Qualcomm

Pretraining LLMs at Scale: Tuning Strategies and Performance Portability

Adrián Pérez Diéguez
Staff Engineer, Qualcomm AI Research

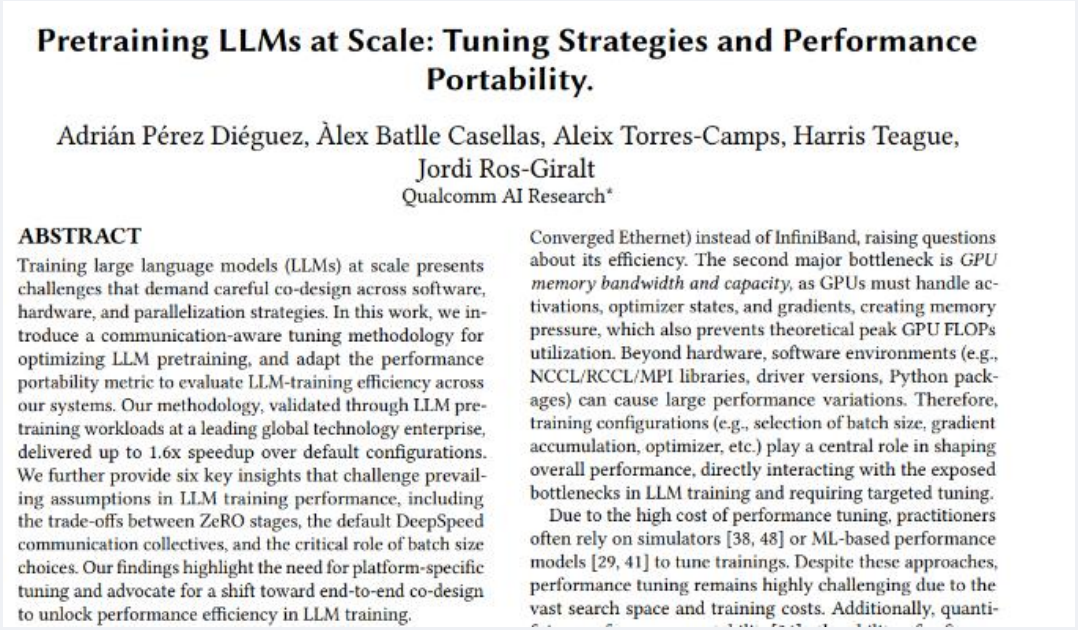
SC PMBS Workshop
November 17th, 2025, St. Louis, Missouri, USA

Pretraining LLMs at Scale

Goal: Optimize LLM training at scale

Contribution: Provide a tuning methodology that accelerates multi-node training across different platforms

More details in paper:



Pretraining LLMs at Scale: Tuning Strategies and Performance Portability.

Adrián Pérez Diéguez, Àlex Batlle Casellas, Aleix Torres-Camps, Harris Teague, Jordi Ros-Giralt
Qualcomm AI Research*

ABSTRACT

Training large language models (LLMs) at scale presents challenges that demand careful co-design across software, hardware, and parallelization strategies. In this work, we introduce a communication-aware tuning methodology for optimizing LLM pretraining, and adapt the performance portability metric to evaluate LLM-training efficiency across our systems. Our methodology, validated through LLM pretraining workloads at a leading global technology enterprise, delivered up to 1.6x speedup over default configurations. We further provide six key insights that challenge prevailing assumptions in LLM training performance, including the trade-offs between ZeRO stages, the default DeepSpeed communication collectives, and the critical role of batch size choices. Our findings highlight the need for platform-specific tuning and advocate for a shift toward end-to-end co-design to unlock performance efficiency in LLM training.

Converged Ethernet) instead of InfiniBand, raising questions about its efficiency. The second major bottleneck is *GPU memory bandwidth and capacity*, as GPUs must handle activations, optimizer states, and gradients, creating memory pressure, which also prevents theoretical peak GPU FLOPs utilization. Beyond hardware, software environments (e.g., NCCL/RCCL/MPI libraries, driver versions, Python packages) can cause large performance variations. Therefore, training configurations (e.g., selection of batch size, gradient accumulation, optimizer, etc.) play a central role in shaping overall performance, directly interacting with the exposed bottlenecks in LLM training and requiring targeted tuning.

Due to the high cost of performance tuning, practitioners often rely on simulators [38, 48] or ML-based performance models [29, 41] to tune trainings. Despite these approaches, performance tuning remains highly challenging due to the vast search space and training costs. Additionally, quanti-

We will challenge some common assumptions made by practitioners, highlighted in frames and referred as *Takeaways*, during the presentation.

Outline

- Motivation
- LLM Training
- Used Platforms
- Performance Analysis (Model 1)
- Tuning Methodology
- Methodology Evaluation: Experimental (Model 2)
- Conclusions

Motivation

Finding the optimal training configuration is challenging and expensive

What factors influence performance?

- **Internode communication**
  - Partition of data, models, gradients and optimizer states across nodes.
  - Costly collectives (all-reduce, all-gather, …): communication and synchronization.
  - Network stack and topology
- GPU memory bandwidth and capacity
  - Memory footprint: 12 x #parameters + Data + Activations + Buffers
- Software environment and framework setup
  - Framework and package versions: bugs / slowdowns
  - Containers, libraries, drivers.

What/How to tune?

- Complex AI stack + interdependencies
- Simulators / performance models: expensive, vast search space.

Motivation

What factors influence performance?

- **Internode communication**
  - Partition of data, models, gradients and optimizer states across nodes.
  - Costly collectives (all-reduce, all-gather, …): communication and synchronization.
  - Network stack and topology
- GPU memory bandwidth and capacity
  - Memory footprint: 12 x #parameters + Data + Activations + Buffers
- Software environment and framework setup
  - Framework and package versions: bugs / slowdowns
  - Containers, libraries, drivers.

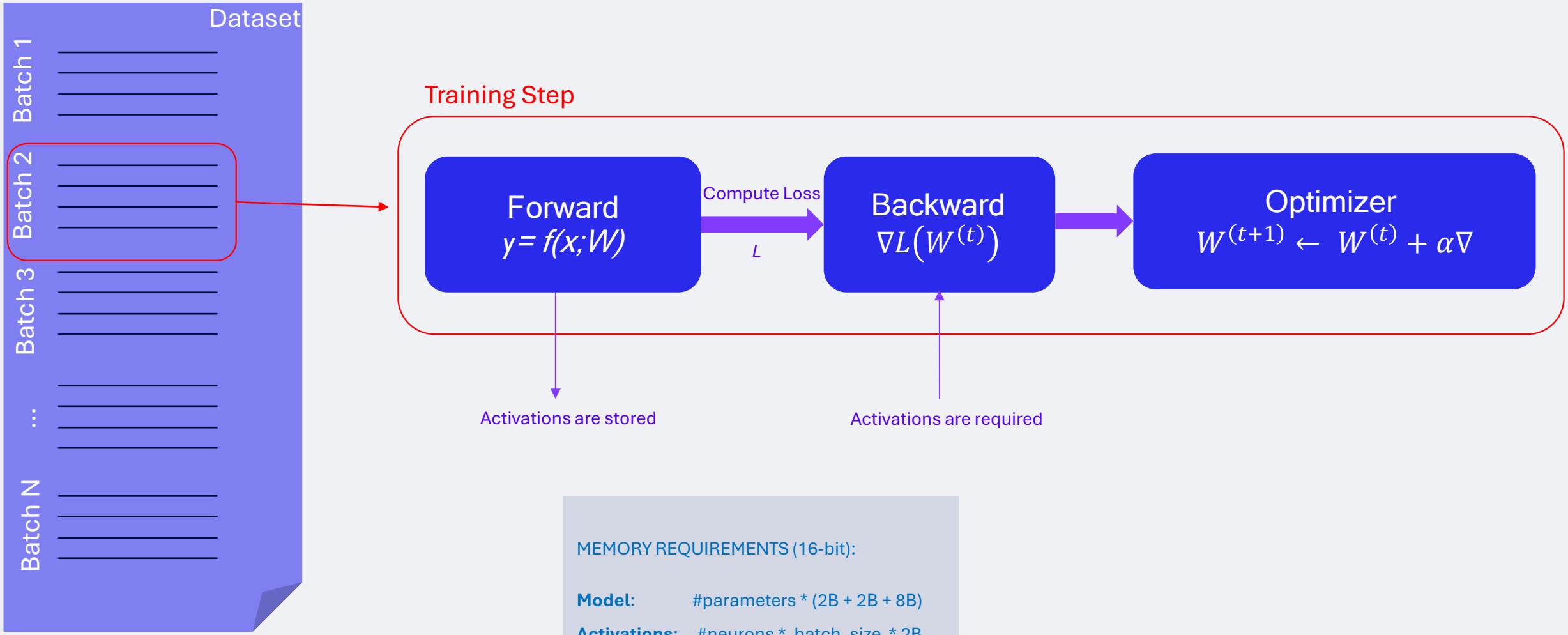
Goal: Optimize LLM training at scale

Contribution: Provide a tuning methodology that accelerates multi-node training across different platforms

What/How to tune?

- Complex AI stack + interdependencies
- Simulators / performance models: expensive, vast search space.

LLM Training



MEMORY REQUIREMENTS (16-bit):

Model: #parameters * (2B + 2B + 8B)

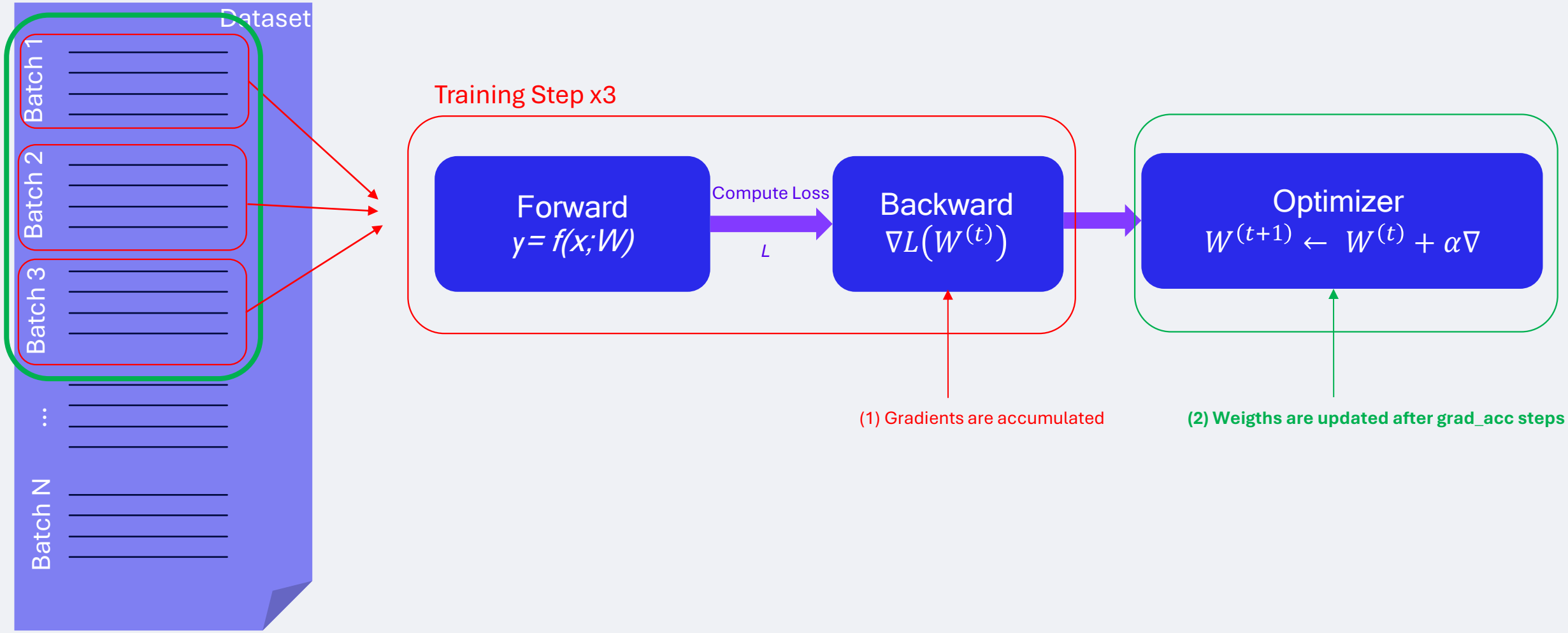
Activations: #neurons * batch_size * 2B

Batch Data

Extra-buffers

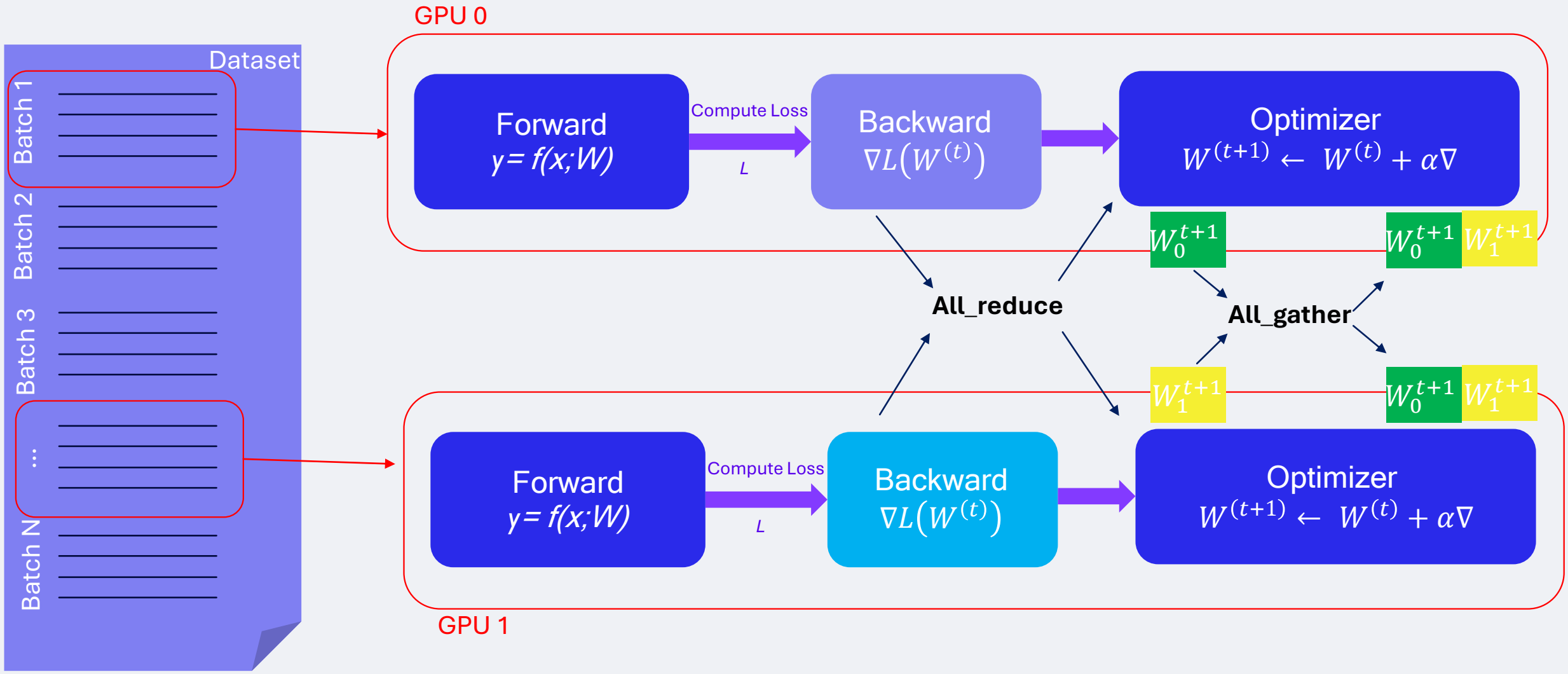
LLM Training

Gradient accumulation accumulates gradientes and updates weights after a few steps



Example: Optimizer is partitioned across multiple nodes

Communication matters in multi-node



DeepSpeed Zero: Memory and communication trade-off

Different multi-node partitions for the model

Name	What's Distributed?	Comm. Collectives
Baseline	Data	all-reduce
ZeRO Stage 1	Data + Optimizer	reduce-scatter + all-gather
ZeRO Stage 2	Data + Opt. + Gradients	reduce-scatter + all-gather
ZeRO Stage 3	Data + Opt. + Grad. + Weights	all-gather + all-gather + reduce-scatter

Evaluation Setup

Used Platforms

- **IB-A100**: IB interconnect 1.6 Tbps, 8x A100 per node. NVlink3 600 GB/s.
- **RoCE-A100**: RoCE interconnect 1.6 Tbps, 8x A100 per node. NVlink3 600 GB/s.
- **RoCE-H100:** RoCE interconnect 1.6 Tbps, 8x H100 per node. NVlink4 900 GB/s.

Models

Models evaluated:

Model 1
- 440m
- LLaMA-based

Model 2
- 8 Billion
- LLaMA-based
- KD from 8B teacher

Evaluation Setup

Used Platforms

- **IB-A100**: IB interconnect 1.6 Tbps, 8x A100 per node. NVlink3 600 GB/s.
- **RoCE-A100**: RoCE interconnect 1.6 Tbps, 8x A100 per node. NVlink3 600 GB/s.
- **RoCE-H100:** RoCE interconnect 1.6 Tbps, 8x H100 per node. NVlink4 900 GB/s.

Models

Models evaluated:

Model 1
- 440m

Model 2
- 8 Billion

Goal: Optimize LLM training at scale

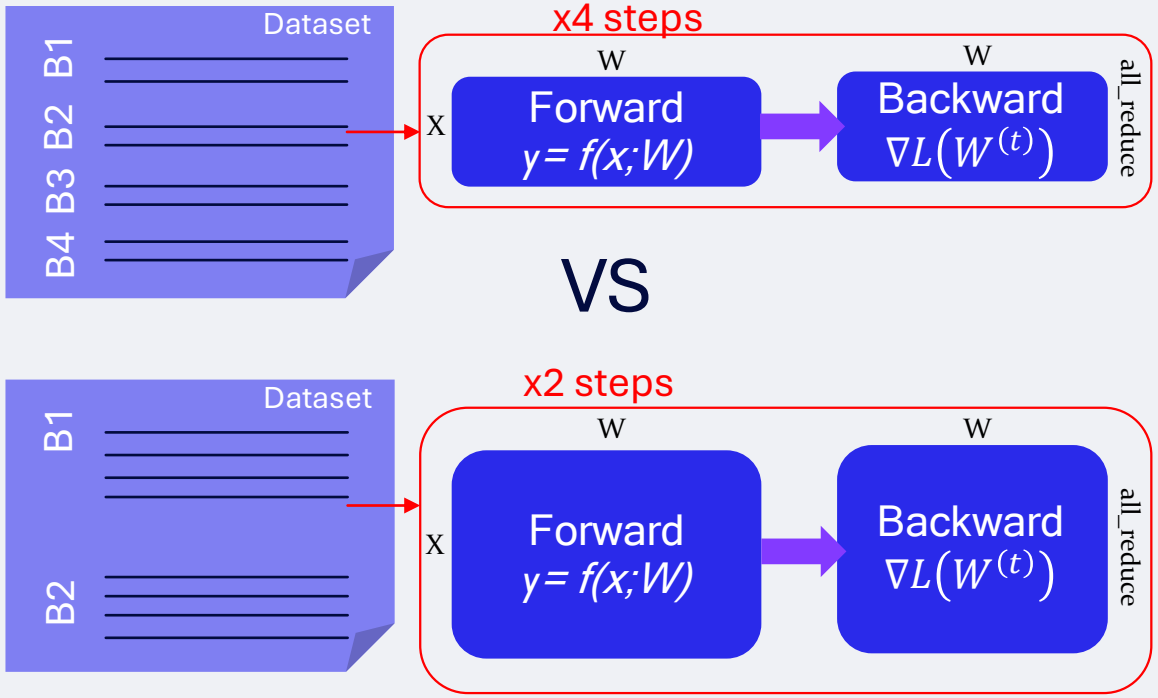
Contribution: Provide a tuning methodology that accelerates multi-node training **across different platforms**

Outline

- Motivation
- LLM Training
- Used Platforms
- Performance Analysis (Model 1)
- Tuning Methodology
- Methodology Evaluation: Experimental (Model 2)
- Conclusions

Performance Analysis: Observations

Increasing batch size reduces the number of training steps (comm. collectives), but each step becomes more compute-intensive.



As many gradients as weights. Therefore, **all-reduce sends the same amount** of data independently of batch size.

Preserve factor *C = batch_size * grad_acc* constant for accuracy.

- Model 1 has *C=1024*

  *e.g. 1024 = 256 * 4 = 512 * 2 = 128 * 8*

Time spent **on optimizer is independent of the batch size,** since it is only determined by model parameters

Optimizer

$$W^{(t+1)} \leftarrow W^{(t)} + \alpha\nabla$$

Therefore, **all-gather sends the same amount** of weights independently of batch size.

Performance Analysis: ZeRO Stage 1

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Averaging time per step every *1024* samples:

Batch_size	Grad_acc	Gradient computation
64	16	123.6
128	8	233.8
256	4	452.8

Makes sense. Doubling batch size, doubles the computation

Performance Analysis: ZeRO Stage 1

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Averaging time per step every *1024* samples:

Batch_size	Grad_acc	Gradient computation	Gradient all-reduce	Optimizer all-gather
64	16	123.6	2.58	4.5
128	8	233.8	16.78	4.51
256	4	452.8	8.07	4.41

Makes sense. Amount of weights does not depend on batch size.

Performance Analysis: ZeRO Stage 1

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Averaging time per step every *1024* samples:

Batch_size	Grad_acc	Gradient computation	Gradient all-reduce	Optimizer all-gather
64	16	123.6	2.58	4.5
128	8	233.8	16.78	4.51
256	4	452.8	8.07	4.41

This should be constant.
Number of gradients does not depend on batch size.

Performance Analysis: ZeRO Stage 1

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Averaging time per step every *1024* samples:

Batch_size	Grad_acc	Gradient computation	Gradient all-reduce	Optimizer all-gather
64	16	123.6	2.58	4.5
128	8	233.8	16.78	4.51
256	4	452.8	8.07	4.41

Let's break down step by step:

grad_acc = 8

Operations	Total Time	Computation	Communication
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2	233.90	233.84	0
bwd_microstep_3	235.06	235.00	0
bwd_microstep_4	235.61	235.56	0
bwd_microstep_5	234.09	233.92	0
bwd_microstep_6	233.82	233.76	0
bwd_microstep_7	290.12	235.80	54.25

all-reduce only happens in the last step (8th) of Stage 1 implementation...

... the larger *grad_acc*, the fewer all-reduces.

We shouldn't just maximize *batch_size*.

TRADE-OFF between *grad_acc* and *batch_size* !

Nodes	Batch Size	Grad ac	ZeRO	Batch Time	Estimated Hours
2	32	32	1	0.106	158.68
2	64	16	1	0.197	**147.05**
2	128	8	1	0.398	148.05
2	256	4	1	0.786	147.08
2	512	2	1	Out of memory	

Performance Analysis: ZeRO Stage 1

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Averaging time per step every *1024* samples:

Batch_size	Grad_acc	Gradient computation	Gradient all-reduce	Optimizer all-gather
64	16			
128	8			
256	4			

Let's break d

Takeaway 1:

When using gradient accumulation and ZeRO Stage 1, maximize batch size is not always the best strategy.

ll-reduce only happens in the last step (8th) of Stage 1
nplementation...

. the larger *grad_acc*, the fewer all-reduces.

Operations			
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2	233.90	233.84	0
bwd_microstep_3	235.06	235.00	0
bwd_microstep_4	235.61	235.56	0
bwd_microstep_5	234.09	233.92	0
bwd_microstep_6	233.82	233.76	0
bwd_microstep_7	290.12	235.80	54.25

grad_acc = 8

We shouldn't just maximize *batch_size*.

TRADE-OFF between *grad_acc* and *batch_size* !

Nodes	Batch Size	Grad ac	ZeRO	Batch Time	Estimated Hours
2	32	32	1	0.106	158.68
2	64	16	1	0.197	**147.05**
2	128	8	1	0.398	148.05
2	256	4	1	0.786	147.08
2	512	2	1	Out of memory	

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Operations	Total Time	Computation	Communication
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2	233.90	233.84	0
bwd_microstep_3	235.06	235.00	0
bwd_microstep_4	235.61	235.56	0
bwd_microstep_5	234.09	233.92	0
bwd_microstep_6	233.82	233.76	0
bwd_microstep_7	290.12	235.80	54.25

Stage 1

Operations	Total Time	Computation	Communication
bwd_microstep_0	465.04	452.29	12.67
bwd_microstep_1	471.9	453.98	17.87
bwd_microstep_2	472.84	450.67	22.07
bwd_microstep_3	470.91	450.05	20.79
bwd_microstep_4	473.14	449.91	23.18
bwd_microstep_5	484.4	449.41	24.93
bwd_microstep_6	467.27	454.16	13.05
bwd_microstep_7	471.32	449.75	21.54

Stage 2

When should we use Stage 1 and when Stage 2?

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Operations	Total Time	Computation	Communication
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2	233.90	233.84	0
bwd_microstep_3	235.06	235.00	0
bwd_microstep_4	235.61	235.56	0
bwd_microstep_5	234.09	233.92	0
bwd_microstep_6	233.82	233.76	0
bwd_microstep_7	290.12	235.80	54.25

Stage 1

65 seconds every *grad_acc* steps

Operations	Total Time	Computation	Communication
bwd_microstep_0	465.04	452.29	12.67
bwd_microstep_1	471.9	453.98	17.87
bwd_microstep_2	472.84	450.67	22.07
bwd_microstep_3	470.91	450.05	20.79
bwd_microstep_4	473.14	449.91	23.18
bwd_microstep_5	484.4	449.41	24.93
bwd_microstep_6	467.27	454.16	13.05
bwd_microstep_7	471.32	449.75	21.54

Stage 2

19 seconds every step

$$19 \times grad\_acc \leq 65$$

Stage 2 becomes potentially better when *grad_acc* < 4

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Stage 2 becomes potentially better when *grad_acc* < 4

Nodes	Batch Size	Grad ac	ZeRO	Batch Time	Estimated Hours
2	32	32	1	0.106	158.68
2	64	16	1	0.197	**147.05**
2	128	8	1	0.398	148.05
2	256	4	1	0.786	147.08
2	512	2	1	Out of memory	
2	32	32	2	0.118	176.64
2	64	16	2	0.211	157.93
2	128	8	2	0.393	150.08
2	256	4	2	0.752	**140.71**
2	512	2	2	Out of memory	

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Stage 2 becomes potentially better when *grad_acc* < 4

Node					
2					
2					
2					
2					
2					
2					
2					
2	128	8	2	0.393	150.08
2	256	4	2	0.752	**140.71**
2	512	2	2	Out of memory	

Takeaway 2:

Stage 2 can outperform Stage 1 and viceversa.

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Operations	Total Time	Computation	Communication
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2	233.90	233.84	0
bwd_microstep_3	235.06	235.00	0
bwd_microstep_4	235.61	235.56	0
bwd_microstep_5	234.09	233.92	0
bwd_microstep_6	233.82	233.76	0
bwd_microstep_7	290.12	235.80	54.25

Stage 1

Operations	Total Time	Computation	Communication
bwd_microstep_0	465.04	452.29	12.67
bwd_microstep_1	471.9	453.98	17.87
bwd_microstep_2	472.84	450.67	22.07
bwd_microstep_3	470.91	450.05	20.79
bwd_microstep_4	473.14	449.91	23.18
bwd_microstep_5	484.4	449.41	24.93
bwd_microstep_6	467.27	454.16	13.05
bwd_microstep_7	471.32	449.75	21.54

Stage 2

Regardless number of all-reduces, why Stage 1's all-reduce is slower than Stage 2's?

NCCL Profiler:

(1) Stage 1 is calling two NCCL's collectives, doubling latency.

(2) Stage 1 and Stage 2 are using NCCL's all-reduce

Performance Analysis: ZeRO Stage 2

Testing different *1024 = batch_size * grad_acc* values for Model 1 on IB-A100

Operations	Total Time	Computation	Communication
bwd_microstep_0	232.46	232.46	0
bwd_microstep_1	234.32	234.23	0
bwd_microstep_2			
bwd_microstep_3			
bwd_microstep_4			
bwd_microstep_5			
bwd_microstep_6			
bwd_microstep_7			

Stage 1

Operations	Total Time	Computation	Communication
bwd_microstep_0	465.04	452.29	12.67
bwd_microstep_1	471.9	453.98	17.87
	472.84	450.67	22.07
	470.91	450.05	20.79
	473.14	449.91	23.18
	484.4	449.41	24.93
	467.27	454.16	13.05
	471.32	449.75	21.54

Takeaway 3:

Contrary to claims made in DS paper, Stage 1 and 2 do not implement reduce-scatter for gradients but all-reduce.

Regardless number of all-reduces, why Stage 1's all-reduce is slower than Stage 2's?

NCCL Profiler:

(1) Stage 1 is calling two NCCL's collectives, doubling latency.

(2) Stage 1 and Stage 2 are using NCCL's all-reduce

Tuning Methodology: Communication-aware

Collectives control overall performance fluctuation

Isolate collective performance in the system.

- Run standalone NCCL all-reduce test and tune its perf. parameters (NCCL_ALGO, NCCL_SOCKET_NTHREADS, NCCL_MIN_NCHANNELS, etc)
- Empirical search if doable; Bayesian Optimization otherwise

Explore parallelism strategies

- Evaluate different *grad_acc*, *batch_size* and ZeRO stages
– Empirical search if doable; Bayesian Optimization otherwise

Tuning of other DS communication-related parameters

- *overlap_comm*
- Hack DeepSpeed code to use *reduce-scatter**

* Check Appendix B in paper

Takeaway 4:

Tuning NCCL performance parameters yields minimal performance gains at moderate scale, but beneficial at larger scales.

Outline

- Motivation
- LLM Training
- Used Platforms
- Performance Analysis (Model 1)
- Tuning Methodology
- Methodology Evaluation: Experimental (Model 2)
- Conclusions

Methodology Evaluation

Testing methodology for training Model 2 on three platforms

C = 256 = batch_size x grad_acc

Step 1. Tune standalone NCCL all-reduce

Category	NCCL Parameter
Algorithm Selection	NCCL_ALGO
Cross-NIC Communication	NCCL_CROSS_NIC
Threading Parameters	NCCL_NTHREADS, NCCL_SOCKET_NTHREADS
Socket/Channel Configuration	NCCL_NSOCKS_PERTHREAD, NCCL_MIN_NCHANNELS
Interface and Stack Settings	NCCL_SOCKET_IFNAME, NCCL_IB_DISABLE, NCCL_CHECK_DISABLE, NCCL_SET_STACK_SIZE

Methodology Evaluation

Testing methodology for training Model 2 on three platforms

C = 256 = batch_size x grad_acc

Step 1. Tune standalone NCCL all-reduce

Category	NCCL Parameter
Algorithm Selection	NCCL_ALGO
Cross-NIC Communication	NCCL_CROSS_NIC
Threading Parameters	NCCL_NTHREADS, NCCL_SOCKET_NTHREADS
Socket/Channel Configuration	NCCL_NSOCKS_PERTHREAD, NCCL_MIN_NCHANNELS
Interface and Stack Settings	NCCL_SOCKET_IFNAME, NCCL_IB_DISABLE, NCCL_CHECK_DISABLE, NCCL_SET_STACK_SIZE

Step 2. Evaluate parallelism strategies

Platform	DS Stage	Batch Size	Grad Acc	Samples/s
IB-A100	1	64	4	66.78
	1	128	2	OOM
	2	64	4	62.131
	2	**128**	**2**	**76.489**
	2	256	1	OOM
	3	64	4	23.680
	3	128	2	39.862
	3	256	1	49.131
RoCE-A100	1	64	4	58.51
	1	128	2	OOM
	2	64	4	25.741
	2	**128**	**2**	**72.891**
	2	256	1	OOM
	3	64	4	18.89
	3	128	2	35.33
	3	256	1	47.89
RoCE-H100	1	64	4	92.35
	1	128	2	OOM
	2	64	4	84.07
	2	**128**	**2**	**134.00**
	2	256	1	OOM
	3	64	4	25.12
	3	128	2	44.76
	3	256	1	68.53

In all platforms, grad_acc = 2 and Stage 2 got best results.

Methodology Evaluation

Testing methodology for training Model 2 on three platforms

C = 256 = batch_size x grad_acc

Step 3. Tune DS other communication-related parameters.

Platform	Baseline samples/s	Tuning samples/s	Overlap_comm?	Best Collective
IB-A100	65.78	90.71	Disabled	Reduce-Scatter
RoCE-A100	58.51	79.9	Disabled	Reduce-Scatter
RoCE-H100	92.35	146.37	Enabled	Reduce-Scatter

* Baseline config. Uses ZeRO Stage 1 with the largest batch size possible, the default all-reduce collective and overlap_comm enable

Methodology Evaluation

Testing methodology for training Model 2 on three platforms

C = 256 = batch_size x grad_acc

Step 3. Tune DS other communication-related parameters.

Platform	Baselin samples
IB-A100	65.78
RoCE-A100	58.51
RoCE-H100	92.35

Takeaway 5:

Performance tuning is necessary, and each model and platform require separated tuning searches.

* Baseline config. Uses ZeRO Stage 1 with the largest batch size possible, the default all-reduce collective and overlap_comm enable

Methodology Evaluation

Testing methodology for training Model 2 on three platforms

C = 256 = batch_size x grad_acc

Step 3. Tune DS other communication-related parameters.

Platform	Baseline samples/s	Tuning samples/s	Overlap_comm?	Best Collective
IB-A100	65.78	90.71	Disabled	Reduce-Scatter
RoCE-A100	58.51	79.9	Disabled	Reduce-Scatter
RoCE-H100	92.35	146.37	Enabled	Reduce-Scatter

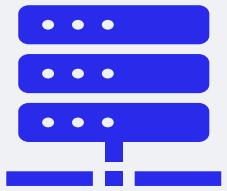
1) H100 platform trained *1.85x* faster than A100s
2) Tunning communication-related parameters boost training throughput
3) Default *all-reduce* does not provide best results in any platform.
4) Optimal configuration varies by platform.
5) IB-A100 platform slightly outperforms RoCE-A100.
6) Disabling RDMA and using TCP led to avg. *12x* slowdown for backward pass.

* Baseline config. Uses ZeRO Stage 1 with the largest batch size possible, the default all-reduce collective and overlap_comm enable

Conclusions

Our findings challenge many common assumptions in LLM training.

Maximize batch size is not always the best strategy

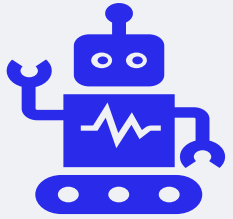


RoCE-based clusters can achieve similar performance to IB-based. TCP is a perf. killer

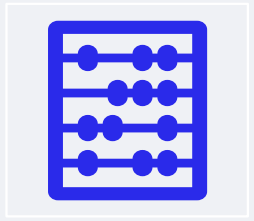
Achieving full compute and communication overlapping remains a fundamental challenge

Communication collectives are responsible for most overall performance fluctuation

Default DeepSpeed implementation uses *all-reduce* for gradient reduction



Separated performance tuning is required per model and platform



Our 3-step methodology delivered 1.6x over baseline

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog