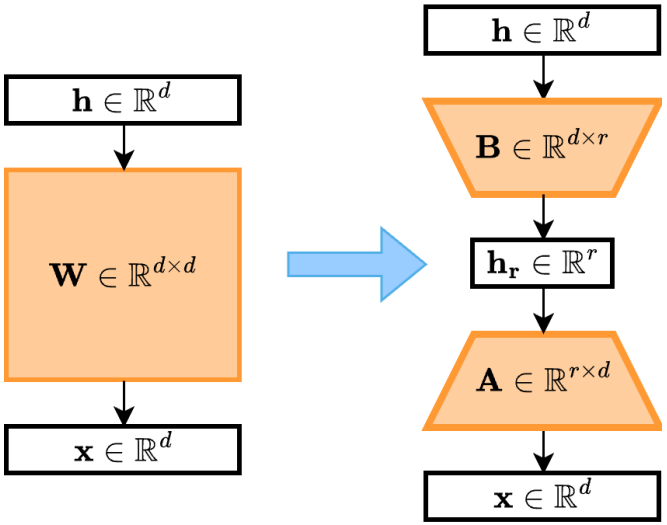
BOOST: BOttleneck-Optimized Scalable Training Framework for Low-Rank Large Language Models

Zhengyang Wang*, Ziyue Liu*, Ruijie Zhang, Avinash Maurya, Paul Hovland, Bogdan Nicolae, Franck Cappello, Zheng Zhang
UC Santa Barbara & Argonne National Laboratory
MLSys 2026

Argonne NATIONAL LABORATORY
UC SANTA BARBARA

Low-Rank Bottleneck Architecture



Representative Low-Rank LLM Methods

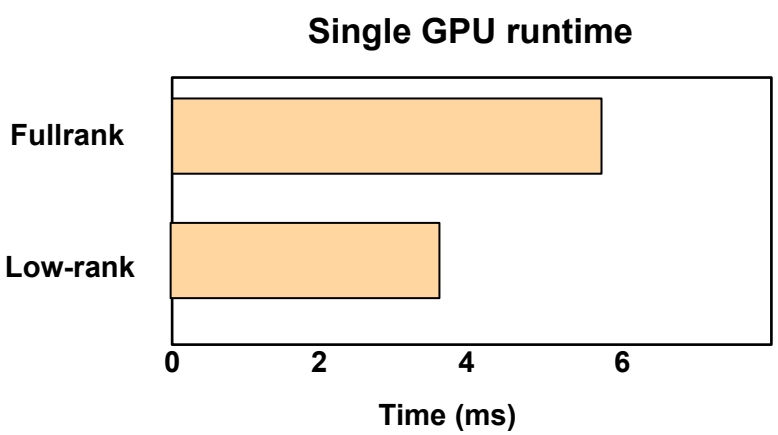
- CoLA[1]: Low-rank **activations**
- LaX [2]: Low-rank **residuals**
- LORO [3]: Low-rank Riemannian **optimizer**

Benefit

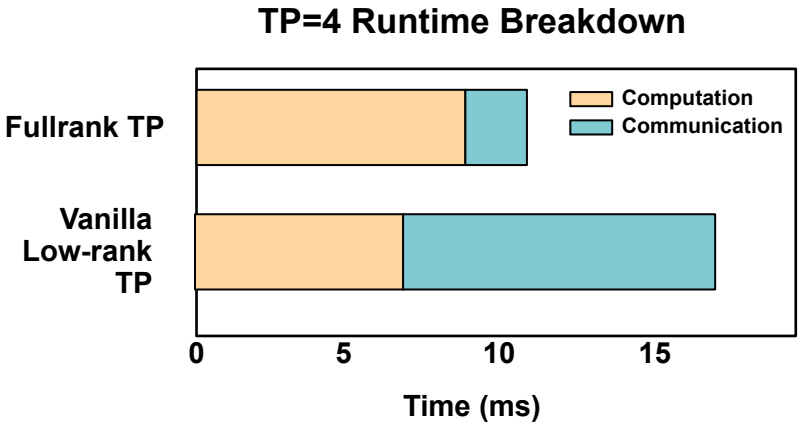
- ~2× fewer **parameters**
- ~2× fewer **FLOPs**
- ~2x lower **model-state memory**
- Comparable or better **perplexity**

[1] Liu et al., CoLA: Compute-Efficient Pre-Training of LLMs via Low-Rank Activation, **EMNLP** 2025.
[2] Zhang et al., LaX: Boosting Low-Rank Training of Foundation Models via Latent Crossing, **Neurips** 2025.
[3] Mo et al., LORO: Low-Rank Riemannian Optimization for LLMs, **ICLR** 2025.

Low-Rank Bottleneck Architecture



~1.6x Speedup On Single GPU

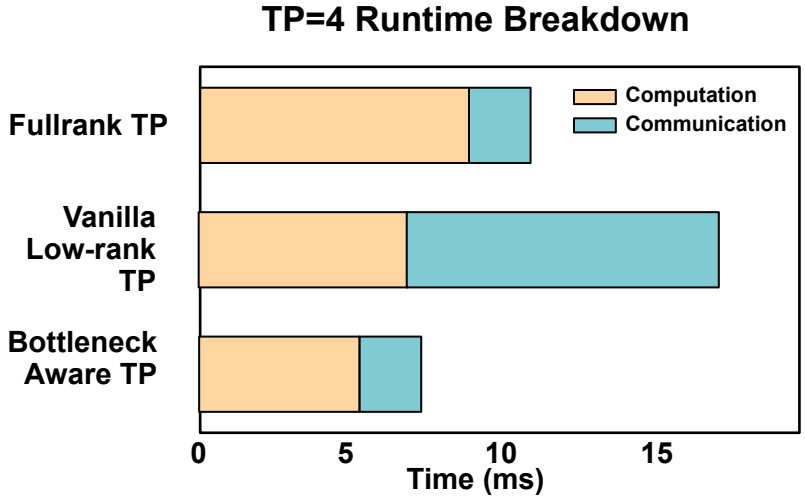


~40% Slowdown On Multi GPU

Communication dominates at scale,
erasing low-rank architectural efficiency

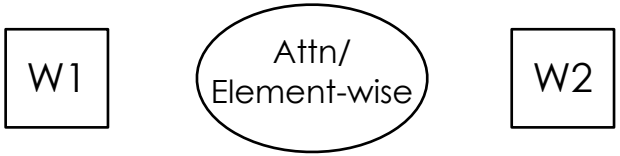
BOOST: Bottleneck-Optimized Scalable Training

- Goal: Preserve bottleneck efficiency in distributed training
- Core contribution
  - Performance Model on bottleneck parallelism efficiency
  - Bottleneck-aware Tensor Parallelism
  - System optimizations to expose end-to-end efficiency



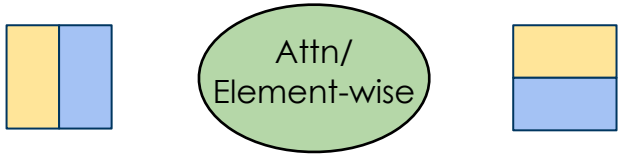
#1: Vanilla TP Inefficiency Analysis

Full-rank TP



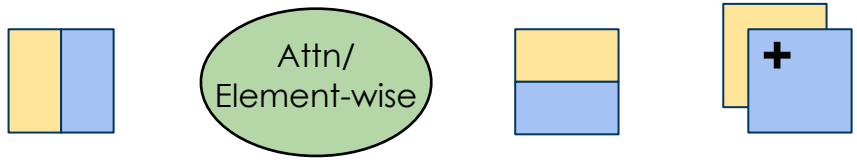
#1: Vanilla TP Inefficiency Analysis

Full-rank TP



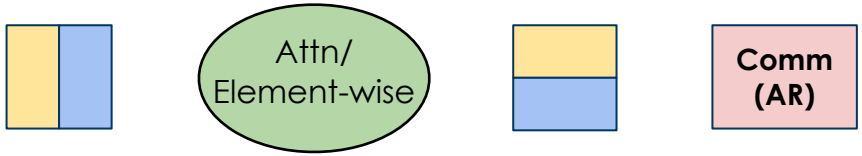
#1: Vanilla TP Inefficiency Analysis

Full-rank TP

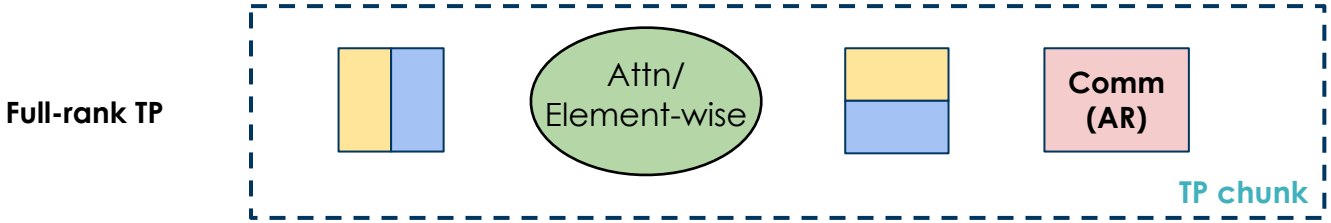


#1: Vanilla TP Inefficiency Analysis

Full-rank TP

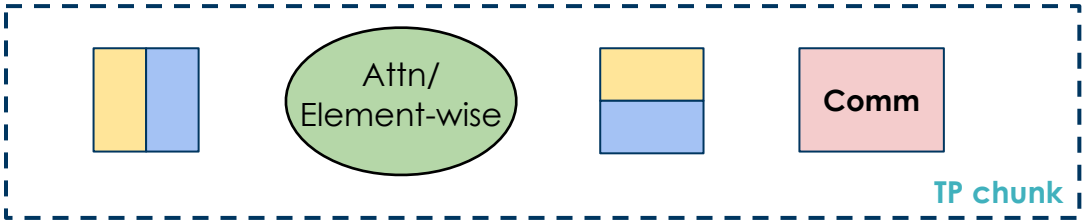


#1: Vanilla TP Inefficiency Analysis

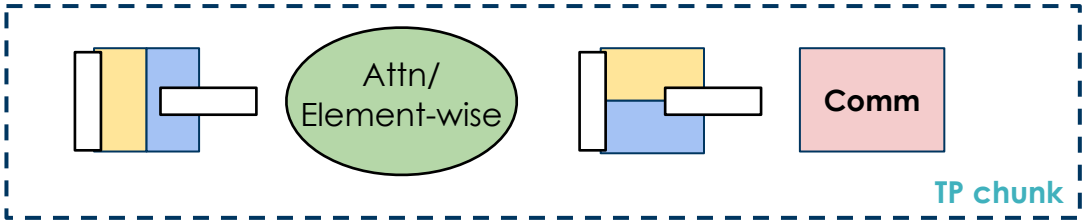


#1: Vanilla TP Inefficiency Analysis

Full-rank TP

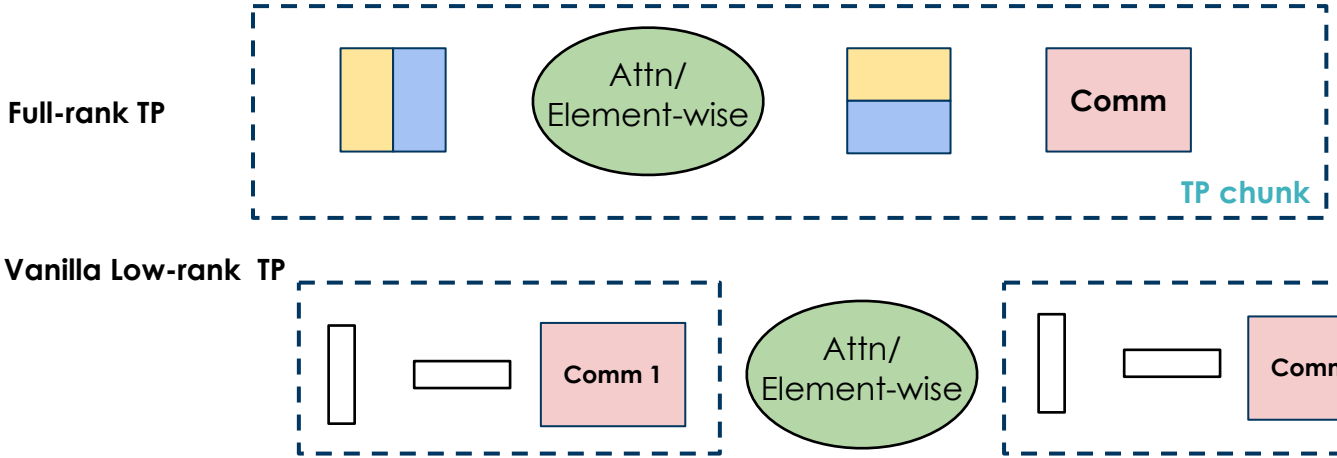


Vanilla Low-rank TP



- Deeper Layer
- Narrow bottleneck

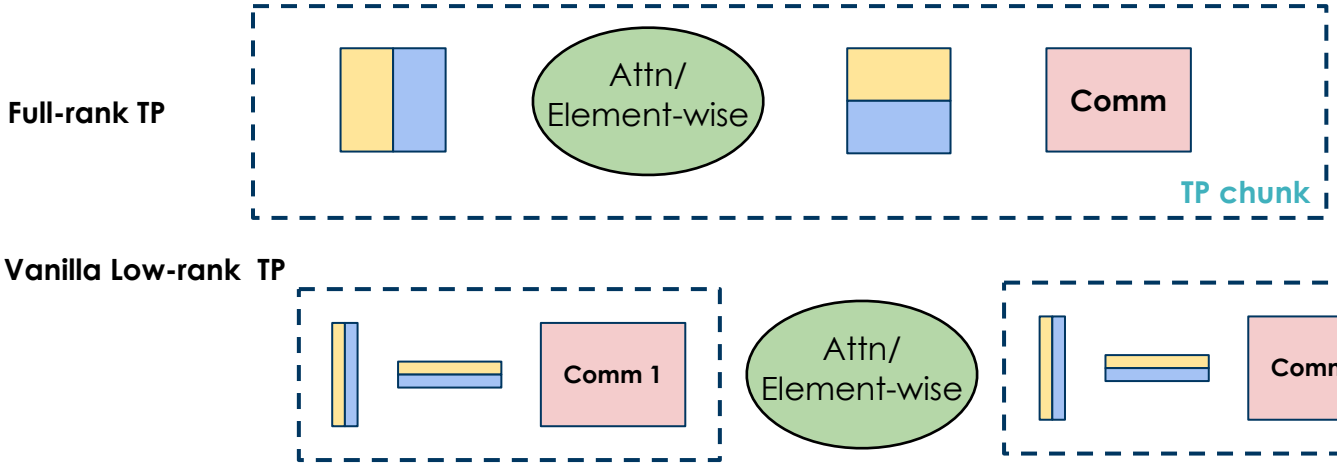
#1: Vanilla TP Inefficiency Analysis



- Deeper Layer
- Narrow bottleneck

Extra Sync Point at full dim
Communication Volume ↑

#1: Vanilla TP Inefficiency Analysis

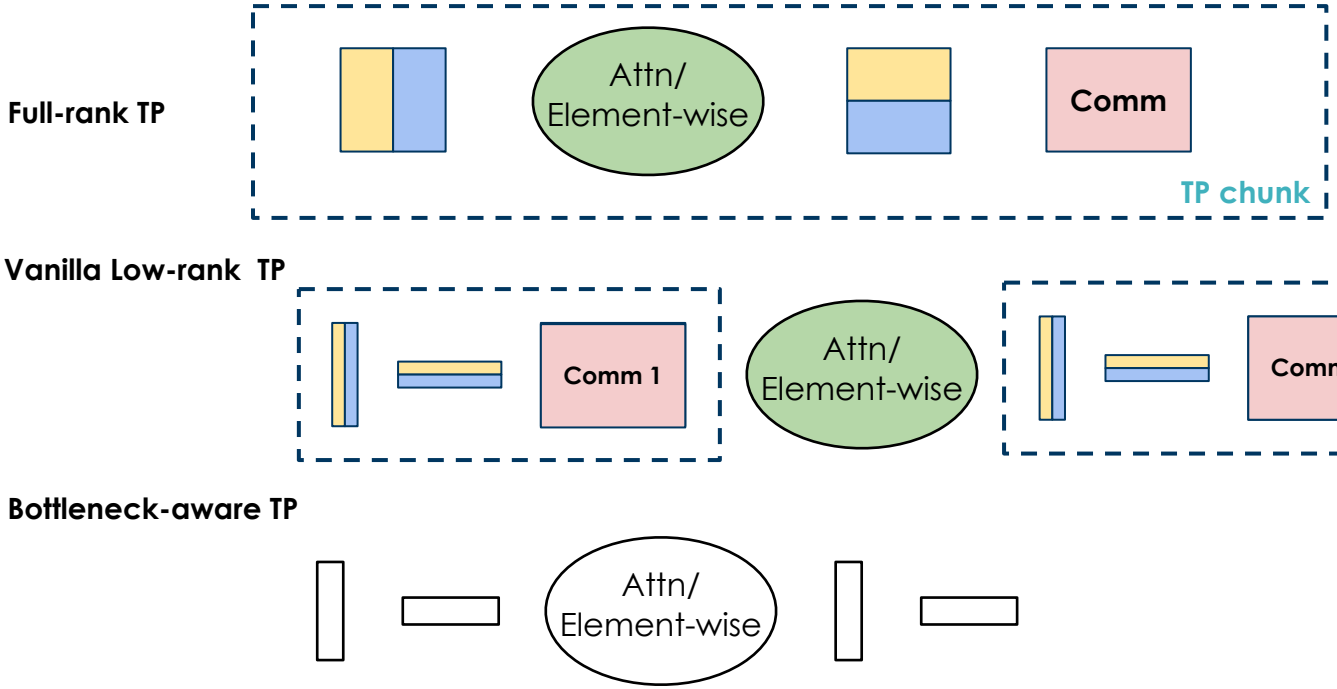


- Deeper Layer
- Narrow bottleneck

Extra Sync Point at full dim
Communication Volume ↑

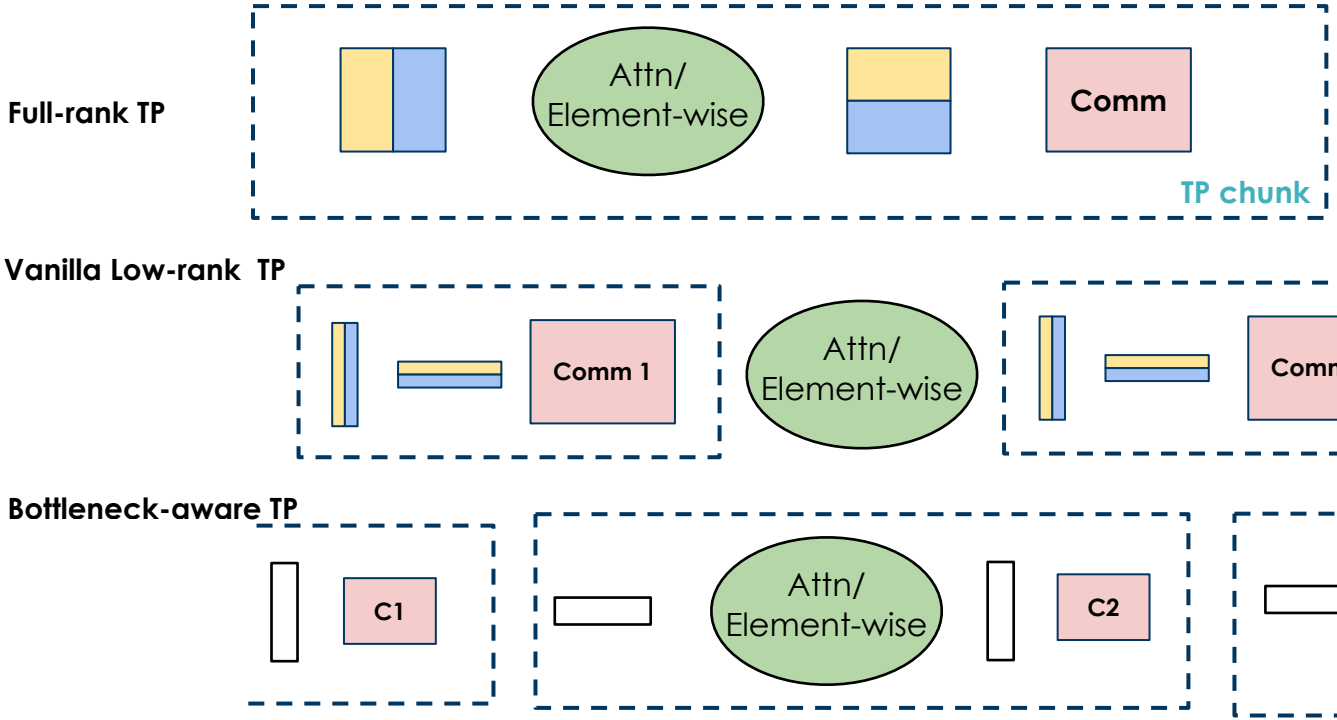
Suboptimal sharded at low rank dim
Arithmetic intensity ↓

#2: Bottleneck-Aware Tensor Parallelism



- **Deeper Layer**
- **Narrow bottleneck**

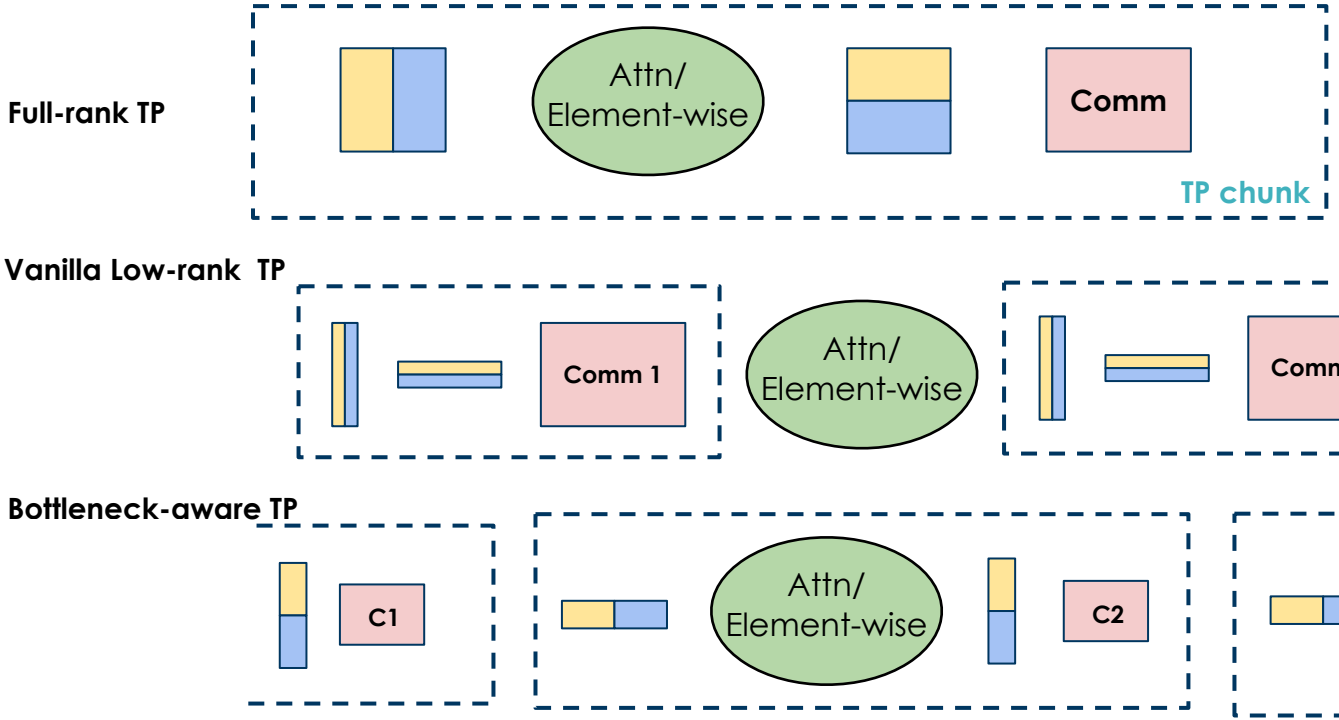
#2: Bottleneck-Aware Tensor Parallelism



- Deeper Layer
- Narrow bottleneck

Extra Sync Point at low-rank dim
Communication Volume ↓

#2: Bottleneck-Aware Tensor Parallelism



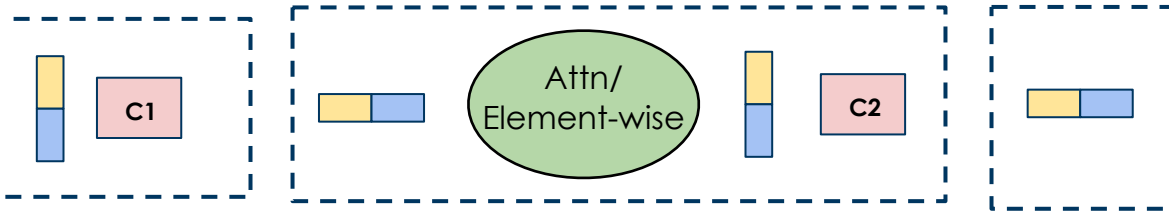
- Deeper Layer
- Narrow bottleneck

Extra Sync Point at low-rank dim
Communication Volume ↓

Healthy sharded at full dim
Arithmetic intensity ↑

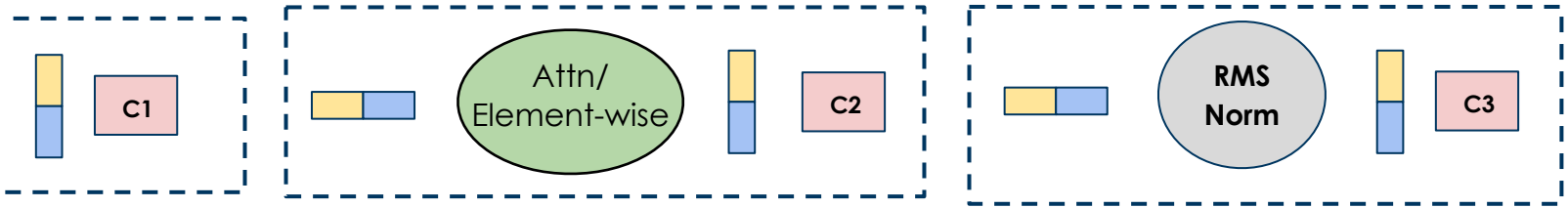
#3: RMSNorm Inside a Sharded BTP Chunk

Bottleneck-aware TP



#3: RMSNorm Inside a Sharded BTP Chunk

Bottleneck-aware TP

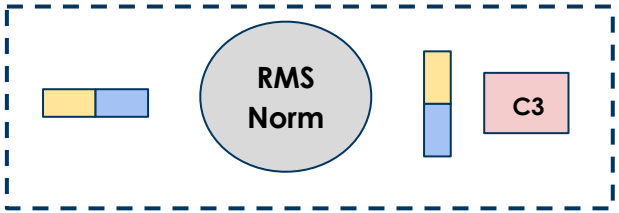


Sharded-Unsafe Operator

$$\mathrm{RMSNorm}(\mathbf{X}) = \frac{\mathbf{X} \cdot \boldsymbol{\gamma}}{\mathrm{RMS}(\mathbf{X})},$$

$$\text{with} \quad \mathrm{RMS}(\mathbf{X}) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} \mathbf{X}_{:,j}^2 + \epsilon}.$$

#3: RMSNorm Inside a Sharded BTP Chunk

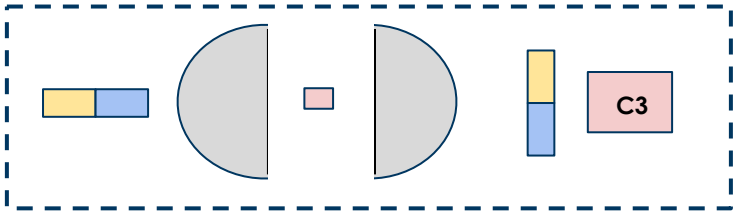


Sharded-Unsafe Operator

$$\text{RMSNorm}(\mathbf{X}) = \frac{\mathbf{X} \cdot \gamma}{\text{RMS}(\mathbf{X})},$$

$$\text{with} \quad \text{RMS}(\mathbf{X}) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} \mathbf{X}_{:,j}^2 + \epsilon}.$$

#3: RMSNorm Inside a Sharded BTP Chunk



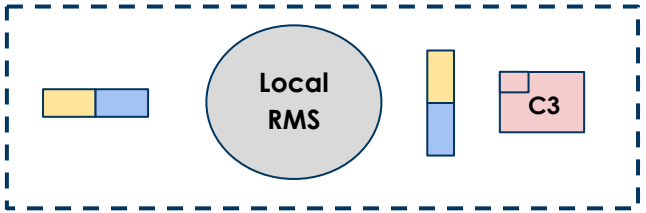
- Variant 1: Sync RMSNorm
  - Explicitly all-reduce RMS statistics: A scalar
  - Correct, but latency-bound

Sharded-Unsafe Operator

$$\mathrm{RMSNorm}(\mathbf{X}) = \frac{\mathbf{X} \cdot \gamma}{\mathrm{RMS}(\mathbf{X})},$$

$$\text{with} \quad \mathrm{RMS}(\mathbf{X}) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} \mathbf{X}_{:,j}^2 + \epsilon}.$$

#3: RMSNorm Inside a Sharded BTP Chunk



- Variant 1: Sync RMSNorm
  - Explicitly all-reduce RMS statistics: A scalar
  - Correct, but latency-bound
- Variant 2: Online RMSNorm
  - Removes the extra small collective by fusing RMS statistics into the next tensor-parallel collective

Sharded-Unsafe Operator

$$\mathrm{RMSNorm}(\mathbf{X}) = \frac{\mathbf{X} \cdot \gamma}{\mathrm{RMS}(\mathbf{X})},$$

$$\text{with} \quad \mathrm{RMS}(\mathbf{X}) = \sqrt{\frac{1}{d}\sum_{j=1}^{d} \mathbf{X}_{:,j}^2 + \epsilon}.$$

$$\sum_i \frac{\mathbf{W_i X_i}}{\mathrm{RMS}(\mathbf{X})} = \left(\sum_i \frac{\mathbf{W_i X_i}}{\mathrm{RMS}(\mathbf{X_i})} \cdot \mathrm{RMS}(\mathbf{X_i})\right) \cdot \frac{1}{\mathrm{RMS}(\mathbf{X})}$$

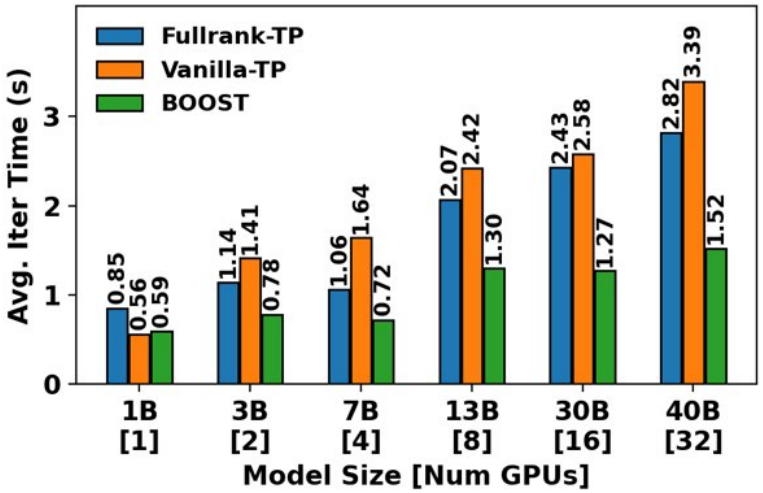
#3: Expose End-to-End Efficiency

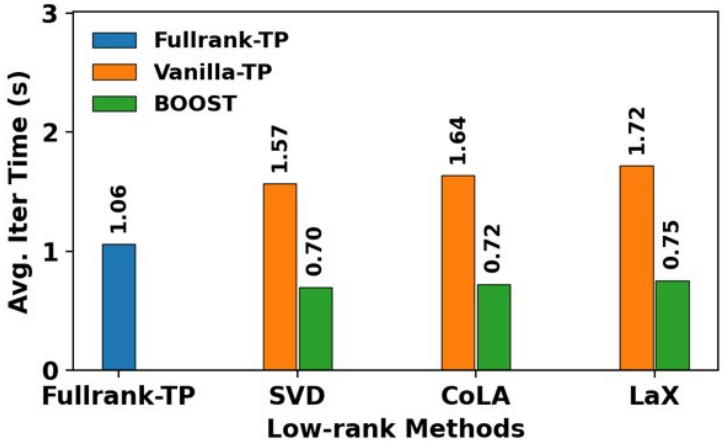
- Online RMSNorm: remove extra small latency-bound collectives
- Layer grouping: improve arithmetic intensity and reduce launch overhead
- Comm-free low-rank checkpointing: store bottleneck activations and communication-free recomputation

Evaluation

- Implementation: Nanotron-based 3D distributed training framework
- Hardware: up to 8 nodes / 32 NVIDIA A100 80GB GPUs
- Models: LLaMA-style 1B - 40B
- Baselines:
  - Full-rank TP
  - Vanilla low-rank TP
  - BOOST with BTP + optimizations
- Architectures tested: SVD-style low-rank, CoLA, and LaX

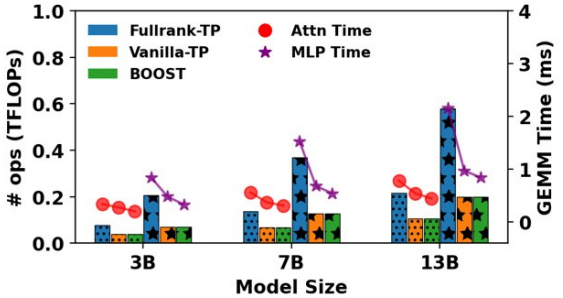
End-to-end: BOOST restores low-rank scaling Efficiency

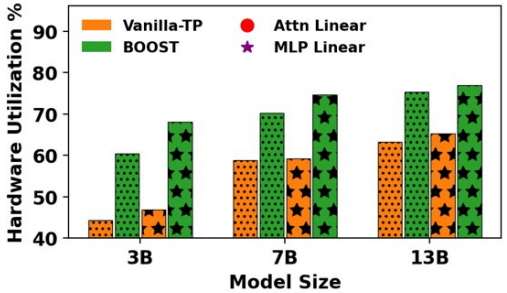


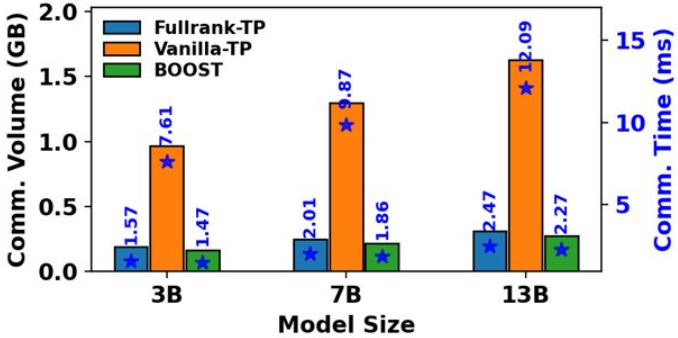


- Across model scale and low-rank variants, BOOST turns algorithmic compression into actual distributed training speedup.
- Achieving 1.46–1.91× faster than FullRank-TP, 1.87–2.27× faster than Vanilla Low-Rank TP

Computation & Communication Efficiency



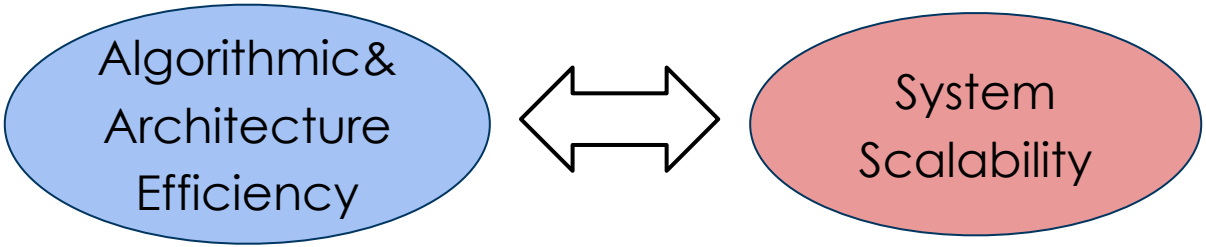




- Better hardware utilization in GEMM and less computation time
- Less communication volume & communication time

Conclusion & Future Work

- Low-rank architecture + Bottleneck-aware parallelism = Scalable efficient Bottleneck LLM training



Code Repo

Paper

Email: zhengyangwang@ucsb.edu

Thanks!