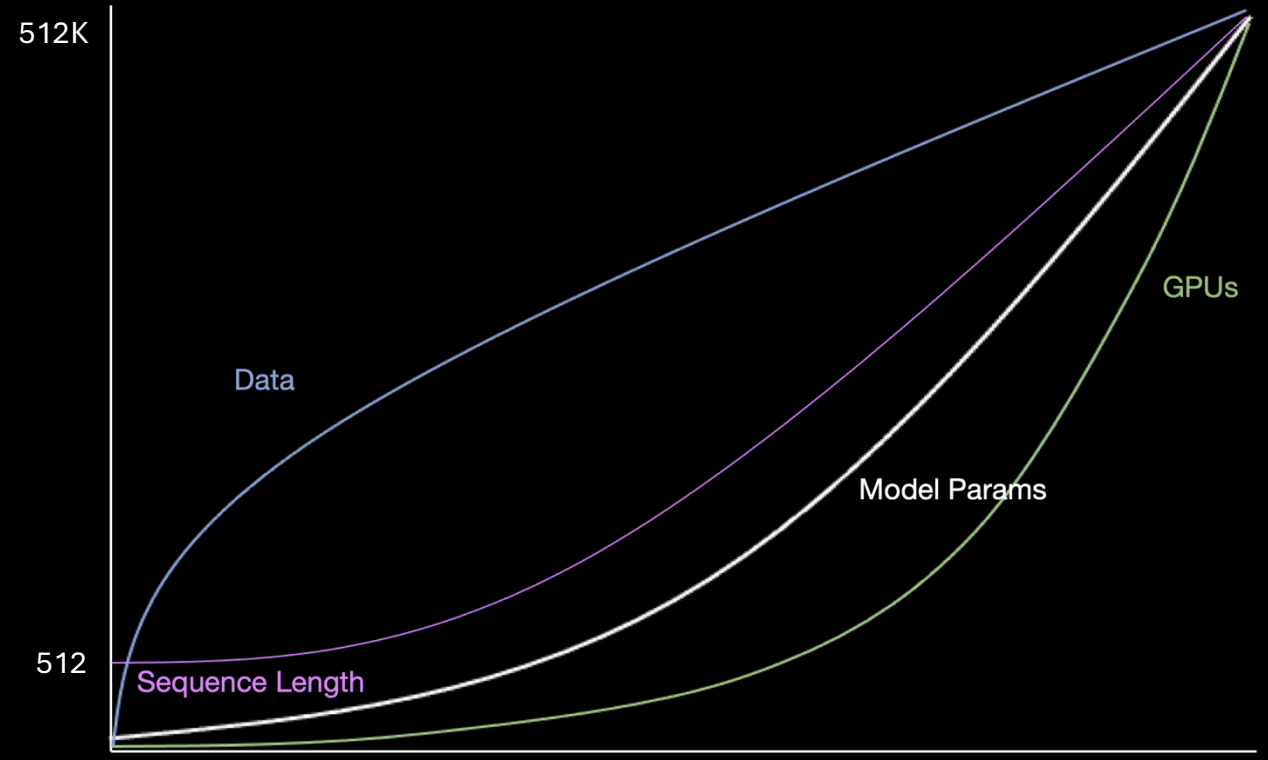
(y-axis not to scale)

Scaling: Amount of Data



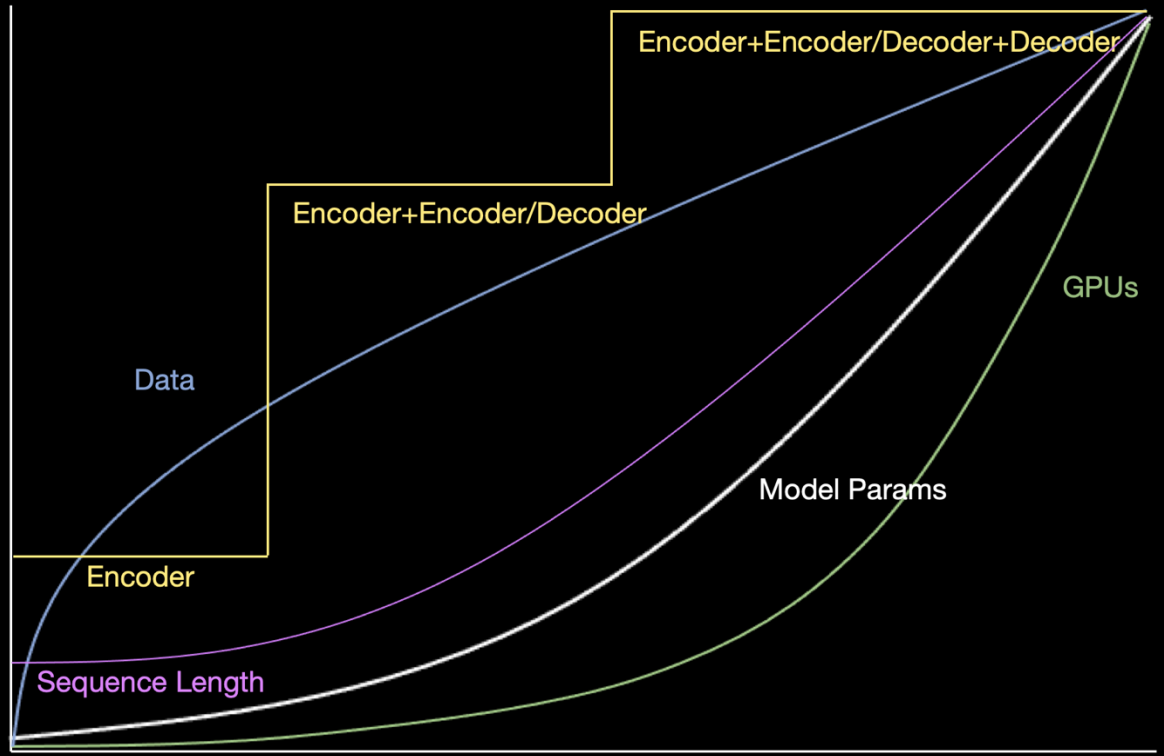
(y-axis not to scale)

Scaling: Sequence Length



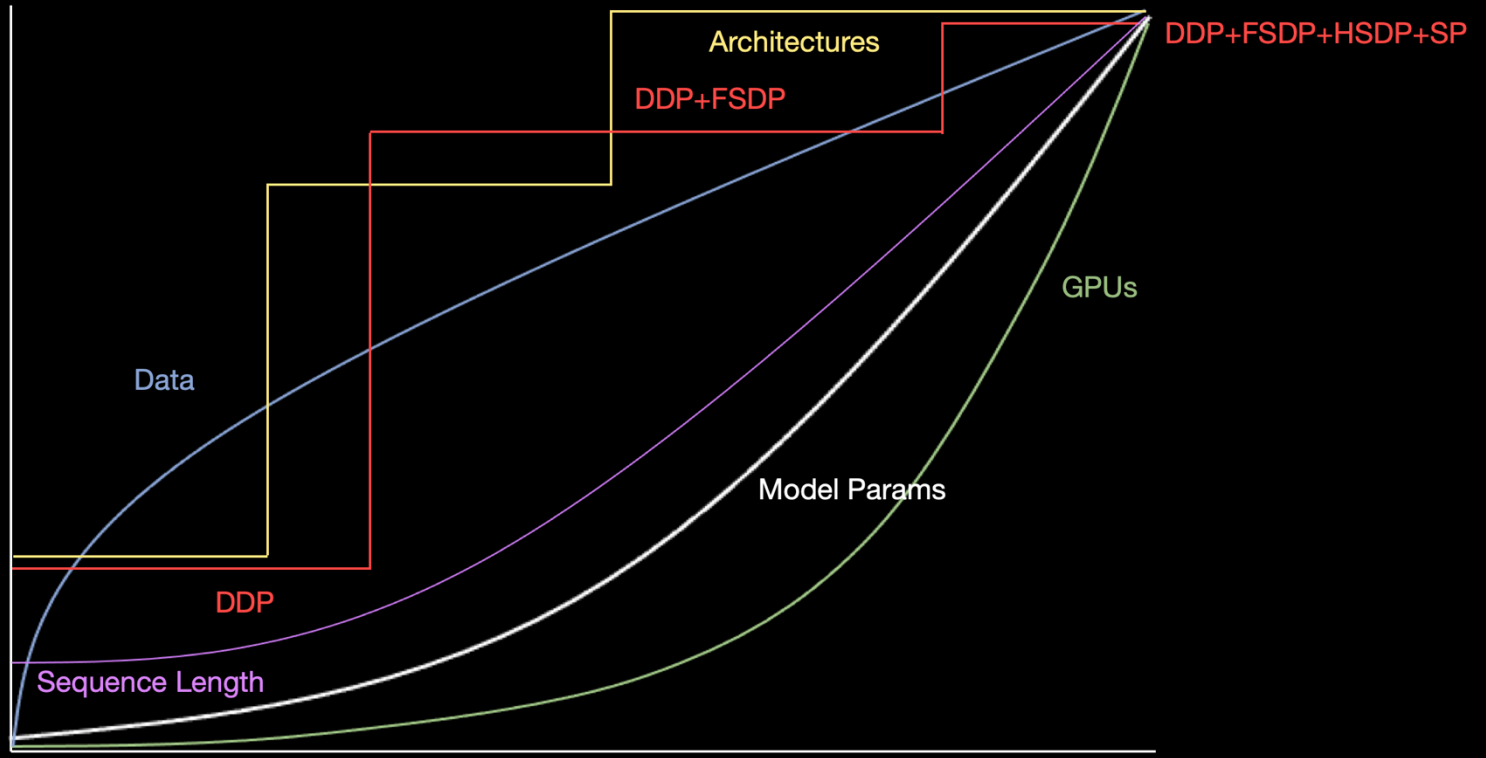
(y-axis not to scale)

Scaling: Supported Architectures



(y-axis not to scale)

Scaling: Parallelization Schemes

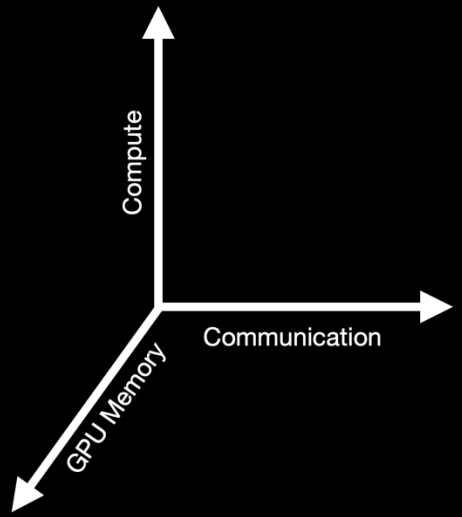


(y-axis not to scale)

A 3D View of Distributed Training

Design space defined along three axes:

- Compute
  - Massive datasets
  - Architecture design (e.g. Transformer parallelism)
  - Parallelization strategies (DDP / TP / PP / SP / …)
- GPU communication overhead
  - Multi-node clusters
  - Infiniband, Ethernet, RoCE, …
- GPU memory pressure
  - Model size, sequence length, batch size
  - bf16 Llama-33B fits in 80GB “reasonably”



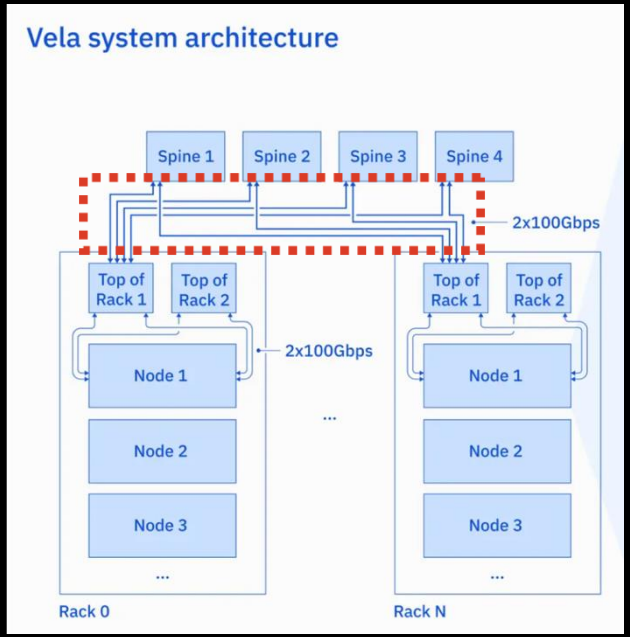
IBM

SIG METRICS

A 3D View of Distributed Training

Design space defined along three axes:

- Compute
  - Massive datasets
  - Architecture design (e.g. Transformer parallelism)
  - Parallelization strategies (DDP / TP / PP / SP / …)
- GPU communication overhead
  - Multi-node clusters
  - Infiniband, Ethernet, RoCE, …
- GPU memory pressure
  - Model size, sequence length, batch size
  - bf16 Llama-33B fits in 80GB “reasonably”



Source: *https://research.ibm.com/blog/AI-supercomputer-Vela-GPU-cluster*

A 3D View of Distributed Training

Design space defined along three axes:

- Compute
  - Massive datasets
  - Architecture design (e.g. Transformer parallelism)
  - Parallelization strategies (DDP / TP / PP / SP / …)
- GPU communication overhead
  - Multi-node clusters
  - Infiniband, Ethernet, RoCE, …
- GPU memory pressure
  - Model size, sequence length, batch size
  - bf16 Llama-33B fits in 80GB “reasonably”

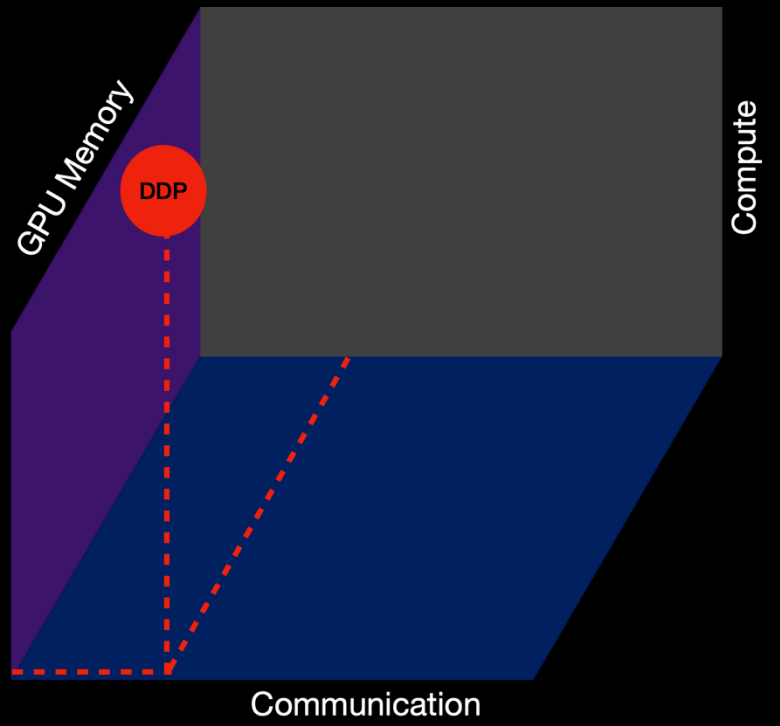
GPU	Memory
Tesla P100	16GB
Tesla V100	32GB, 16GB
Ampere A10	24GB
Ampere A100	80GB
Hopper H100	80GB, 120GB

Source: *https://github.com/cedrickchee/llama/blob/main/chattyllama/hardware.md#memory-requirements-for-each-model-size*

A 3D Design Space

We worked with PyTorch to establish and scale operating points in this design space

- DDP (Distributed Data Parallel)
  - Mature baseline
  - Replicate model, sync gradients
  - High memory/compute, low comm



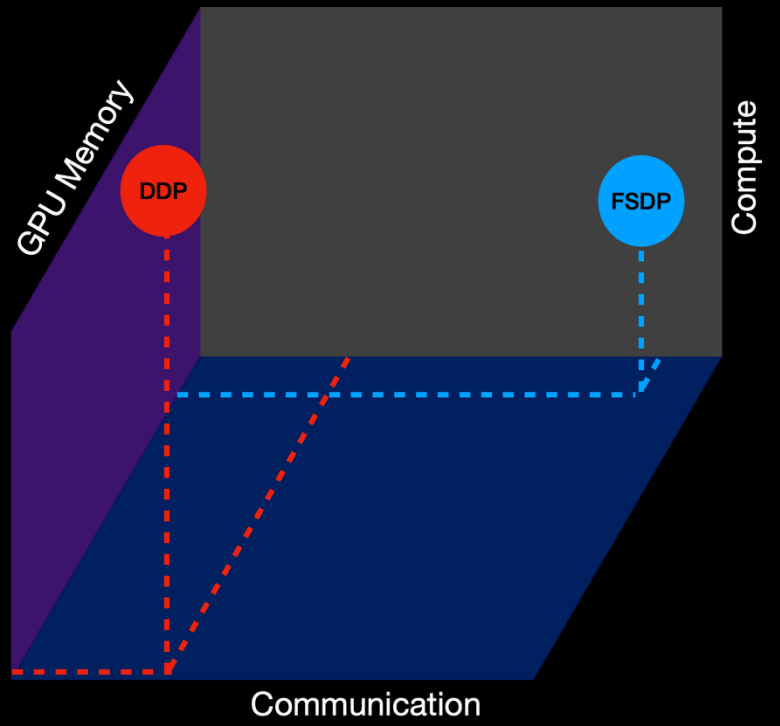
IBM

SIG METRICS

A 3D Design Space

We worked with PyTorch to establish and scale operating points in this design space

- DDP (Distributed Data Parallel)
- FSDP (Fully Sharded Data Parallel)
  - At the time, in beta
  - Split model, get weights on demand
  - 1.5x comms, much lower memory



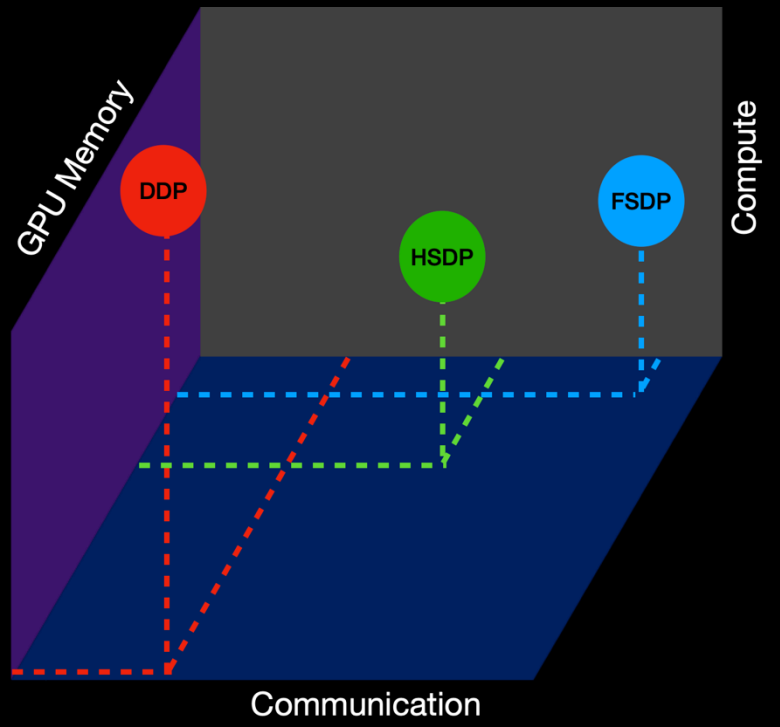
IBM

SIG METRICS

A 3D Design Space

We worked with PyTorch to establish and scale operating points in this design space

- DDP (Distributed Data Parallel)
- FSDP (Fully Sharded Data Parallel)
- HSDP (Hybrid Sharded Data Parallel)
  - At the time, in beta
  - Split model within node, duplicate across
  - Trade memory pressure for less comm



IBM

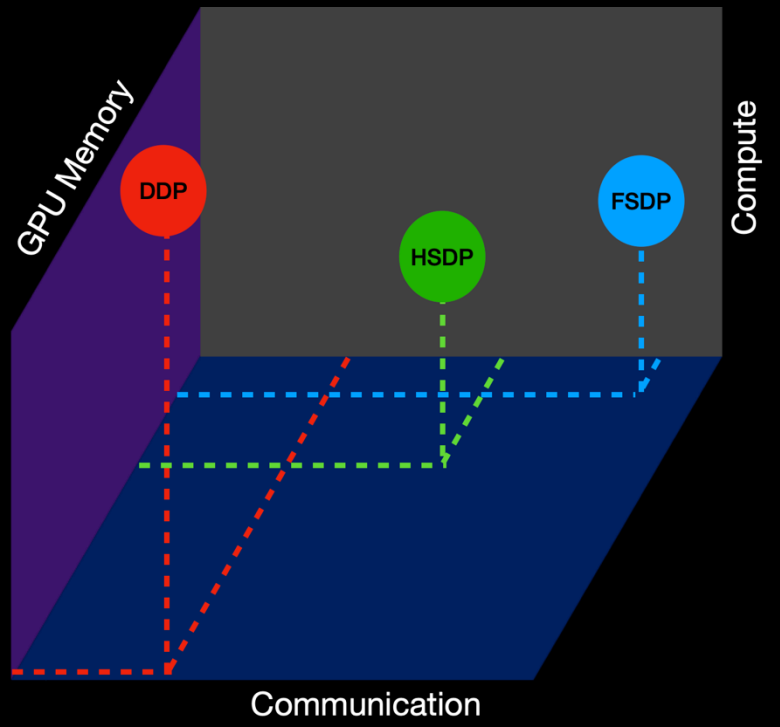
SIG METRICS

A 3D Design Space

We worked with PyTorch to establish and scale operating points in this design space

- DDP (Distributed Data Parallel)
- FSDP (Fully Sharded Data Parallel)
- HSDP (Hybrid Sharded Data Parallel)
- Tensor and Sequence Parallel when required (>30B params, >32k tokens)

Great! So are we ready to train 70B models?

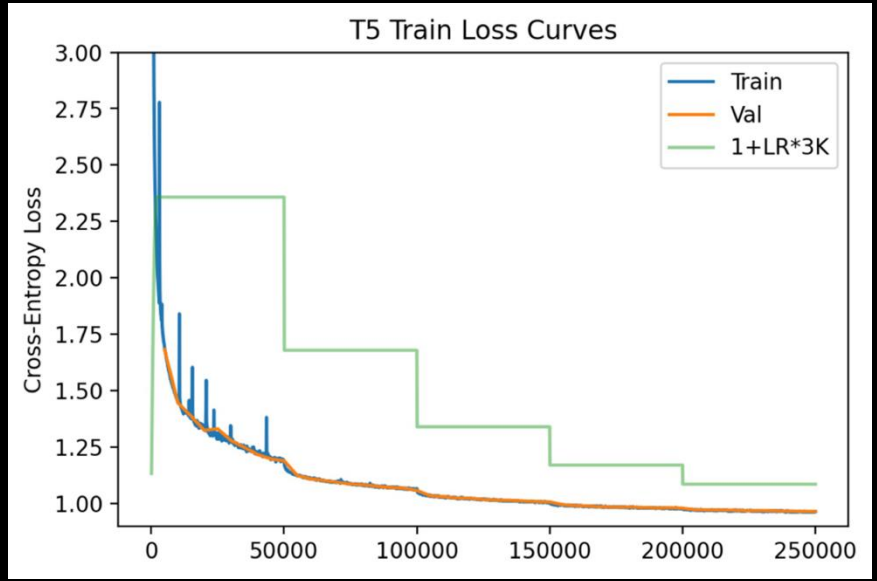


IBM

SIG METRICS

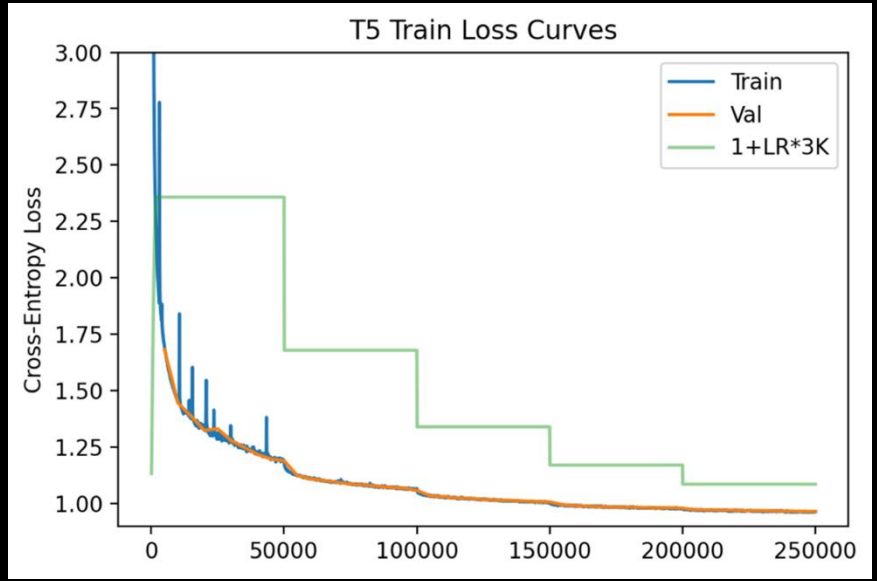
Stability Over Time

- Non-homogeneous workloads
  - FSDP/HSDP require tight synchronization of multiple moving parts
  - Inner validation loop caused hangs and memory fragmentation



Stability Over Time

- Non-homogeneous workloads
  - FSDP/HSDP require tight synchronization of multiple moving parts
  - Inner validation loop caused hangs and memory fragmentation



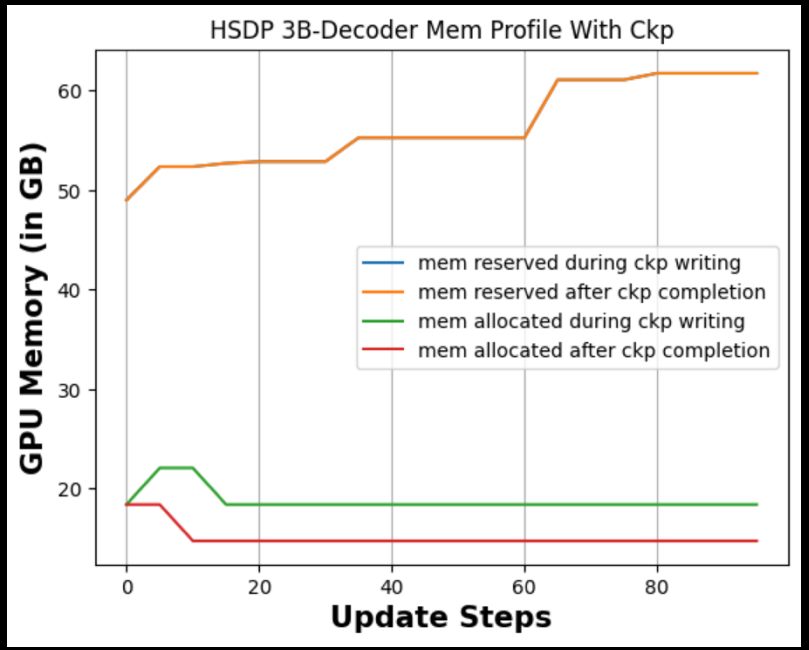
Fix: remove/externalize validation and other utilities

IBM

SIG METRICS

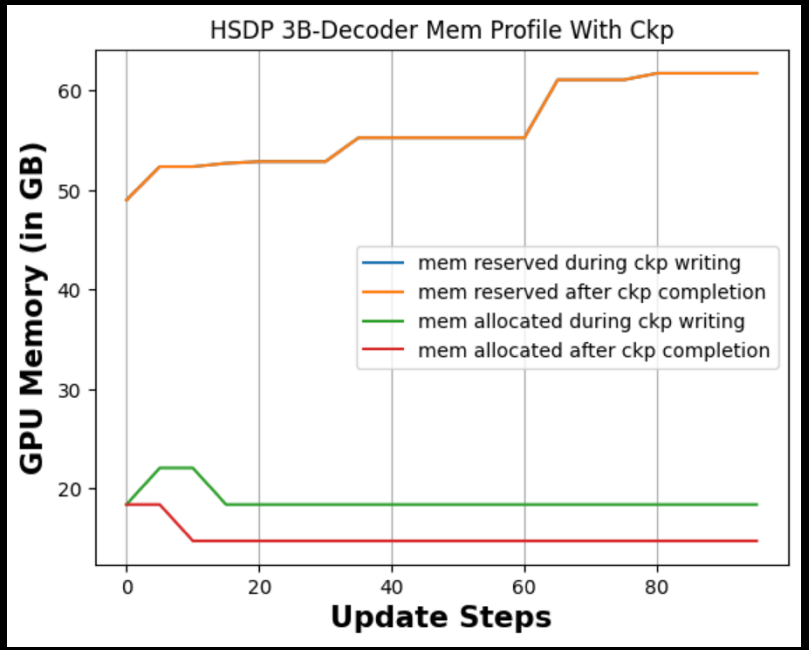
Stability Over Time

- Non-homogeneous workloads
- Checkpoint increasing memory
  - Fluctuations in reserved vs allocated cuda memory
  - Across devices and when saving checkpoints



Stability Over Time

- Non-homogeneous workloads
- Checkpoint increasing memory
  - Fluctuations in reserved vs allocated cuda memory
  - Across devices and when saving checkpoints



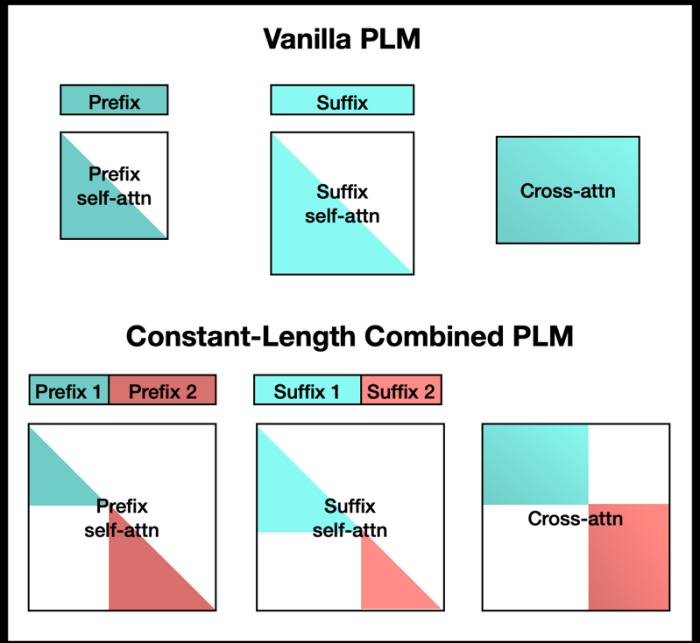
Fix: work with PyTorch to identify and expose the environment variable causing memory fragmentation

IBM

SIG METRICS

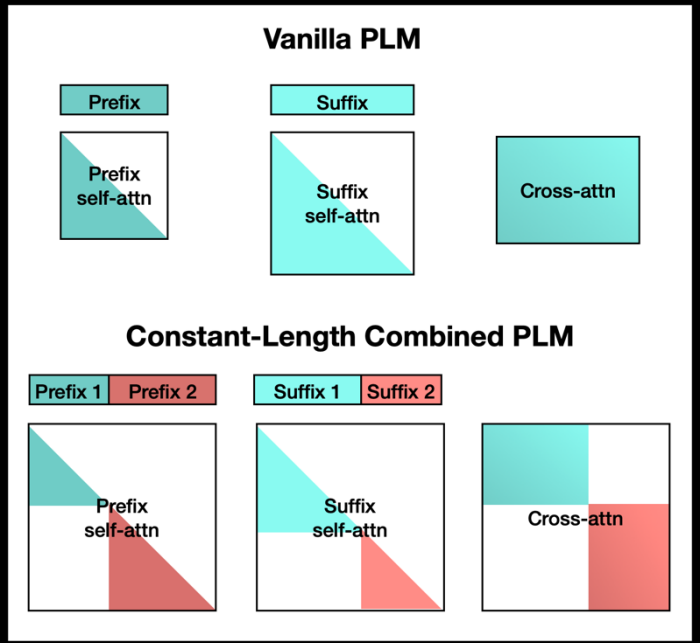
Stability Over Time

- Non-homogeneous workloads
- Checkpoint increasing memory
- Prefix Language Modeling
  - PLM: from beginning of sequence, predict the rest
  - Encoder-Decoder model
  - Variable prompt length: *RAPID* desynchronization



Stability Over Time

- Non-homogeneous workloads
- Checkpoint increasing memory
- Prefix Language Modeling
  - PLM: from beginning of sequence, predict the rest
  - Encoder-Decoder model
  - Variable prompt length: *RAPID* desynchronization



Fix: combine mirrored sequences and masks to keep tensor dimensions constant

IBM

SIG METRICS

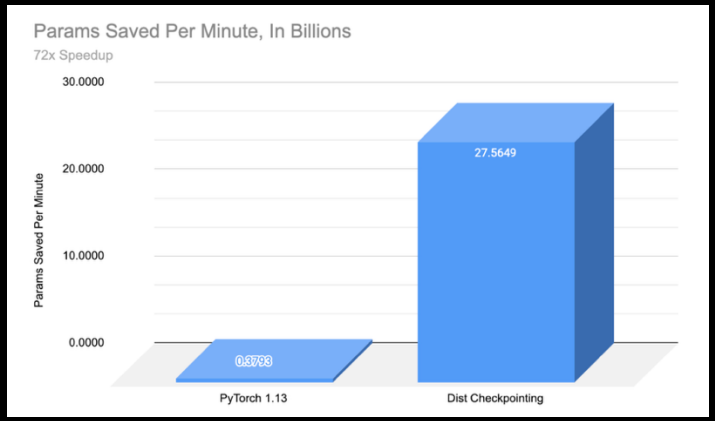
Stability Over Time

- Non-homogeneous workloads
- Checkpoint increasing memory
- Prefix Language Modeling
- FSDP is out of beta and much more stable now, but the point stands

Great! So *now* are we ready to train 70B models?

A Long Tail of Practical Challenges

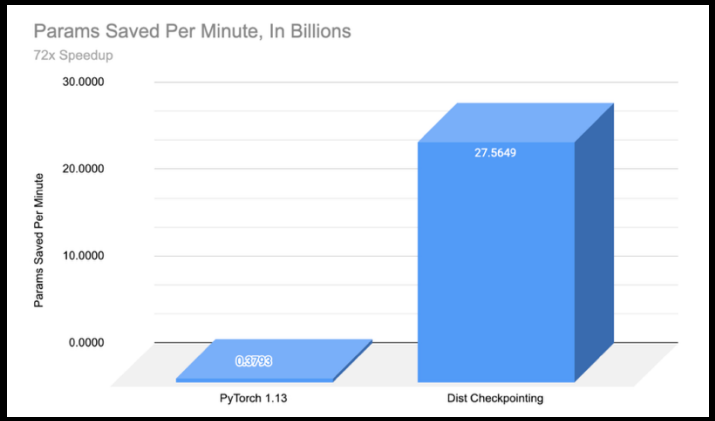
- Distributed model checkpointing
  - 7B checkpoint is ~81GB
  - Streaming that one file to cloud takes an hour!
    - 12hr interval: 8% slowdown
    - 8hr interval: 13% slowdown
    - 6hr interval: 17% slowdown
    - 4hr interval: 25% slowdown
    - … if you can write these at all



Source: *https://pytorch.org/blog/performant-distributed-checkpointing/*

A Long Tail of Practical Challenges

- Distributed model checkpointing
  - 7B checkpoint is ~81GB
  - Streaming that one file to cloud takes an hour!
    - 12hr interval: 8% slowdown
    - 8hr interval: 13% slowdown
    - 6hr interval: 17% slowdown
    - 4hr interval: 25% slowdown
    - … if you can write these at all
  - Fix: work with PyTorch on sharded checkpointing
    - Every worker writes 1/n of the checkpoint
    - Tweaks for cloud-based storage (eventual consistency)
    - A major release feature for PyTorch 2.1



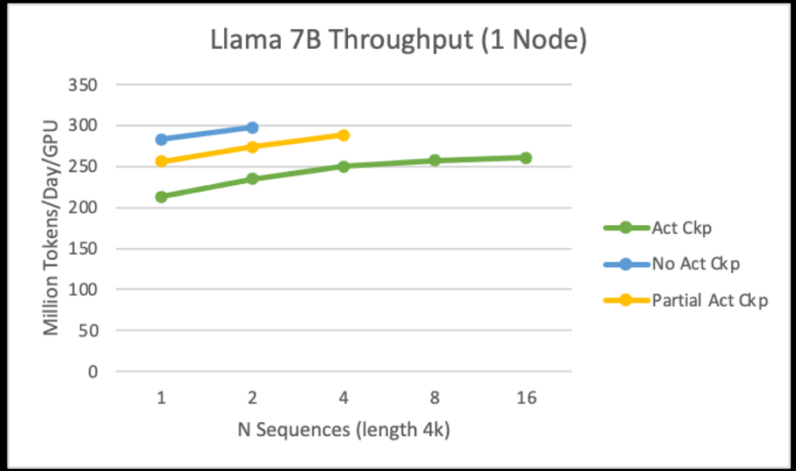
Source: *https://pytorch.org/blog/performant-distributed-checkpointing/*

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
  - Training is < 1 epoch: mid-epoch recovery necessary
  - Stateful
  - Distributed
  - Rescalable
  - Subdataset re-weighting
  - Modular/extensible

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
  - Recompute activations in backward pass (instead of storing)
  - Sophisticated fine-grained policies
  - Trade compute for memory
  - If comm << comp, you shouldn't use this! (use gradient accumulation instead)



IBM

SIG METRICS

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
  - Each cpu owns 1/n of the full dataset (will this fit in RAM?)
  - Streaming data loaders need extra:
    - File metadata for sharding
    - File ownership logic to minimize pulls
    - Shuffling and mixing steps

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
- CPU latency
  - Many sequential, unfused CUDA calls: gpu/cpu sync
  - Different hardward → up to 2x forward pass speed

IBM

SIG METRICS

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
- CPU latency
- Document size
  - Long documents with > 840M characters
  - Documents with length zero

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
- CPU latency
- Document size
- Batch size limits
  - More gpus → larger batches → less stochasticity → lower data efficiency
  - Above ~4M tokens, more gpus slows convergence!

IBM

SIG METRICS

A Long Tail of Practical Challenges

- Superimpose Earth's mountains
- What location has the highest average elevation?

Source: *https://en.wikipedia.org/wiki/Mount_Everest#/media/File:Mount_Everest_as_seen_from_Drukair2_PLW_edit_Cropped.jpg*
Source: *https://en.wikipedia.org/wiki/Half_Dome#/media/File:Half_Dome10.jpg*

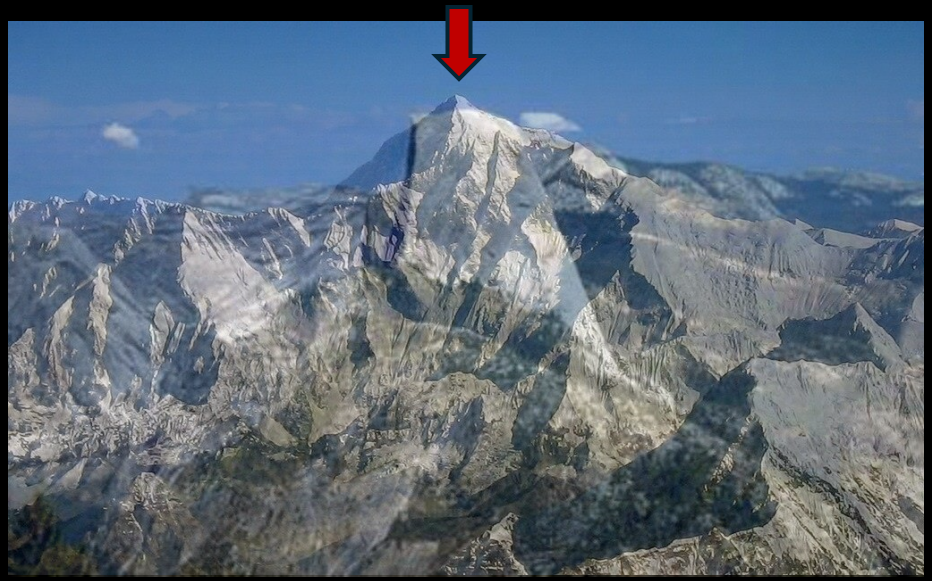
A Long Tail of Practical Challenges

- Superimpose Earth's mountains
- What location has the highest average elevation?

Source: *https://en.wikipedia.org/wiki/Mount_Everest#/media/File:Mount_Everest_as_seen_from_Drukair2_PLW_edit_Cropped.jpg*
Source: *https://en.wikipedia.org/wiki/Half_Dome#/media/File:Half_Dome10.jpg*

A Long Tail of Practical Challenges

- Superimpose Earth's mountains
- What location has the highest average elevation?
- You have no maps, and visibility is poor
- You can:
  - Look around
  - Walk for a bit
  - Teleport to the same location on a different mountain



Source: *https://en.wikipedia.org/wiki/Mount_Everest#/media/File:Mount_Everest_as_seen_from_Drukair2_PLW_edit_Cropped.jpg*
Source: *https://en.wikipedia.org/wiki/Half_Dome#/media/File:Half_Dome10.jpg*

A Long Tail of Practical Challenges

- Superimpose Earth's mountains
- What location has the highest average elevation?
- You have no maps, and visibility is poor
- You can:
  - Look around
  - Walk for a bit
  - Teleport to the same location on a different mountain

Source: Ceausu Cristian, *https://www.facebook.com/photo.php?fbid=990341093093506*

A Long Tail of Practical Challenges

- Superimpose Earth's mountains
- What location has the highest average elevation?
- You have no maps, and visibility is poor
- You can:
  - Look around
  - Walk for a bit
  - Teleport to the same location on a different mountain

Source: Ceausu Cristian, *https://www.facebook.com/photo.php?fbid=990341093093506*

A Long Tail of Practical Challenges

Any batch containing Ronda will never explore to the left

but

If you wander into the gorge, you can climb up the other side

so

You need the freedom to wander

and

This mountain-climbing exercise is an exact description of SGD

Source: Ceausu Cristian, *https://www.facebook.com/photo.php?fbid=990341093093506*

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
- CPU latency
- Document size
- Batch size limits
  - More gpus → larger batches → less stochasticity → lower data efficiency
  - Above ~4M tokens, more gpus slows convergence!
  - Further increases OK if done later in training

IBM

SIG METRICS

A Long Tail of Practical Challenges

- Distributed model checkpointing
- Stateful, distributed data loading
- Activation checkpointing
- CPU memory constraints
- CPU latency
- Document size
- Batch size limits
- Data file size
- Code versioning
- Mixed precision
- LR tuning
- LR scheduling
- Model initialization schemes
- Pretraining task(s)
- Network architecture
- …

An 80/20 Rule for Distributed Training

80% of academic literature focuses on three main aspects of training

however

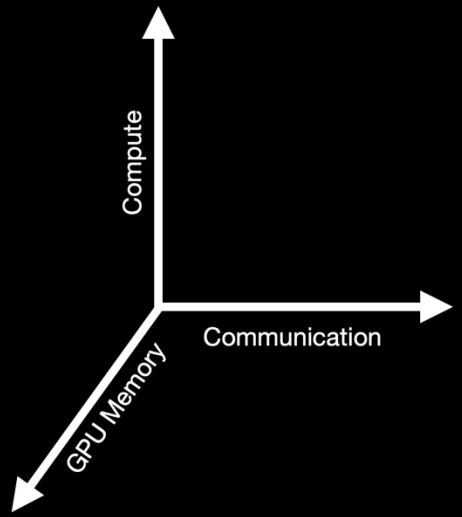
We spent only 20% of our time working on these three dimensions

and

80% of our time on a host of smaller and less-discussed challenges

so

Even with matured platforms, expect a similar distribution of effort



IBM

SIG METRICS

Contents

IBM

SIG METRICS acm

Recap

- **General concepts and intro to training at scale**
  - Basic design philosophy of PyTorch
  - Types of parallelism
    - Data
    - Model
  - Running PyTorch at scale
- **Optimizing training at scale**
  - Optimization as a process of scaling up
  - 3D design space
  - Stability over time
  - Long tail of practical implementation challenges
- **Next session:** a bag of optimization tricks, resolving the data question, some remarks on inference

Maximizing LLM Throughput in PyTorch: Optimized Pipelines for Modern Deep Learning Workloads

Davis Wertheimer
IBM Research

IBM

SIG METRICS
acm

Contents

IBM

SIG METRICS acm

Recap

- **General concepts and intro to training at scale**
  - Basic design philosophy of PyTorch
  - Types of parallelism
    - Data
    - Model
  - Running PyTorch at scale
- **Optimizing training at scale**
  - Optimization as a process of scaling up
  - 3D design space
  - Stability over time
  - Long tail of practical implementation challenges

Contents

IBM

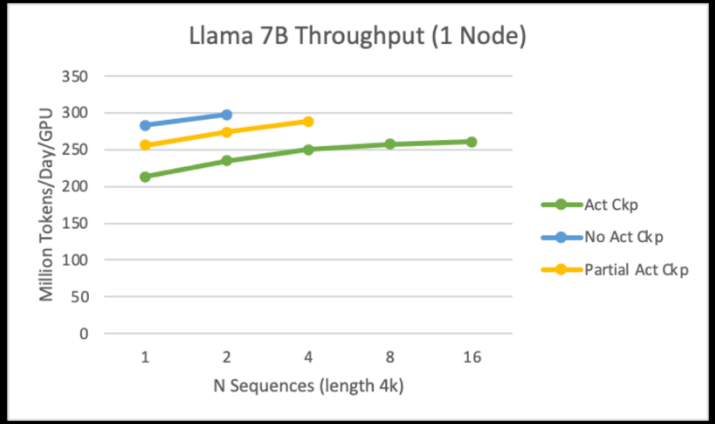
SIG METRICS acm

Activation Checkpointing

- Recall that the backward pass requires both weights and inputs

$$WX = Y, \qquad L = f(Y), \qquad \frac{dW}{dL} = X(\frac{dY}{dL})^T, \qquad \frac{dX}{dL} = W^T \frac{dY}{dL}$$

- FSDP avoids storing full weights by reconstructing on-demand
- Do the same thing with inputs!
- Selectivity in where/what to wrap
- VS gradient accumulation (33% extra compute vs ~2x comms)

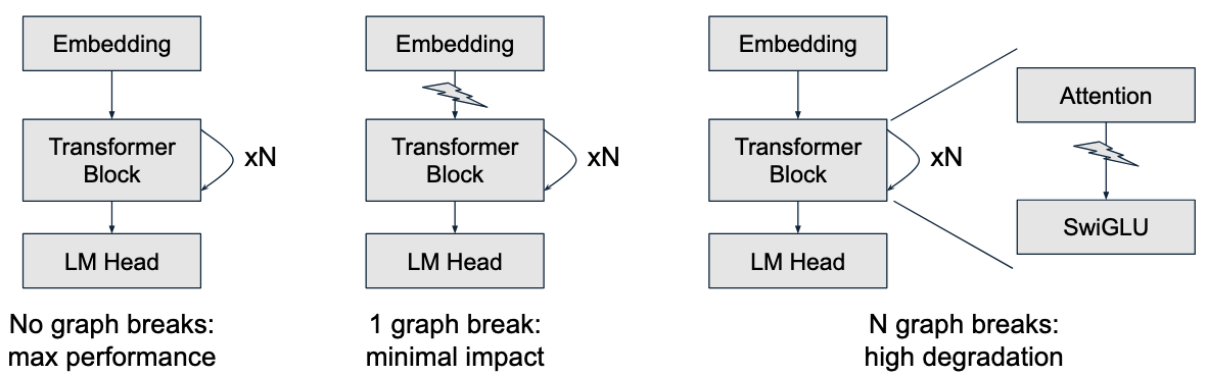


Torch Compile

- **Re-compile and optimize the execution graph**
  - Remove frequent CPU/GPU syncs
  - Fuse operations
- **Does NOT support all ops:**
  - No ops depending on CPU (print, nonzero, where, tolist)
  - No ops depending on data (e.g. if x>0 then…)
  - No functions with variable numbers/formats of output
- **Variable tensor size is supported, but re-compiles every time**
- **Does support custom kernels (Triton-native)**

Torch Compile

- **Unsupported ops cause "graph breaks"**
  - The more graph breaks, the worse the performance
- **Llama training pain points:**
  - Reimplementing RoPE (real-valued)
  - Extra RoPE caching to support input-dependent scaling
  - Training vs inference if/elses
  - FSDP boundaries



Torch Compile

DO use torch compile…

- If your workload is already well-optimized
- If you're compute-bottlenecked
- For inference

DO NOT use torch compile…

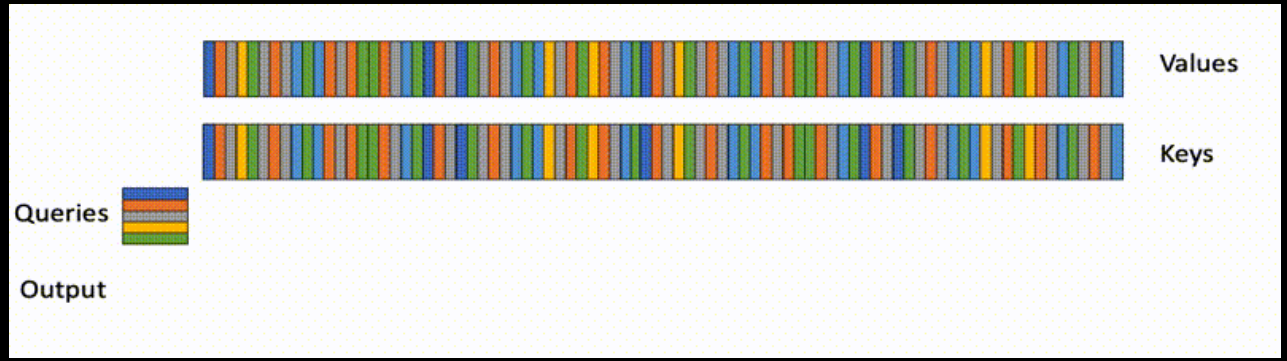
- If your model has complex code
- If your model does data-based indexing a lot (i.e. MoE)
- If you're communication-bottlenecked

IBM

SIG METRICS

Flash Attention

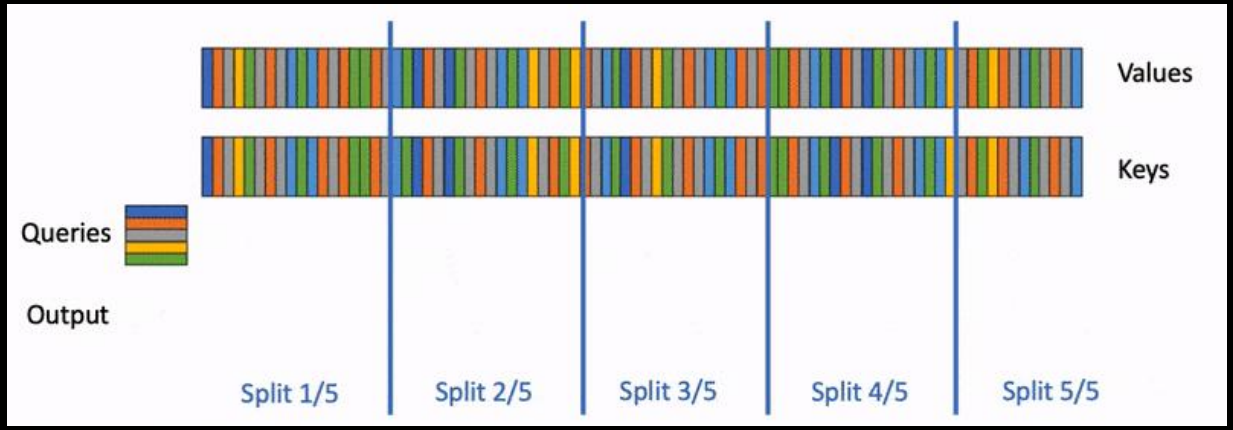
- Optimized implementation of QKV attention by Tri Dao
- Main bottleneck of attention is reading QK' into HBM
- Avoid materializing QK' by computing σ(QK')V chunkwise
  - Separate denominator tracking lets us do softmax cumulatively
- Replaces QK' with K and V (much smaller)



Source: *https://pytorch.org/blog/flash-decoding/*

Flash Attention

- A separate implementation for inference (“Flash Decoding”)
- On-device sequence parallel
  - Again, extra denominator tracking for the reduction over softmax



Source: *https://pytorch.org/blog/flash-decoding/*

Flash Attention

- Always appropriate, but gains don't always translate
- Attention layer is at least ~1/3 of model computation (best-case)

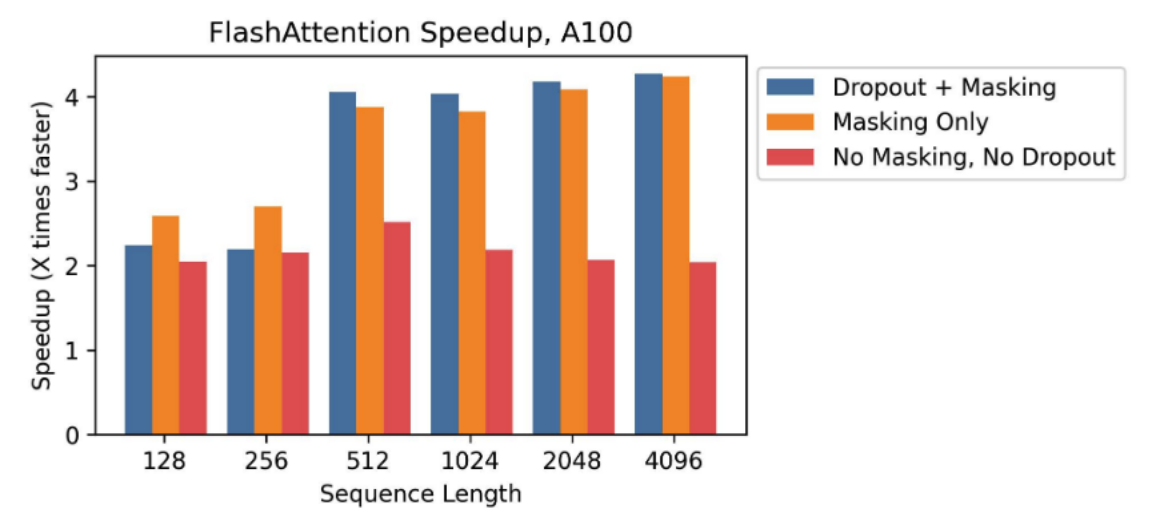
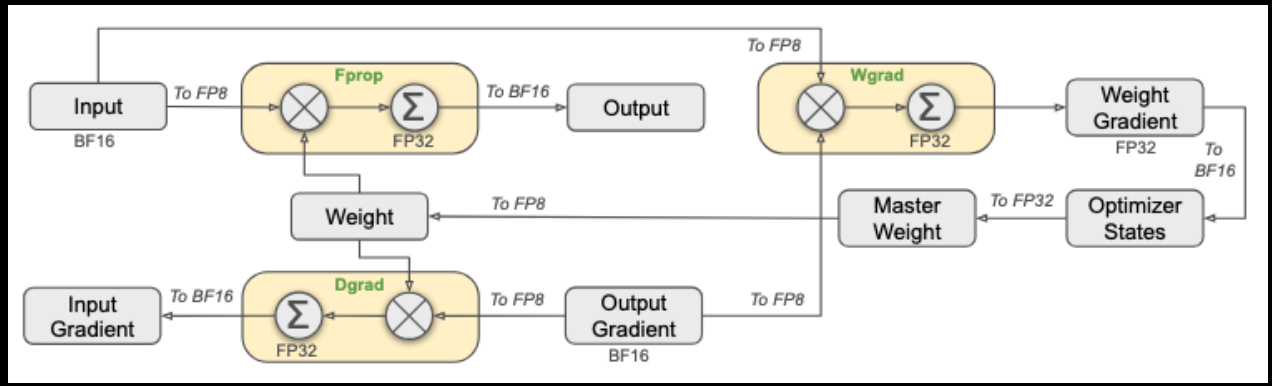


Figure 5: Speedup over standard PyTorch attention at different sequence lengths, on A100.

Source: *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness* (2022)

Mixed Precision

- **Lower precision runs faster, but isn't as stable**
- **Some rules of thumb:**
  - Master weights and gradients in fp32 (FSDP helps)
  - Activations and ephemeral weights in bf16
  - Matmuls can do fp8, but should accumulate in higher precision
  - All division (softmax, layernorm) in fp32



Source: *DeepSeek-V3 Technical Report* (2024)

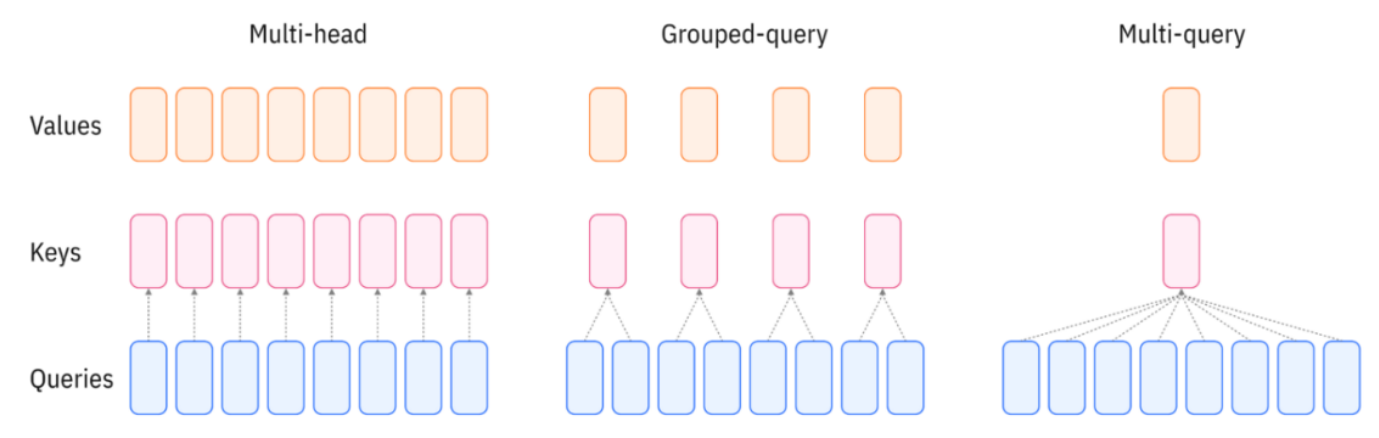
Mixed Precision

- For FSDP, policies allow for fine-grained control
- Other options:
  - torchao (https://pytorch.org/blog/pytorch-native-architecture-optimization/)
  - AMP (https://docs.pytorch.org/docs/stable/notes/amp_examples.html)

```python
import torch
from torch.distributed.fsdp import MixedPrecision
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
from model_code import model

# Master weights and grads are fp32, unless we manually cast the model to something else
bfSixteen = MixedPrecision(
    # Ephemeral weights
    param_dtype=torch.bfloat16,
    # Gradient communication
    reduce_dtype=torch.bfloat16,
    # Activations
    buffer_dtype=torch.bfloat16,
)

model = FSDP(
    model(),
    mixed_precision = bfsixteen,
    ...
)
```

Group-Query Attention

- KV cache is data-rich – why only query it once?
- Shrink number of KV heads, group Q heads
- Saves massive inference overhead
  - Memory limits
  - Overhead from reading KV cache into HBM (i.e. Flash)



IBM

SIG METRICS

Demo:
Bag of Tricks

https://tinyurl.com/davis-sigm25

- Activation Checkpointing
- Torch Compile
- Flash Attention
- Mixed Precision
- GQA

Contents

IBM

SIG METRICS

Torch Profiler

- A (too) powerful diagnosis tool
- A context wrapper that tracks computation
- Exports huge JSON
  - For Chrome users: chrome://tracing
  - Otherwise: ui.perfetto.dev
- Visually examine runtimes

```
# On rank 0 only:
profiler = torch.profiler.profile(
    activities=[
        torch.profiler.ProfilerActivity.CPU,
        torch.profiler.ProfilerActivity.CUDA,
    ],
    schedule=torch.profiler.schedule(wait=1, warmup=2, active=3, repeat=1),
    on_trace_ready=torch.profiler.tensorboard_trace_handler("profile_traces"),
    profile_memory=True,
    with_stack=False,
    record_shapes=True,
)

for step in range(steps):

    # [YOUR COMPUTATION HERE]

    profiler.step()

# Saves to [working directory]/profile_traces/[gibberish].pt.trace.json
```

IBM

SIG METRICS acm

Demo: Profiling the Bag of Tricks

https://tinyurl.com/davis-sigm25

ui.perfetto.dev

- Flash Attention
- Mixed Precision
- Torch Compile
- Activation Checkpointing
- GQA

PART 5: Data at Scale

Contents

IBM

SIG METRICS acm

Data at Scale

1. Stateful and checkpointable, mid-epoch resumption
2. Auto-rescale checkpoints to changing workload/GPU allocations
3. Data streaming with efficient shuffling: queue TB of data quickly
4. Efficient and asynchronous, no peer-to-peer
5. Dynamic data mixing and tokenization (minimize preprocessing)
6. PyTorch-native, modular, extensible

(MosaicML has the first 4, but not 5 or 6)

Data at Scale

So we built it ourselves!

Proven:

- Hundreds of training jobs
- Observed months of continuous operation
- Over 90,000 tokens/gpu/sec while being “async”

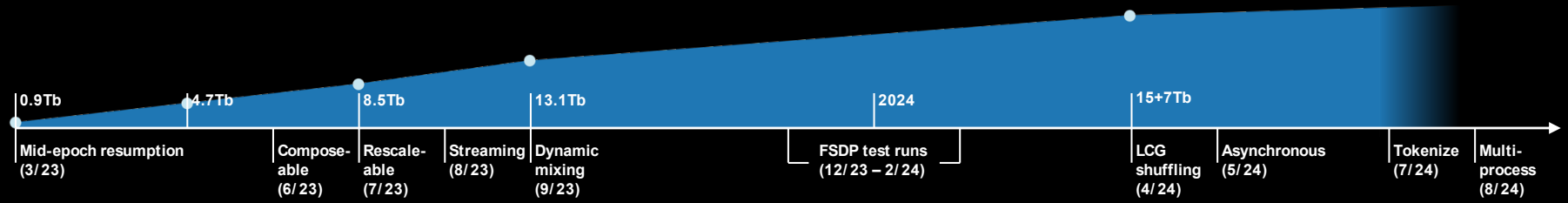
So what?

- Training pipelines move at the speed of the slowest bottleneck
- FSDP+compile+fp8+flash+etc. **demands** a good data loader!

IBM

SIG METRICS acm

Our Dataloader Journey



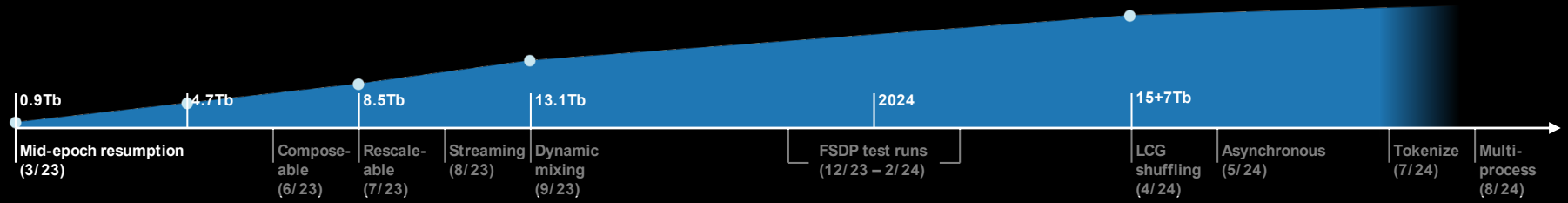
Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Mid-Epoch Resumption

Mar 2023



Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Mid-Epoch Resumption

Mar 2023

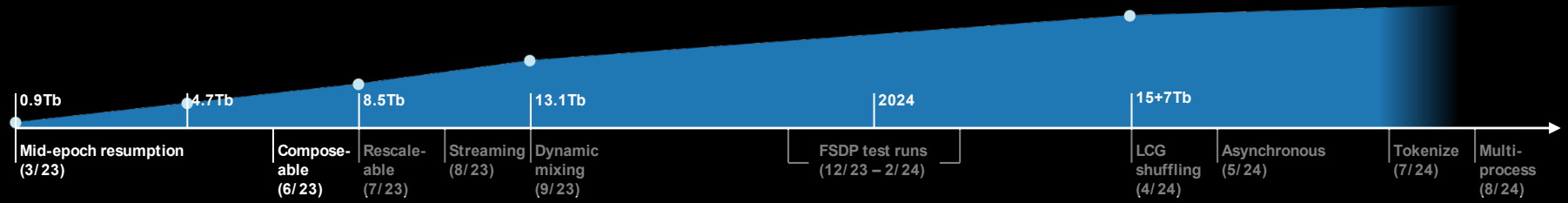
- At the time, no distributed dataloader was available with mid-epoch resumption
- PyTorch *does* support *non-stateful* distributed loading via DistributedSampler
- Baby steps: add n_seen argument to existing DistributedSampler

```python
class Stateful_DistributedSampler(data.distributed.DistributedSampler):
    """
    Extends the PyTorch distributed data sampler with the ability to restart from a given step via the extra n_seen argument.
    """

    def __init__(self, dataset, n_seen, num_replicas=None, rank=None, shuffle=True, seed=42):
        self.n_seen = n_seen
        super().__init__(dataset, num_replicas, rank, shuffle)

    def __iter__(self):
        # LIFTED FROM ORIGINAL PYTORCH CODE:
        if self.shuffle:
            # deterministically shuffle based on epoch and seed
            g = torch.Generator()
            g.manual_seed(self.seed)
            indices = torch.randperm(len(self.dataset), generator=g).tolist()  # type: ignore[arg-type]
        else:
            indices = list(range(len(self.dataset)))  # type: ignore[arg-type]

        if not self.drop_last:
            # add extra samples to make it evenly divisible
            padding_size = self.total_size - len(indices)
            if padding_size <= len(indices):
                indices += indices[:padding_size]
            else:
                indices += (indices * math.ceil(padding_size / len(indices)))[:padding_size]
        else:
            # remove tail of data to make it evenly divisible.
            indices = indices[: self.total_size]
        assert len(indices) == self.total_size

        # subsample
        indices = indices[self.rank : self.total_size : self.num_replicas]
        assert len(indices) == self.num_samples

        # NEW CODE:
        offset = self.n_seen % self.num_samples
        indices = indices[offset:] + indices[:offset]

        return iter(indices)
```

IBM

SIG METRICS acm

Composable Iterable Datasets June 2023



Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

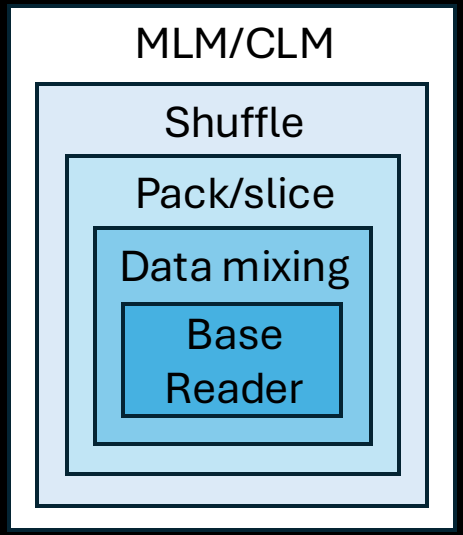
IBM

SIG METRICS

Composable Iterable Datasets

June 2023

- Multi-stage data transformation requires multiple loaders
- Indexing doesn't work when items in ≠ items out (e.g. packing/slicing documents)
- Extend PyTorch's IterableDataset to create wrappers with recursive iter()
- Then extend with recursive state_dict() / load_state_dict()

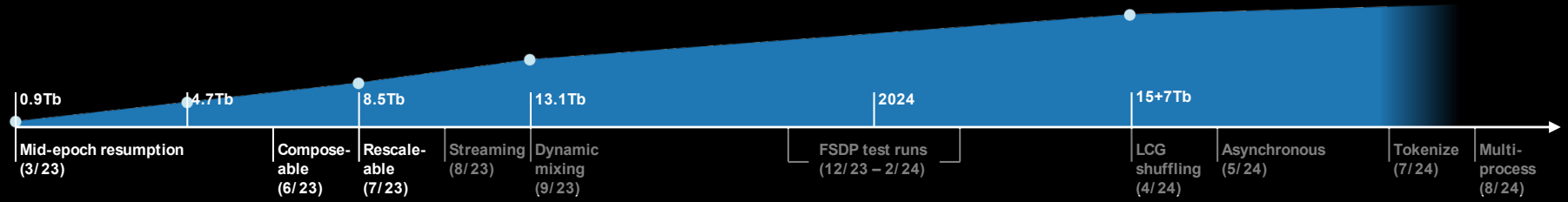


IBM

SIG METRICS

Rescalability Over GPUs

July 2023



Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

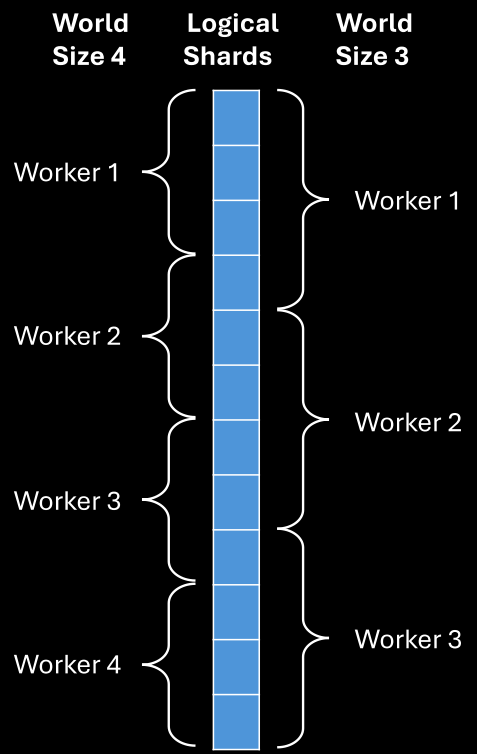
IBM

SIG METRICS

Rescalability Over GPUs

July 2023

- Auto-reshard handling for wrappers:
  - State flags: RNG states, token counts, scalars (discard)
  - Reshard flags: lists, buffers (re-partition)
  - Easily extensible – just specify states/reshards
- Rescalability layer: a list of subloaders / logical shards
  - Everything before this layer stays non-elastic
  - Everything above this layer auto-reshards
- No repartitioning, still no revisiting seen data

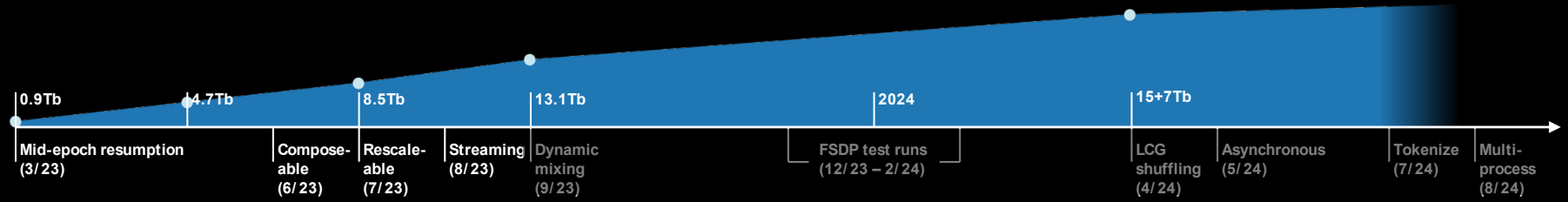


IBM

SIG METRICS

Data Streaming

Aug 2023



Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

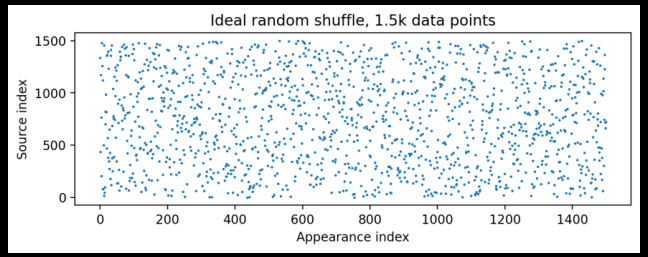
IBM

SIG METRICS

Data Streaming

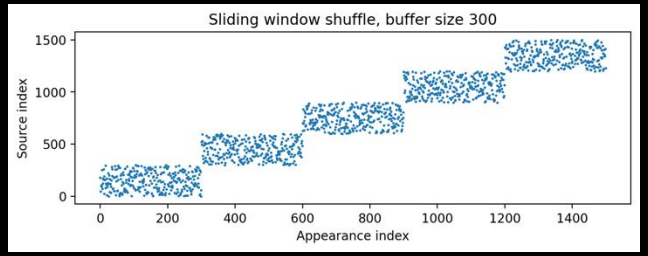
Aug 2023

- Data streaming requires dedicated shuffling
- Minimize cache size when possible
- Within-file: full shuffle
- Across-file: buffer shuffle
  - Given buffer:
  - Choose random index
  - Overwrite with new value
  - Yield old value

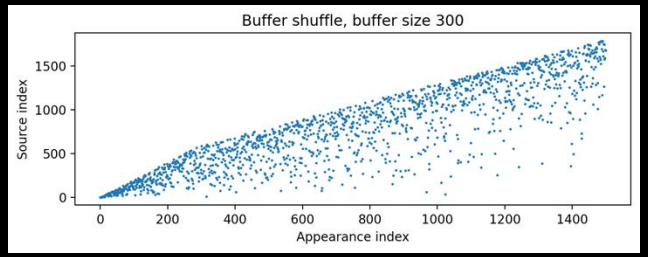


Avg distance between consecutive items:

500



~100



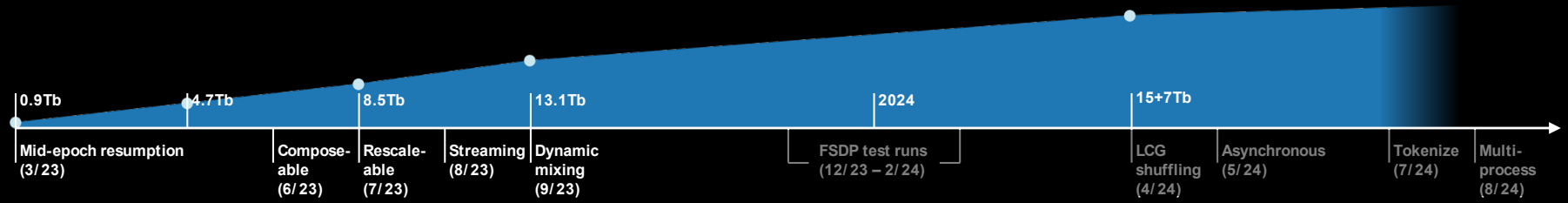
~300

IBM

SIG METRICS

Data Mixing

Sep 2023



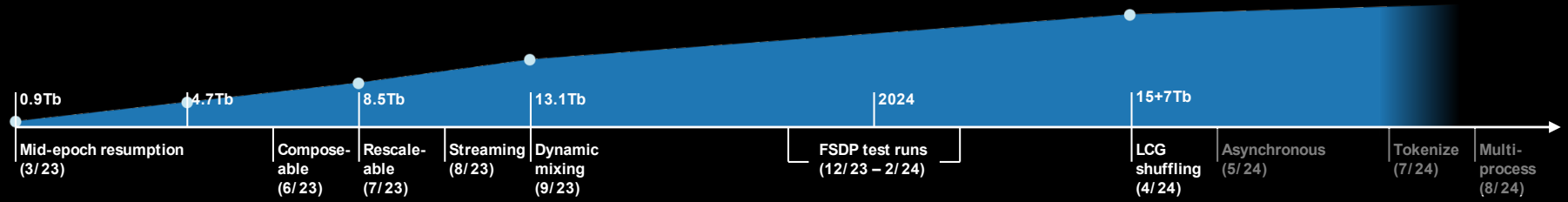
Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Stateless Shuffling

April 2024



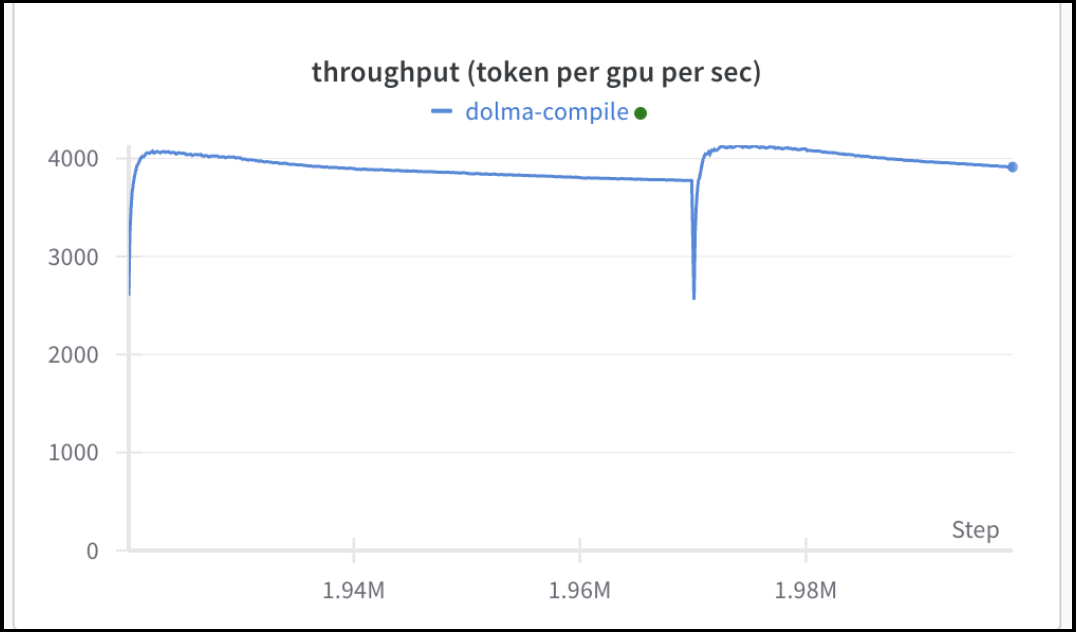
Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Stateless Shuffling

April 2024



CPU bottlenecks matter!

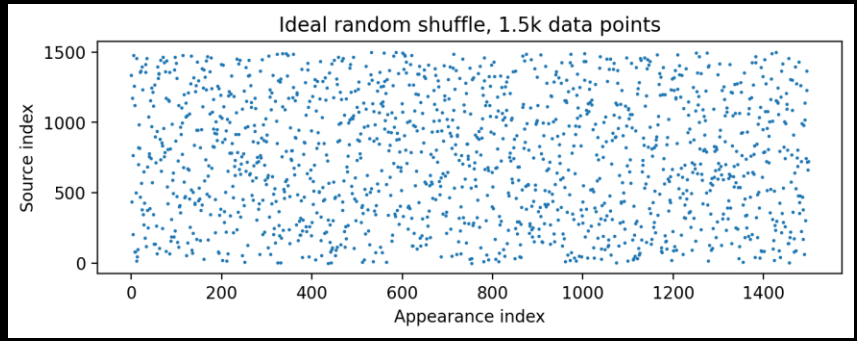
IBM

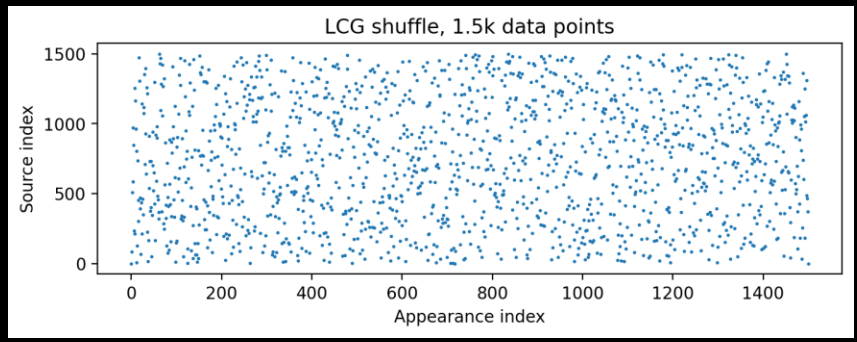
SIG METRICS

Stateless Shuffling

April 2024

- LCG (Linear Congruential Generator) (adapted from Knuth, 1997)
- Random walk over ANY size
- State is three scalars! (seed, size, last index)
- >100Gb overhead to ZERO



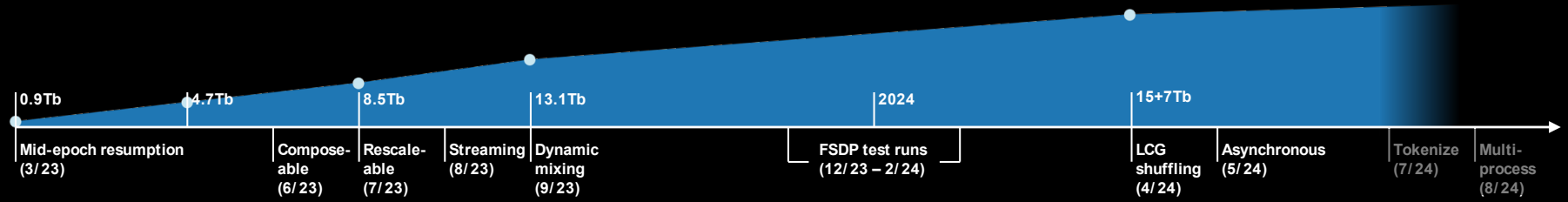


IBM

SIG METRICS

Asynchronous Loading

May 2024



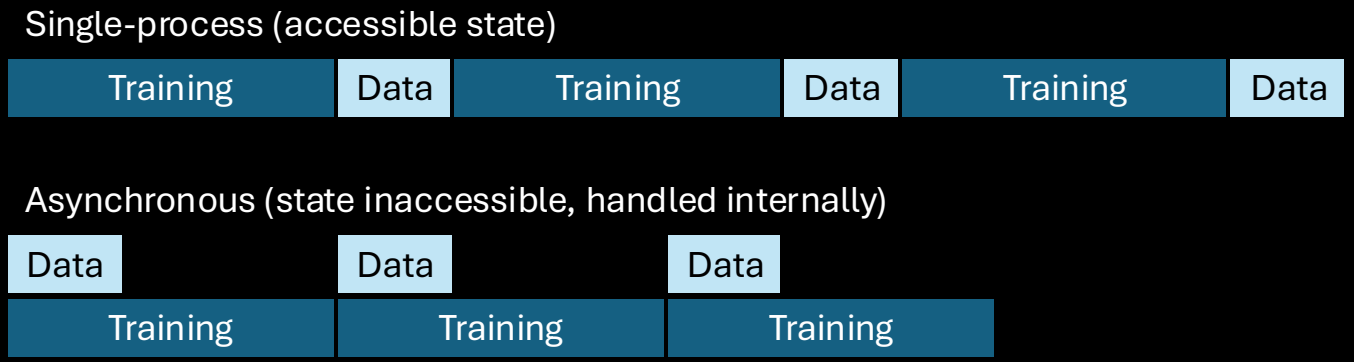
Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Asynchronous Loading

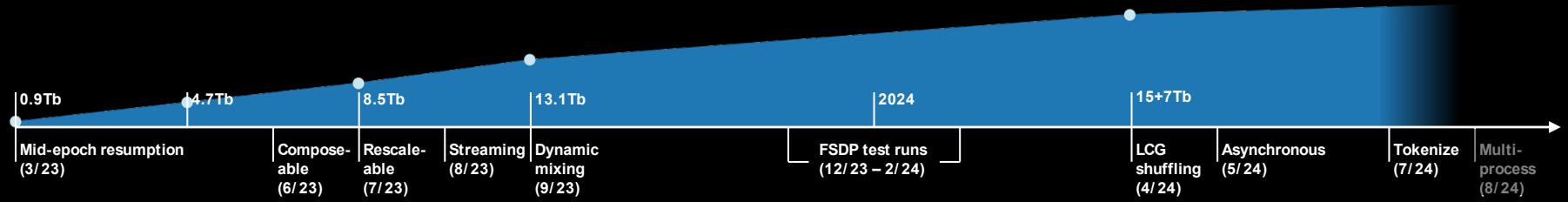
May 2024



- Use existing num_workers arg in torch DataLoader
- Auto-handle checkpoints via another iterable wrapper CheckpointDataset

On-the-Fly Tokenization

July 2024



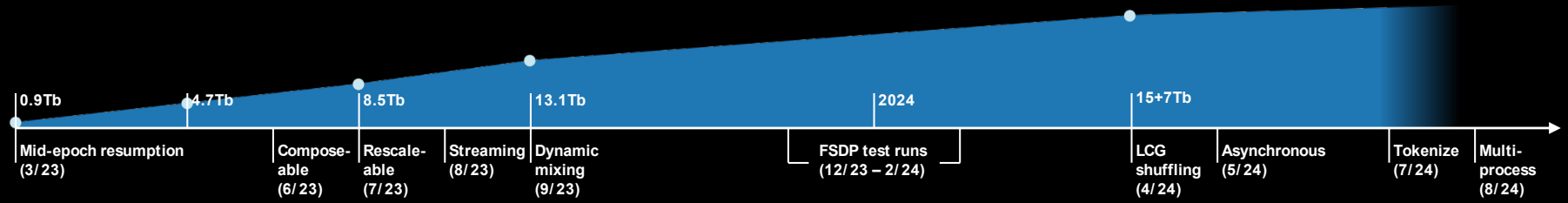
Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

Multi-Process Support

Aug 2024



Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

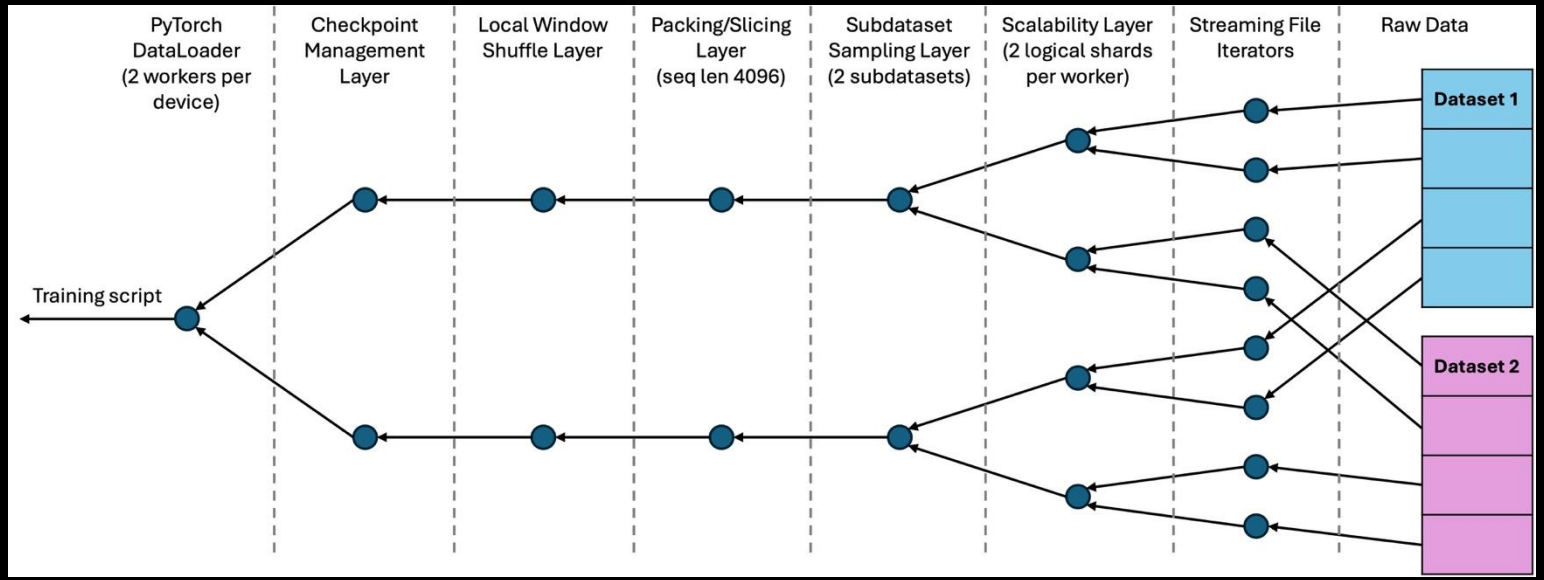
IBM

SIG METRICS

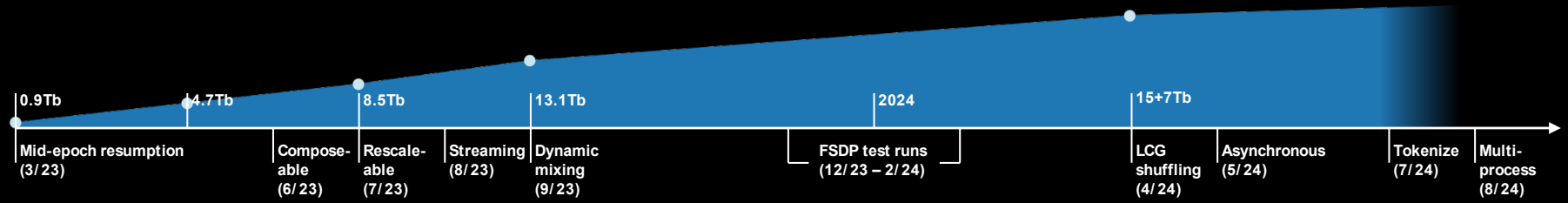
Data Mixing

Aug 2024

- Iterators and sub-iterators: a tree structured loader
- Multiple subdatasets with multiple shards per dataset



More Recently

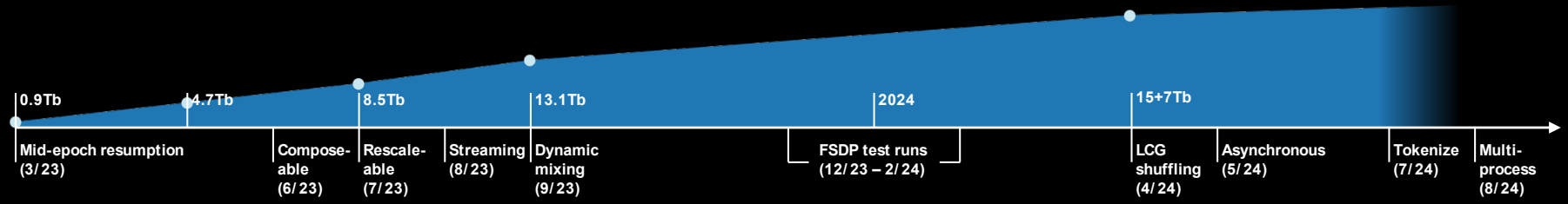


Mid-epoch resumption	Resume training runs with uninterrupted, unchanged behavior
Composability	Support multi-stage processing, with state at each level
Rescalability	Scale to different numbers of GPUs without changing data order
Data streaming	Pull files and documents on-demand with local shuffling
Dynamic data mixing	Just-in-time data sampling, token percentages track targets constantly
Stateless shuffling	Custom LCG random walk enables local shuffling w/o memory overhead
Asynchronicity	Dataloading runs in separate process, auto-checkpoints, does not block
On-the-fly tokenization	Support for multiple file types and degrees of preprocessing
Multi-processing	Defer setup to allow several parallel workers per device

IBM

SIG METRICS

More Recently



- Open PRs contributing to TorchTitan and TorchData
- Adding new capabilities and ironing out edge cases
- Rely on this as we upgrade to FSDP2, FP8 training, new hardware, etc.

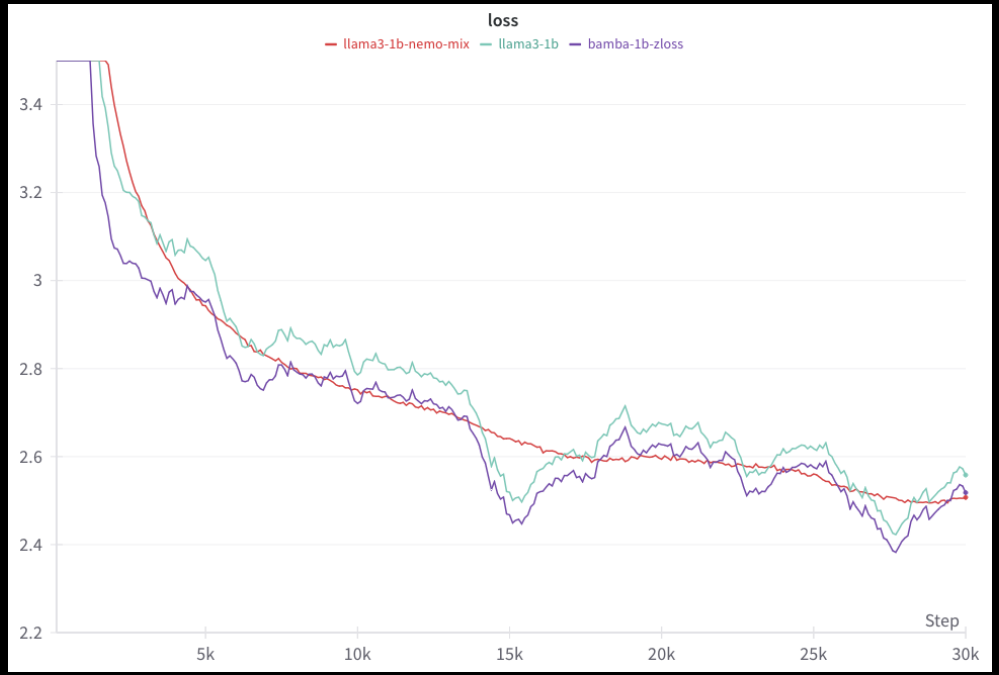
IBM

SIG METRICS

Takeaway

Distributed data loading is just as complex and multidimensional as distributed training!

A recent horror story: an interaction between long documents and small files



Demo: Setting Up a Distributed Dataloader

https://tinyurl.com/davis-sigm25

PART 6: Accelerating Inference

Contents

1. INTRO
2. BASICS OF TRAINING AT SCALE
3. OPTIMIZING TRAINING AT SCALE
4. BAG OF TRICKS
5. DATA AT SCALE
6. ACCELERATING INFERENCE
   - The nature of inference workloads
   - Bag of tricks
   - Speculative decoding

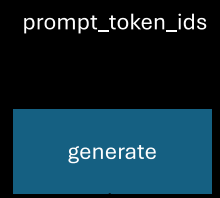
IBM

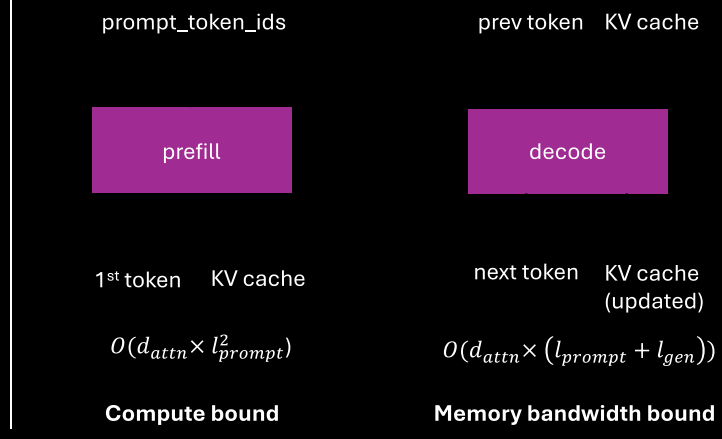
SIG METRICS

The Nature of Inference Workloads

LLM pretraining is homogeneous, inference is not!

- Multi-stage:
  1. Encode the prompt (prefill)
  2. Generate token by token (decode)
- Bottlenecks:
  1. Memory (K/V cache)
  2. Device-inefficiency
  3. (scale)





IBM

SIG METRICS acm

The Nature of Inference Workloads

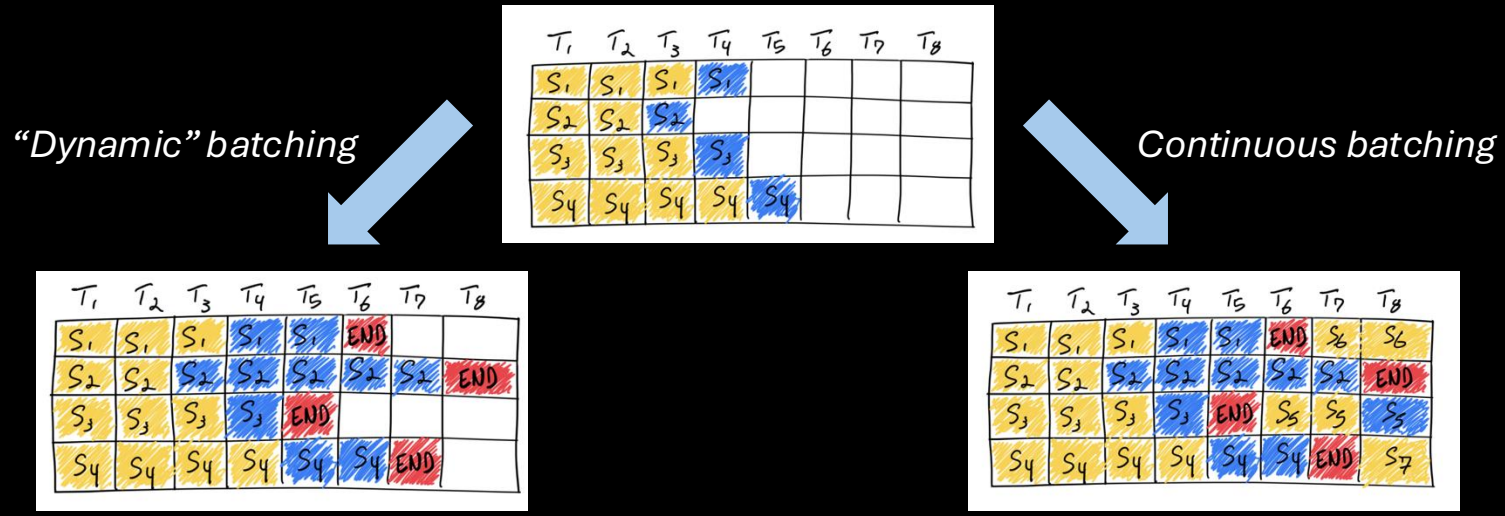
A familiar 80/20 rule:

- 80% of discussion / papers focus on model optimization
- 80% of work is in dealing with scale and heterogeneity
- New solutions are less algorithms, and more names for general concepts
    1. Dynamic vs continuous batching
    2. Chunked vs disaggregated prefill
    3. Prefix caching / paged attention

Dynamic vs Continuous Batching

Fix device-inefficiency by batching

- "Inference server" to schedule and coordinate model calls
- But how to batch heterogeneous workloads?

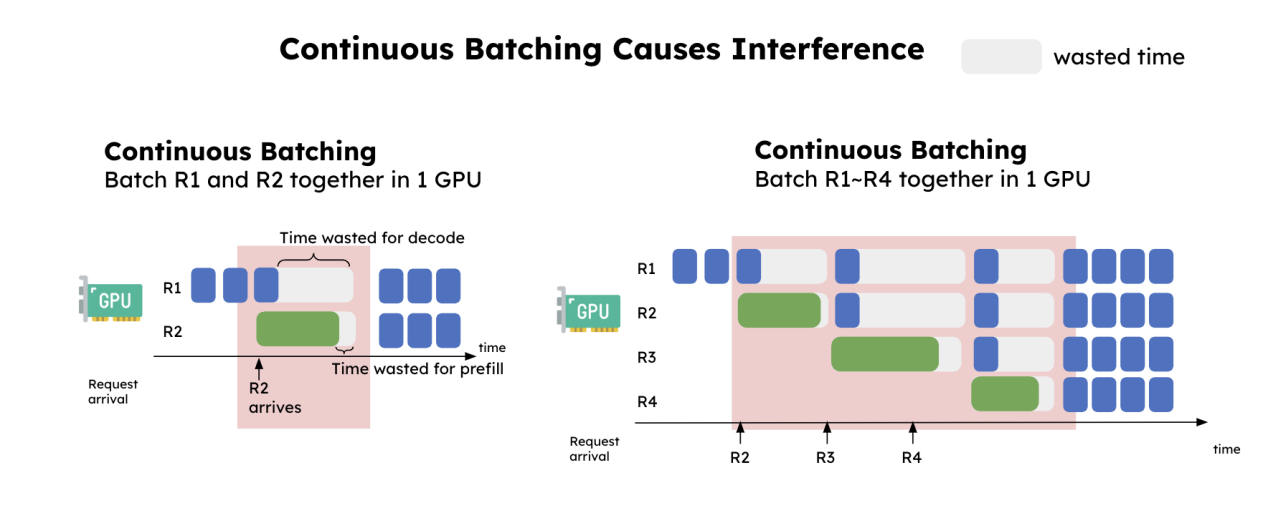


Source: *https://www.anyscale.com/blog/continuous-batching-llm-inference*

Chunked vs Disaggregated Prefill

Continuous batching addresses request heterogeneity, but not stage heterogeneity

- Compute prefill chunk-by-chunk
- Compute prefill in an entirely separate server

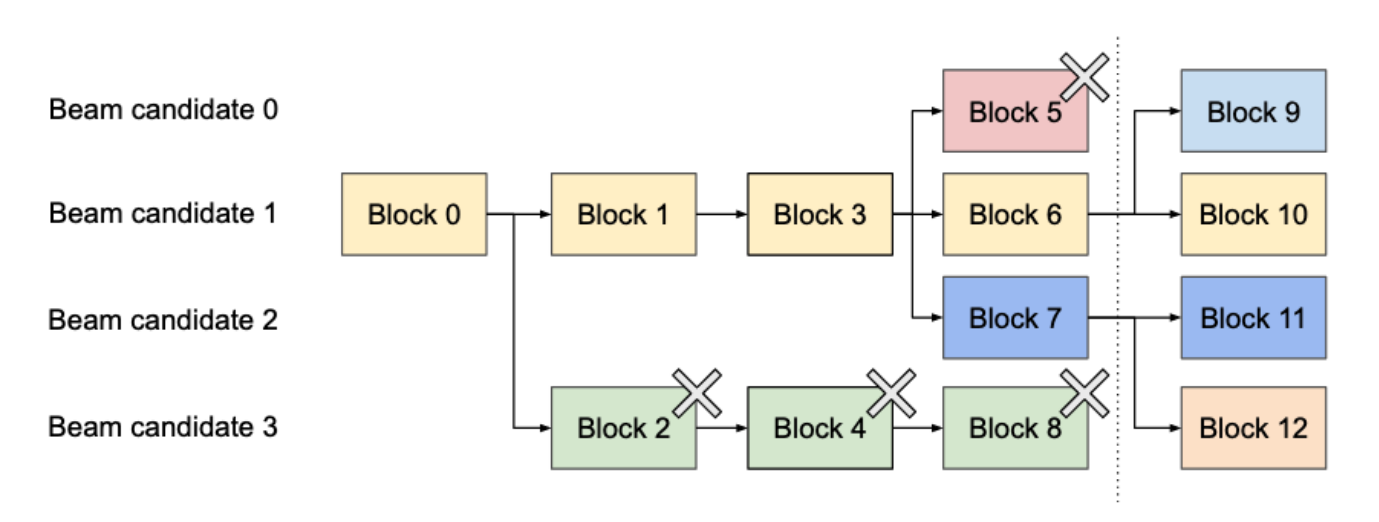


Source: *https://hao-ai-lab.github.io/blogs/distserve/*

Prefix Caching / Paged Attention

Re-use K/V cache entries as much as possible

- Branching inputs (beam search, templates)
- One level of indirection for KV cache addresses

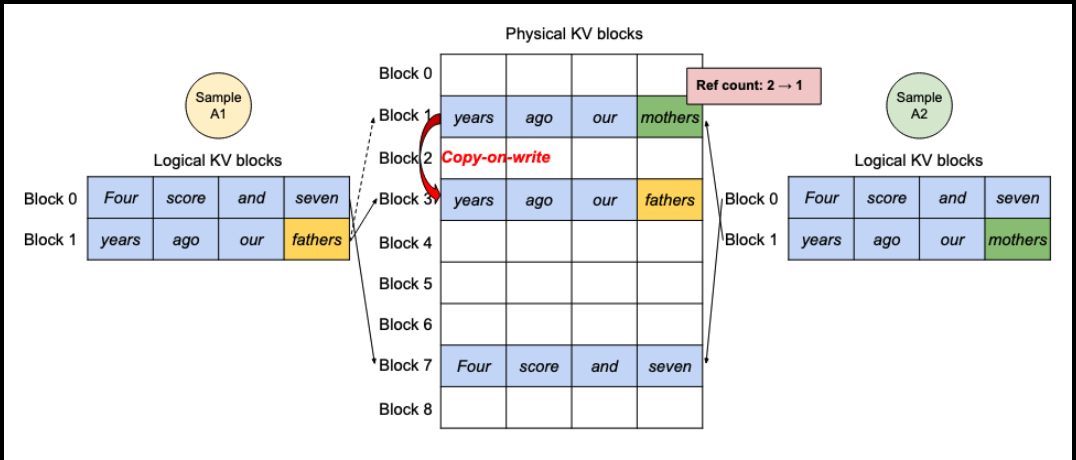


Source: *Efficient Memory Management for Large Language Model Serving with PagedAttention* (2023)

Prefix Caching / Paged Attention

Re-use K/V cache entries as much as possible

- Branching inputs (beam search, templates)
- One level of indirection for KV cache addresses
- Addresses memory and device-inefficiency



Source: *Efficient Memory Management for Large Language Model Serving with PagedAttention* (2023)

A Similar Bag of Tricks

- Parallelism (TP / PP / DP / SP / EP)
- Mixed precision (quantization)
- Torch compile
- GQA
- **Speculative decoding**

Contents

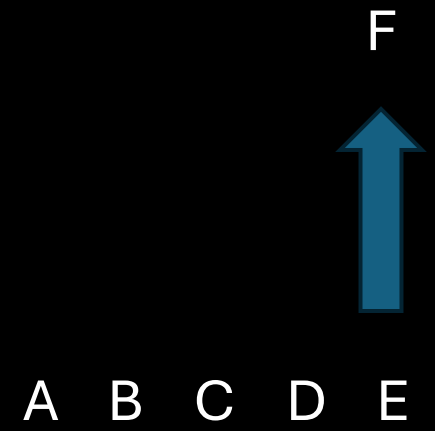
1. INTRO
2. BASICS OF TRAINING AT SCALE
3. OPTIMIZING TRAINING AT SCALE
4. BAG OF TRICKS
5. DATA AT SCALE
6. ACCELERATING INFERENCE
   - The nature of inference workloads
   - Bag of tricks
   - Speculative decoding

IBM

SIG METRICS acm

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized



Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead

A B C D E F G X

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead
  - Forward pass reveals how many match the true model

IBM

SIG METRICS

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead
  - Forward pass reveals how many match the true model
  - Unbroken chain of correct guesses → free tokens

IBM

SIG METRICS

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead
  - Forward pass reveals how many match the true model
  - Unbroken chain of correct guesses → free tokens
  - Dump any incorrect guesses, repeat

IBM

SIG METRICS

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead
  - Forward pass reveals how many match the true model
  - Unbroken chain of correct guesses → free tokens
  - Dump any incorrect guesses, repeat

A B C D E F G H

Speculative Decoding

- Baseline: predict 1 token at a time
  - Memory bottleneck, compute underutilized
- Speculation: use a small model to guess *n* tokens ahead
  - Forward pass reveals how many match the true model
  - Unbroken chain of correct guesses → free tokens
  - Dump any incorrect guesses, repeat
- In practice: multiple candidate sequences per prompt

A B C D E F G X

Speculative Inference Loop

Perform forward pass on prompt, fill kv cache

While n_tokens < requested:

- Fetch latest token and embedding
- Speculator generates k candidate suffix sequences
- Base model scores candidates
- Correct tokens from best candidate are added to result
- K/V for ONLY those tokens and candidate are added to kv-cache

Try it yourself!

https://tinyurl.com/davis-sigm25

IBM

SIG METRICS

Speculator / Draft Model

Some concerns and research questions:

- Speedup depends on speculator accuracy
- Must balance speculator overhead vs power
- Must also balance batch size (candidates) vs recall

So we tried it!

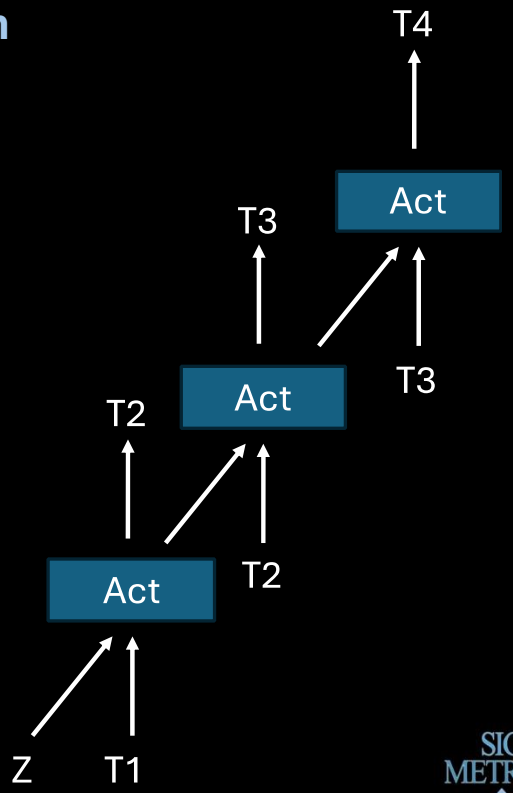
IBM

SIG METRICS

Speculator / Draft Model

Either a smaller LLM, or a bespoke model extension

- Small as possible
- Recurrent architecture, tree of outputs (candidates)
- Bespoke extensions:
  - Latest embedding input
    - Without it, just an n-gram model
    - No understanding of context
  - Sampled token input
    - Without it, conditioning on expectation over multiple tokens
    - Lots of junk candidates



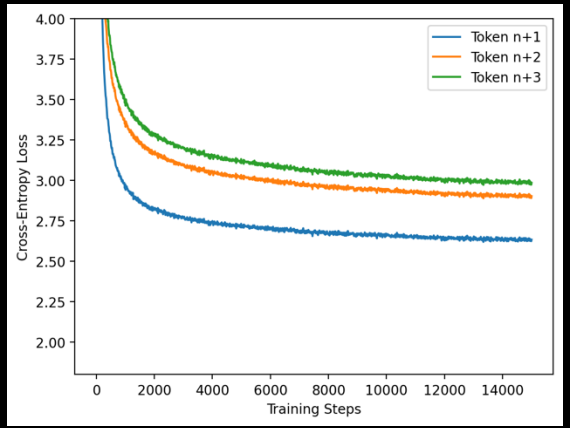
IBM

SIG METRICS

Two-Stage Training Pipeline

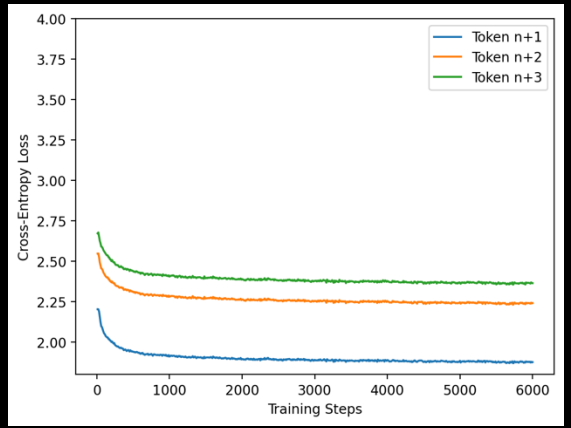
Stage 1: align to text

- Standard CLM task
- **32** gpus, 4k seq len, 1M bsize
- 15k steps / 16B tokens
- **1.1/1.5** days for 7/13B
- **1.95 – 2.49x** logical speedup



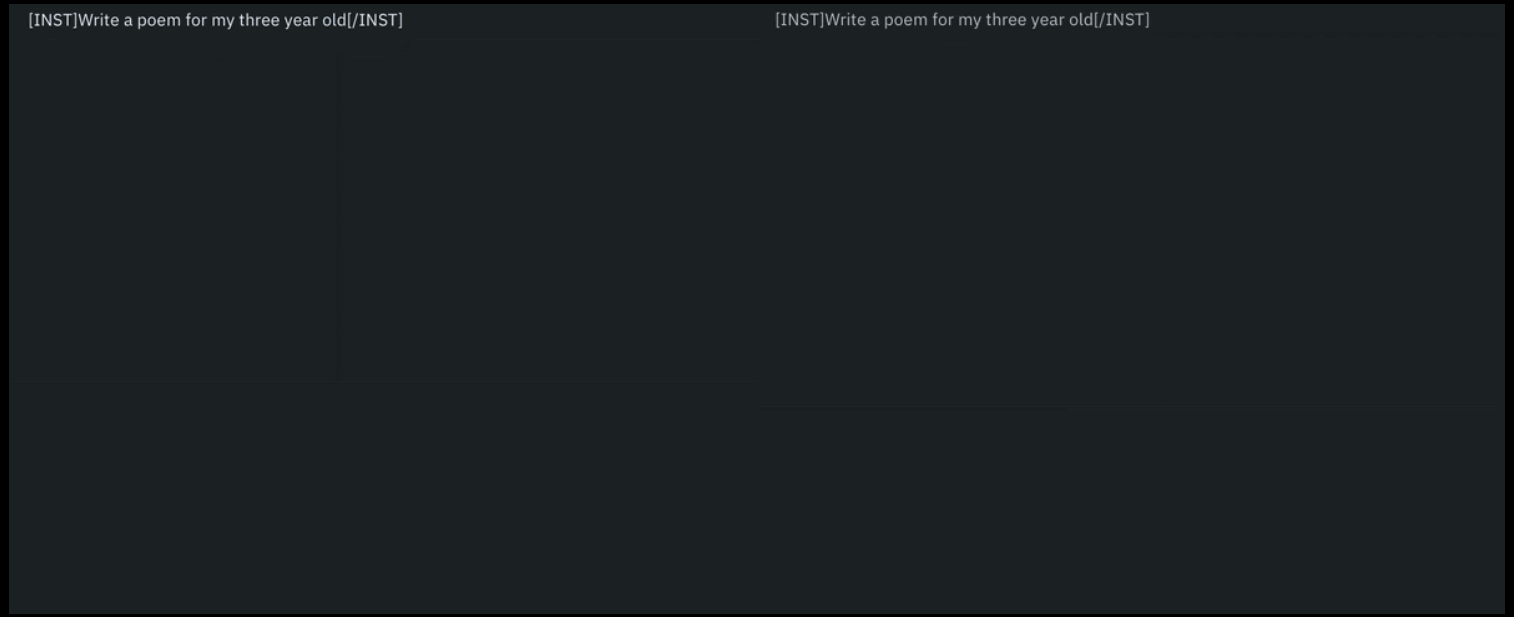
Stage 2: align to base model

- Generate targets from base model
- **32** gpus, 64+256 seq len, 1M bsize
- 6k steps / 6B tokens
- **0.9/1.5** days for 7/13B
- **2.18 – 2.72x** logical speedup



Results

- 2-3x improvement in wall-clock latency
- On top of already-optimized inference (2x other papers)



Source: *https://pytorch.org/blog/hitchhikers-guide-speculative-decoding/*

Results

- 2-3x improvement in wall-clock latency
- On top of already-optimized inference (2x other papers)
- 44k downloads on our Llama3-8B accelerator

Repo Type	Repo Name	Total Downloads
Model	ibm-ai-platform/llama3-8b-accelerator	44.4k
Model	ibm-ai-platform/llama3-70b-accelerator	10.2k

IBM

SIG METRICS

Results

ms/tok (64)	0	1	2	4	8	16	32
1	10.54	6.23	5.76	**5.50**	5.55	6.00	7.21
2	10.66	6.32	6.01	5.91	6.32	7.62	11.15
4	10.78	6.78	6.67	6.95	8.73	11.90	18.31
τ	1.00	2.21	2.44	2.67	2.84	2.96	3.03

ms/tok (2048)	0	1	2	4	8	16	32
1	12.87	9.11	**8.89**	8.91	9.68	11.51	14.78
2	14.98	10.78	10.94	11.56	13.24	16.83	24.96
4	19.62	14.24	15.63	16.78	20.97	29.05	44.88
τ	1.00	2.01	2.21	2.42	2.61	2.74	2.83

Table 1. Iterative latency (milliseconds per token) for Llama2-7B. Rows indicate batch size b, columns indicate number of parallel candidates k. $k = 0$ indicates non-speculative baseline, and logical speedup (tokens per step) is given as τ. Prompt length p is 64 (left) vs 2048 (right). As baseline computational load increases, speculative decoding provides less improvement.

ms/tok (64)	0	1	2	4	8	16
1	18.88	6.3	5.99	5.67	5.76	6.96
2	19.34	9.01	**5.51**	5.64	7.22	8.71
4	19.60	5.55	5.94	7.86	10.08	13.99
τ	1.00	5.02	5.31	5.56	5.76	5.86

ms/tok (512)	0	1	2	4	8	16
1	20.12	6.46	6.44	10.55	6.60	7.98
2	21.14	6.25	**6.19**	6.90	9.03	10.93
4	22.82	6.71	7.55	10.16	13.43	17.91
τ	1.00	5.49	5.73	5.92	6.10	6.21

Table 2. Iterative latency (milliseconds per token) for Codellama-13B-instruct. Rows indicate batch size, columns indicate number of parallel candidates. Candidates 0 indicates non-speculative baseline, and logical speedup (tokens per step) is given as τ. Prompt length is 64 (left) vs 512 (right). As baseline computational load increases, speculative decoding provides less improvement.

IBM

SIG METRICS acm

Speculative Decoding: Takeaways

- Leverage available parallelism bandwidth by turning generation into parallel verification
- Effectiveness decreases as computational load / efficiency increases
- Bespoke accelerators work better in lab conditions
- Separate LLM speculators are more robust

Conclusion

Conclusion

1. Optimization as a practical problem vs an academic problem
   - Multidimensional and open-ended
   - Long tails of practical challenges
     - Training: stability, data handling
     - Inference: batching, scheduling, heterogeneity
   - No universal good solutions!
2. Plenty of useful tools
   - Parallelism, mixed precision, compile, GQA
   - Training: activation checkpointing, torch profiler
   - Inference: prefix caching, speculative decoding
3. The right perspective to reconcile and leverage points 1 and 2

Thank you!

https://tinyurl.com/davis-sigm25

IBM

SIG METRICS acm