DISTRIBUTED DEEP LEARNING

NATHAN NICHOLS & KAUSHIK VELUSAMY

AI/ML Team
Argonne National Laboratory

ATPESC 2025

Argonne NATIONAL LABORATORY

OUTLINE

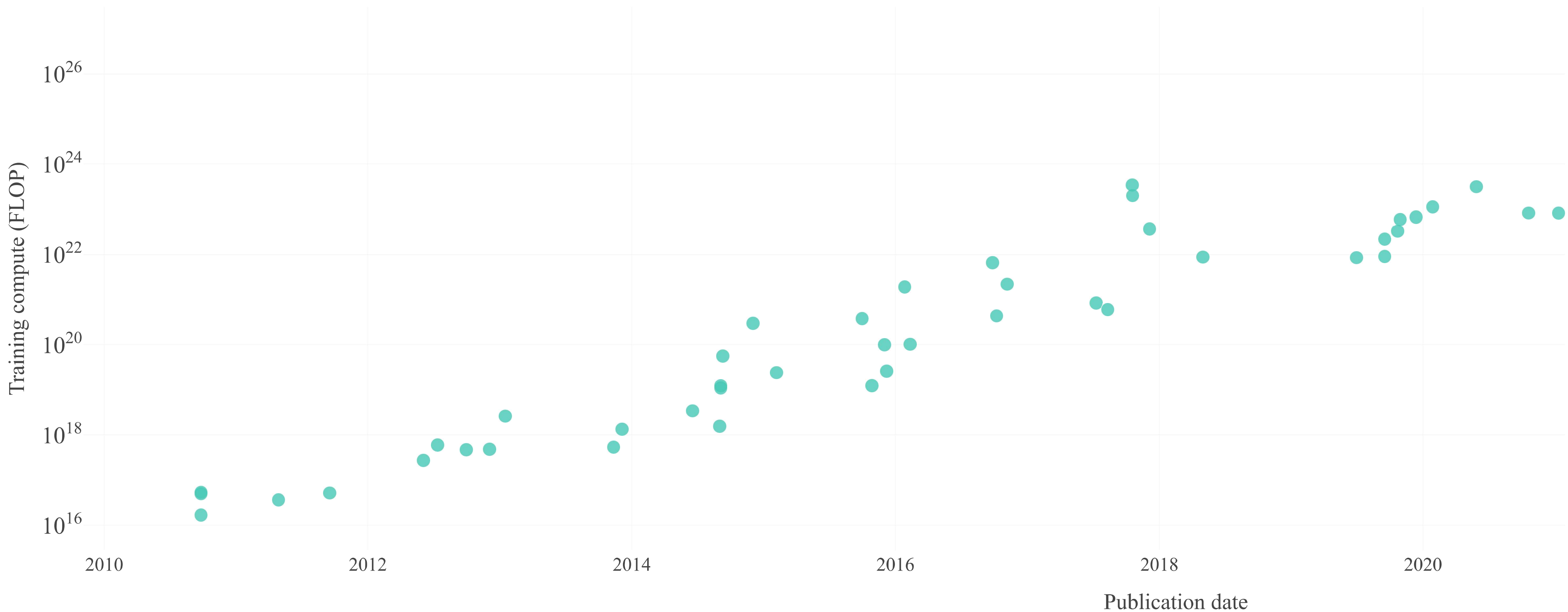
- The need for distributed training
- Communication libraries
- State-of-the-art parallelization schemes
- Data-parallel training in detail
- I/O and data management in distributed training
- Hands-on

THE NEED FOR DISTRIBUTED TRAINING ON HPC

> *"Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.4-month doubling time (by comparison, Moore's Law had a 2-year doubling period)."*

*Dario Amodei & Danny Hernandez, **AI and compute**, OpenAI Blog, May 16 2018*

TRAINING COMPUTE OF FRONTIER MODELS



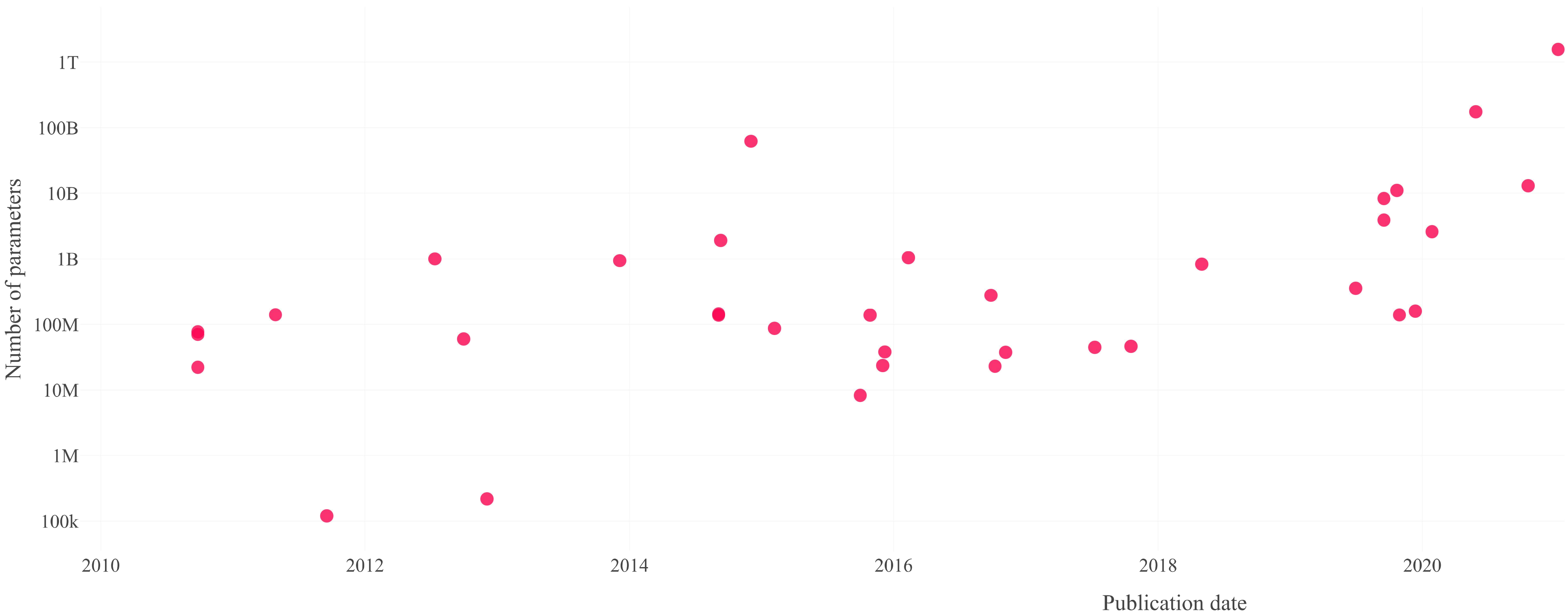
*Epoch AI: **Key Trends and Figures in Machine Learning***

TRANSFORMER ARCHITECTURE INTRODUCED!

ATTENTION IS ALL YOU NEED

- Introduced by Vaswani et al. at NeurIPS 2017
- Replaced recurrence/convolutions with self-attention
- Enabled massive parallelization & modeling of long contexts

MODEL SIZE OF FRONTIER MODELS



*Epoch AI: **Key Trends and Figures in Machine Learning***

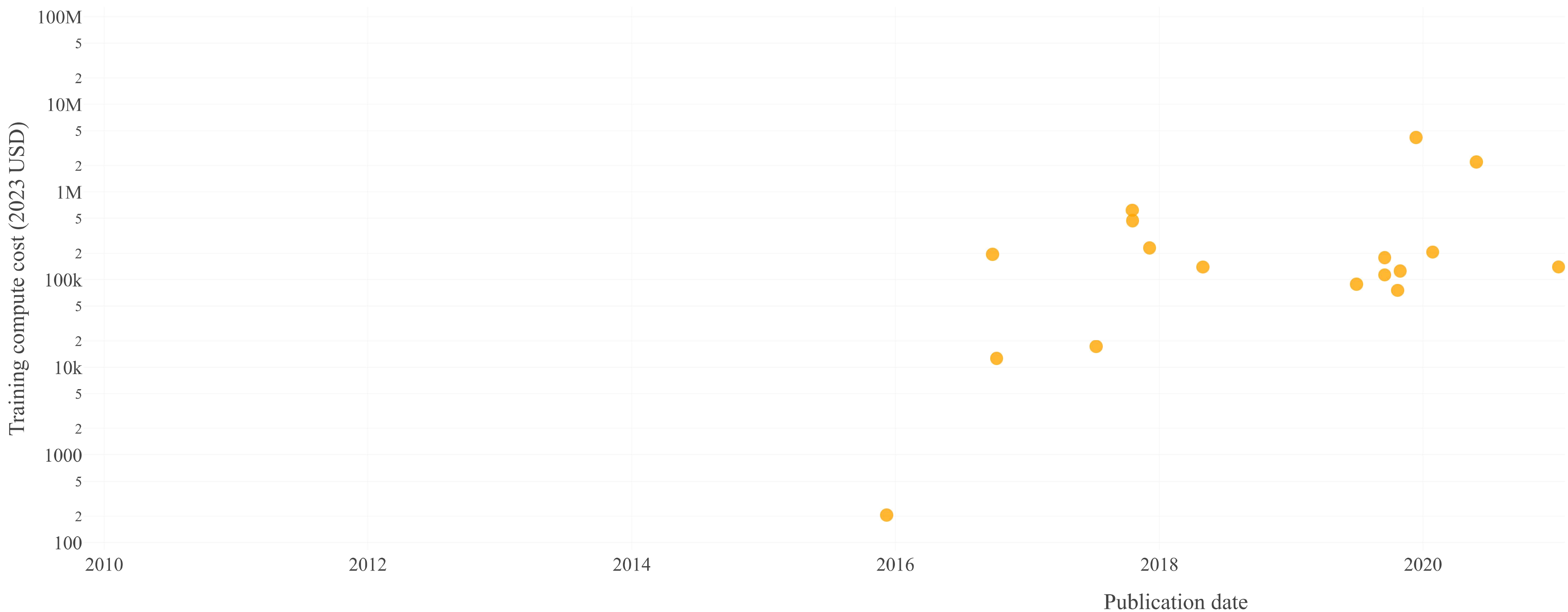
DISTRIBUTED TRAINING: RESNET-50

YET ANOTHER ACCELERATED SGD

Scaling data-parallel SGD slashes ResNet-50/ImageNet training from days to seconds.

Year	Batch Size	Hardware	Library	Time	Accuracy
2015	256	`P100 × 8`	Caffe	29 hrs	75.3 %
2017	8,192	`P100 × 256`	Caffe2	1 hr	76.3 %
2018	8,192 → 16,384	`Full TPU × Pod`	TensorFlow	30 mins	76.1 %
2017	32,768	`P100 × 1,024`	Chainer	15 mins	74.9 %
2018	65,536	`P40 × 2048`	TensorFlow	6.6 mins	75.8 %
2018	65,536	`TPU v3 × 1,024`	TensorFlow	1.8 mins	75.2 %
2019	55,296	`V100 × 3,456`	NNL	2.0 mins	75.29 %
2019	81,920	`V100 × 2048`	MXNet	1.2 mins	75.08 %

TRAINING COST OF FRONTIER MODELS



*Epoch AI: **Key Trends and Figures in Machine Learning***

WHY DISTRIBUTED TRAINING?

- **Exascale compute:** 10^{18} FLOP workloads need multi-node parallelism.
- **Model scale:** Millions → trillions of parameters—beyond single-device RAM.
- **Data volumes:** Petabyte-scale datasets saturate node I/O and storage.
- **HPC & ML:** Coupling large simulations with AI drives heterogeneous scaling.
- **Efficiency:** Distributed frameworks maximize utilization and power on exascale systems.

ALCF, ***Aurora Exascale Supercomputer***

SCIENTIFIC DL AT SCALE

- **Climate Analytics:** Exascale DL for extreme weather modeling (2018)
- **Cancer Research:** Accelerating cancer pathology analysis (2019)
- **Inverse Problems:** Exascale DL for inverse problems (2019)
- **Flood-Filling Networks:** Scaling FFN training on HPC (2019)
- **Dark Energy Survey:** DL at scale for galaxy catalogs (2019)
- **Megatron-LM:** Large-scale transformer training (2021)

Representative publications showcasing scientific deep learning at exascale.

COMMUNICATION LIBRARIES

- Collective ops (all-reduce, all-gather) underpin distributed DL.
- Latency & bandwidth optimizations dictate scaling efficiency.
- Plugins bridge DL frameworks to HPC fabrics transparently.

ONECCL IN DISTRIBUTED TRAINING

- Intel oneAPI Collective Communications Library (oneCCL).
- Optimized for Intel GPUs and CPUs.
- Implements MPI-like collectives with Level Zero & SYCL/DPC++ back-ends.
- Deep integration with PyTorch, TensorFlow, Horovod, IPEX.
- High-throughput collectives over OFI & MPI transport layers.

ONECCL — FEATURE HIGHLIGHTS

- Default hierarchical algorithm (`topo`) optimizes intra-node (scale-up) and inter-node (scale-out) communication
- Collective operations on low-precision datatypes
- Asynchronous progress threads overlap computation and communication
- Unified C and C++ API for host (CPU) and device (GPU) memory buffers

```cpp
// Minimal C++ all-reduce with oneCCL
#include <oneapi/ccl.hpp>

int main(int argc, char** argv) {
  ccl::init();
  auto comm = ccl::create_communicator();
  std::vector send(1024, comm.rank()), recv(1024);
  comm.allreduce(send, recv, ccl::reduction::sum).wait();
  return 0;
}
```

COMMUNICATION LIBRARY LANDSCAPE

- **MPI:** Portable, mature; rich semantics; CPU-centric.
- **NCCL:** NVIDIA GPU collectives; PCIe/NVLink topology-aware.
- **RCCL:** AMD ROCm counterpart to NCCL; HIP-enabled.
- **Gloo:** Simple API; CPU/GPU; best < 1 k ranks.

LIBRARY TRADE-OFFS & SELECTION GUIDE

- **Vendor lock-in:** NCCL (NVIDIA), RCCL (AMD), oneCCL (Intel).
- **Heterogeneous support:** MPI/UCX & oneCCL span CPU + GPU.
- **Ease of integration:** Gloo simple; NCCL/oneCCL have framework plugins.
- **Scalability:** MPI & *CCL proven to 10 k+ GPUs; Gloo ≤ 1 k.

STATE-OF-THE-ART PARALLELISM SCHEMES

- Data Parallelism
  - Distributed Data-Parallel (DDP)
- Model Parallelism
  - Tensor (intra-layer) Parallelism
  - Pipeline (inter-layer) Parallelism
- Hybrid (“3D”) Parallelism

MODEL PARALLELISM OVERVIEW

Splits a model's parameters or ops across devices to handle very large networks.

- **Tensor Parallelism:** shard weight tensors within each layer; all devices work on the same batch.
- **Pipeline Parallelism:** cut the model into sequential stages; devices process different micro-batches in flight.

TENSOR (INTRA-LAYER) PARALLELISM

- Shards each layer’s weight tensors across multiple GPUs.
- GPUs collaborate on the same mini-batch.
- Fine-grained `all-reduce` ops.
- Key benefits:
  - Train layers too large for a single device.
  - Maintains low pipeline latency (no bubbles).
- Best for models with extremely large dense layers.

PIPELINE PARALLELISM

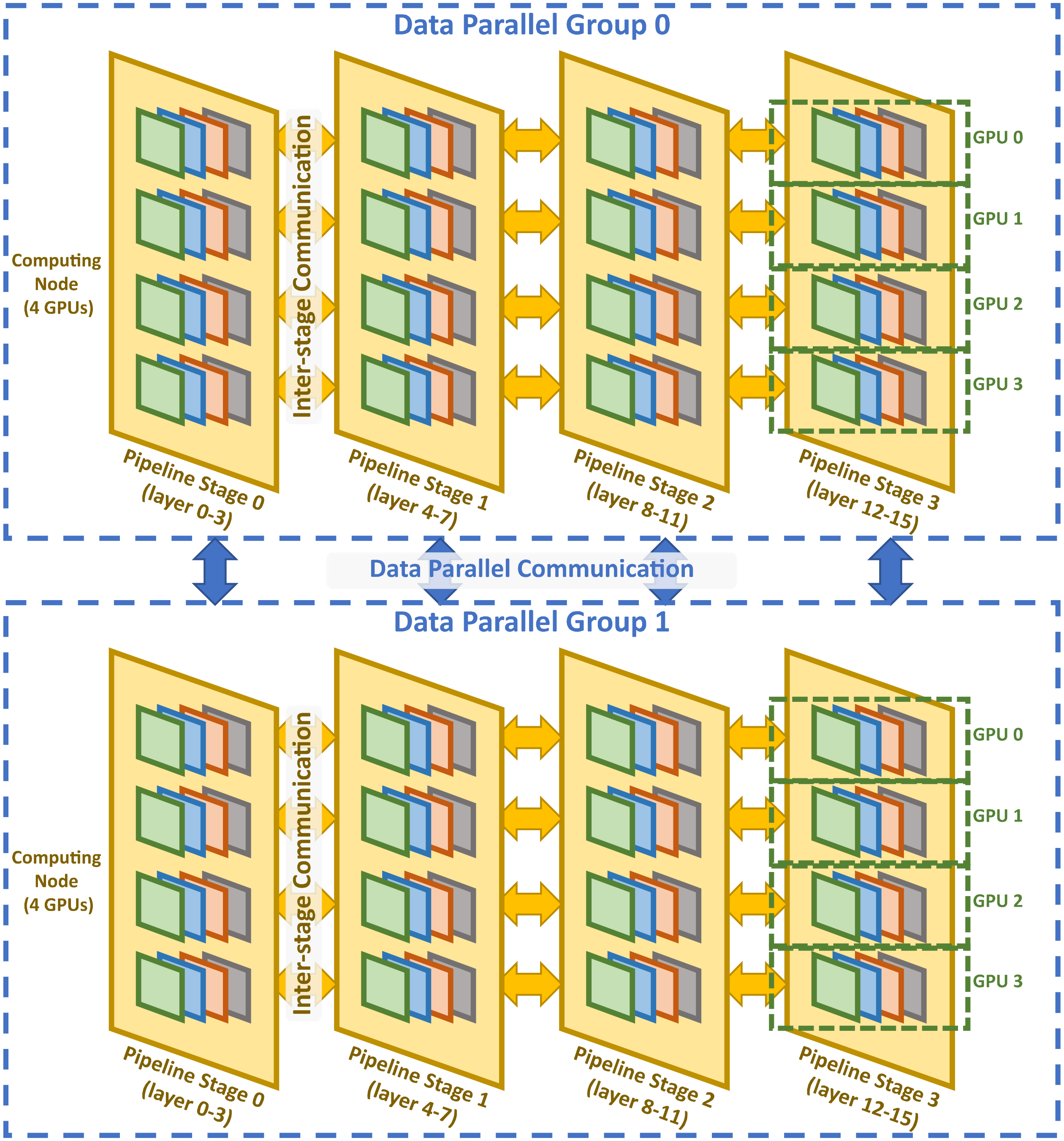
- **Stages:** e.g., layers 1–10 on GPU 0, 11–20 on GPU 1, …
- **Micro-batches:** chunk the batch and stream pieces through stages.
- **Overlap:** compute on one micro-batch overlaps communication of another.
- **Pipeline bubbles:** startup/shutdown idle periods when ramping up/down.

COMPARING PARALLELISM FORMS

Aspect	Tensor Parallelism	Pipeline Parallelism
Granularity	Per-layer shards; sync every op	Only at stage boundaries
Concurrency	All devices on same batch	Different micro-batches on each stage
Overhead	Fine-grained all-reduces	Startup/drain bubbles
Best for	Huge layers, heavy tensor ops	Deep models, balanced stage compute

DATA PARALLEL

- Replicate full model on each worker.
- Each rank processes unique mini-batch shard.
- Gradients all-reduced after backward pass.
- Simple; scales to 10 k+ GPUs.
- Bandwidth-bound at very large scale.


Data Parallel Group 0
Computing
Node
(4 GPUs)
Inter-stage Communication
GPU 0
GPU 1
GPU 2
GPU 3
Pipeline Stage 0
(layer 0-3)
Pipeline Stage 1
(layer 4-7)
Pipeline Stage 2
(layer 8-11)
Pipeline Stage 3
(layer 12-15)
Data Parallel Communication
Data Parallel Group 1
Computing
Node
(4 GPUs)
Inter-stage Communication
GPU 0
GPU 1
GPU 2
GPU 3
Pipeline Stage 0
(layer 0-3)
Pipeline Stage 1
(layer 4-7)
Pipeline Stage 2
(layer 8-11)
Pipeline Stage 3
(layer 12-15)

DATA PARALLEL TRAINING: PYTORCH DDP

LINEAR SCALING RULE

When global batch size multiplies by *k*, scale the learning rate by *k*.

$$w_{t+1} = w_t - \eta \frac{1}{|B|} \sum_{x \in B} \nabla L(x, w_t)$$

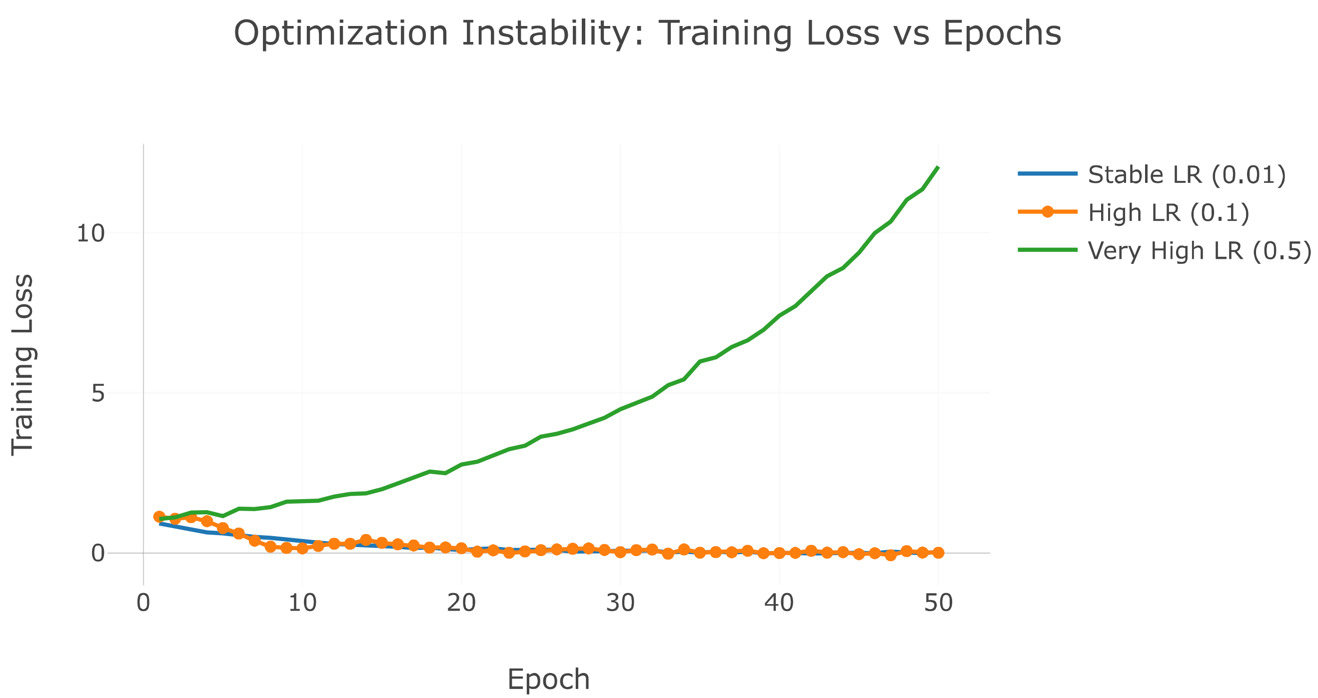
$$\mathbf{lr_{new} = lr_{base} \times world\_size}$$

- Keep local batch size per worker.
- Increase global batch size & learning rate proportionally.

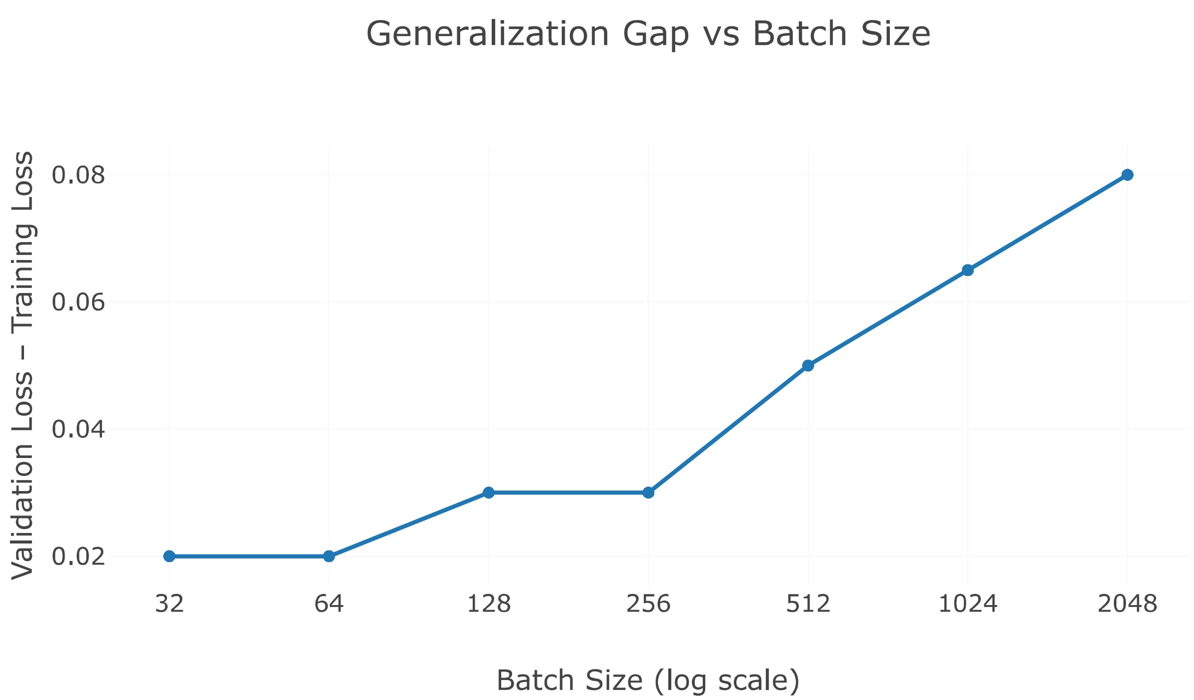
LARGE-BATCH CHALLENGES & SOLUTIONS

- **Optimization Instability** – use LR warm-up (cosine or linear).
- **Generalization Gap** – apply LR decay, regularization, longer warm-up.

OPTIMIZATION INSTABILITY



GENERALIZATION GAP (LOG SCALE)



PYTORCH DDP WORKFLOW

```
import torch
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

dist.init_process_group('nccl')

model = DDP(MyModel().cuda(), device_ids=[local_rank])
optimizer = torch.optim.Adam(model.parameters(), lr=base_lr * world_

for inputs, targets in loader:
    outputs = model(inputs.cuda())
    loss    = criterion(outputs, targets.cuda())
    loss.backward()
    optimizer.step()
```

DATA MANAGEMENT & I/O CHALLENGES

- Growing data volumes (TB–PB) demand efficient ingestion pipelines.
- Complex workflows: preprocessing, augmentation, caching, staging.
- Balancing throughput, latency & compute utilization.

DL I/O TRAITS

Read-Intensive:

Sustained high-throughput reads.

Metadata-Hungry:

Millions of small files & frequent directory ops.

Random & Sparse Access:

Non-sequential reads across dataset.

Multi-format:

Images, JSON, TFRecord, HDF5, custom archives.

Hierarchical Storage:

Leveraging DRAM, SSD/NVMe, parallel file systems.

I/O VS COMPUTE BOTTLENECKS IN DL WORKLOADS

UNet3D

3-D convolutional U-Net for volumetric data.

BERT

Transformer-based language model pre-training on large text corpora.

UNET3D I/O BOTTLENECK ON GPFS

- **I/O-bound:** Storage limits throttle sustained reads.
- **GPU/CPU idle:** Frequent I/O stalls leave compute under-utilized.
- **Weak scaling:** Throughput plateaus as cluster size increases.

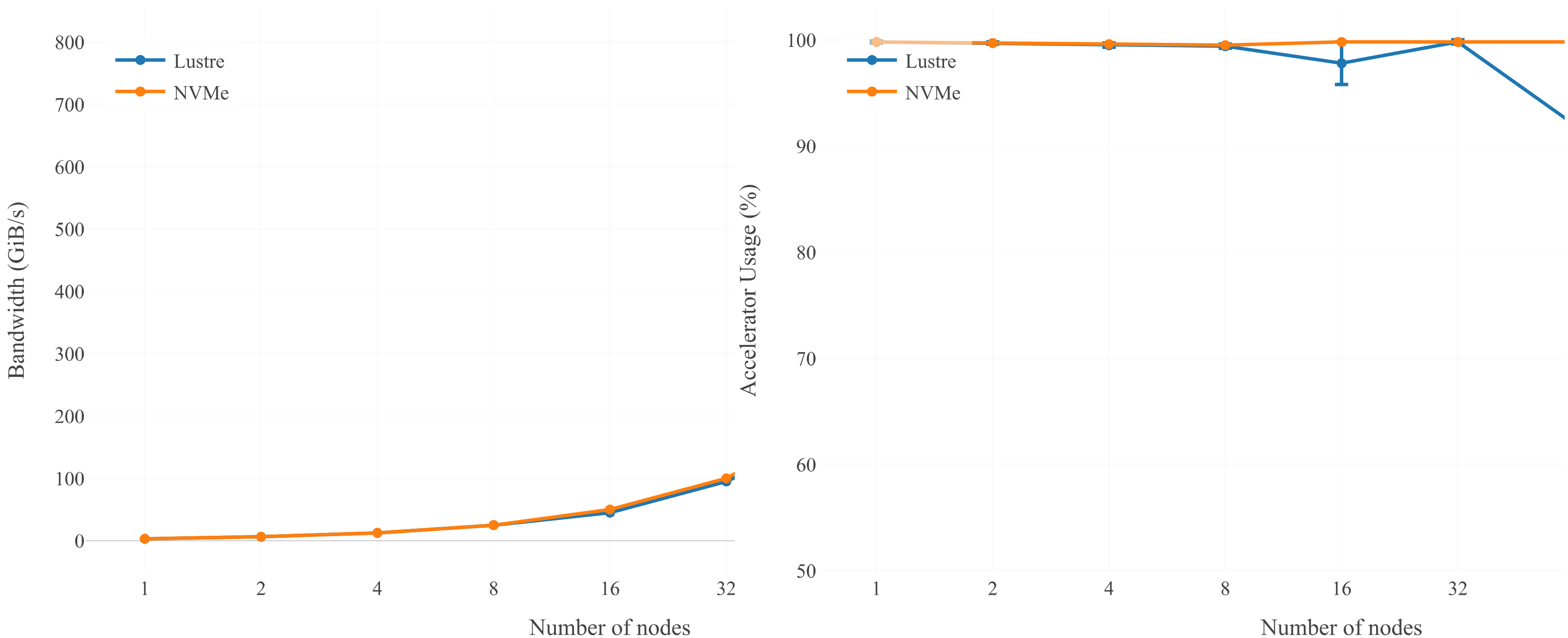
BERT PRE-TRAINING SCALING

- **Compute-bound:** Floating-point workloads saturate GPUs before I/O.
- **Linear weak scaling:** Performance grows nearly linearly with GPUs.
- **I/O overhead:** Well below storage limits, so not the bottleneck.

COMPUTE VS I/O BOUND: KEY TAKEAWAYS

- **Data-intensive (UNet3D):** Prioritize I/O optimizations—caching, parallel reads.
- **Compute-intensive (BERT):** Scale GPU capacity & optimize kernels.
- **Choose your focus:** Storage tuning vs hardware/algorithmic scaling.

UNET3D I/O THROUGHPUT & UTILIZATION



*ALCF, **DLIO Benchmark** (Polaris)*

OPTIMIZING DATA PIPELINES

- **Efficient Formats:** preprocess to TFRecord/LMDB or binary archives.
- **Sharding & Layout:** pack samples per file, bucket by size, shard across workers.
- **Parallel I/O:** async prefetch, multi-thread/process workers.
- **Caching & Staging:** in-memory buffers, SSD/NVMe lanes, burst buffers.
- **Filesystem Tuning:** stripe count/size, object-store optimizations.

SUMMARY

- **Scaling:** Multi-node training for ever-larger models
- **Communication:** Optimized collectives & hybrid parallelism
- **Best Practices:** LR scaling, overlap & low-precision ops
- **Data Pipeline:** Sharding, caching & parallel I/O