# Adaptive Communication-Aware Pipeline Parallelism for Multimodal Transformers: A Proof-of-Concept Study

Distributed Deep Learning Course Project
Taught by Zhao Zhang

Taqiya Ehsan, Khizar Anjum
{te137, ma1629}@rutgers.edu

Rutgers University
Department of Electrical and Computer Engineering

Fall 2025

# Original Vision: Adaptive Pipeline Parallelism for Multimodal Transformers

**Initial Goal:**

- **Adaptive pipeline parallelism system** for multimodal vision-language models (BLIP)

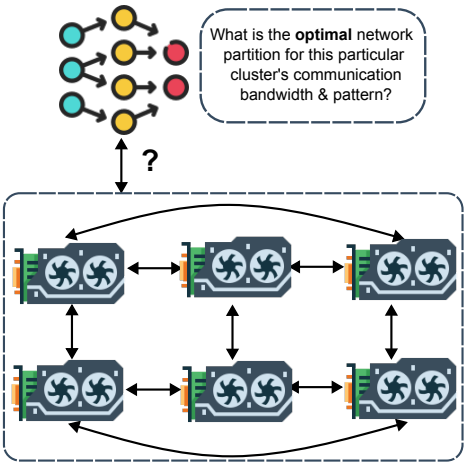



Figure: Envisioned adaptive system responding to network conditions

[a] M. Shoeybi et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019 .

[b] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

# Original Vision: Adaptive Pipeline Parallelism for Multimodal Transformers

**Initial Goal:**

- **Adaptive pipeline parallelism system** for multimodal vision-language models (BLIP)
- **Dynamic adjustment** of pipeline configurations based on:
  - Real-time network conditions
  - Heterogeneous compute patterns

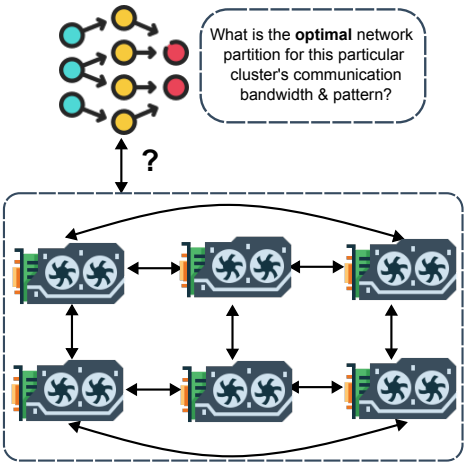



Figure: Envisioned adaptive system responding to network conditions

[a] M. Shoeybi et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019 .

[b] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

# Original Vision: Adaptive Pipeline Parallelism for Multimodal Transformers

**Initial Goal:**

- **Adaptive pipeline parallelism system** for multimodal vision-language models (BLIP)
- **Dynamic adjustment** of pipeline configurations based on:
  - Real-time network conditions
  - Heterogeneous compute patterns
- Address inefficiencies in **static** pipeline implementations

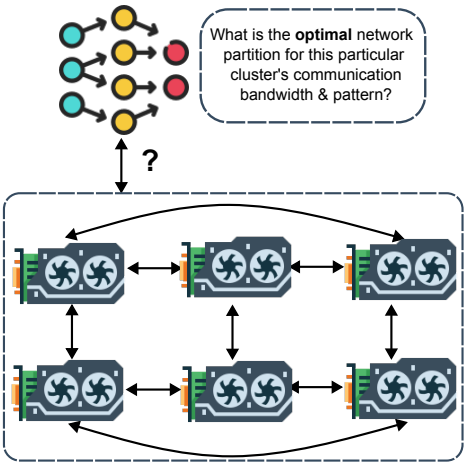



Figure: Envisioned adaptive system responding to network conditions

[a] M. Shoeybi et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019 .

[b] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

# Original Vision: Adaptive Pipeline Parallelism for Multimodal Transformers

**Initial Goal:**

- **Adaptive pipeline parallelism system** for multimodal vision-language models (BLIP)
- **Dynamic adjustment** of pipeline configurations based on:
  - Real-time network conditions
  - Heterogeneous compute patterns
- Address inefficiencies in **static** pipeline implementations
- Extends NVIDIA's Megatron-LM framework[a,b]

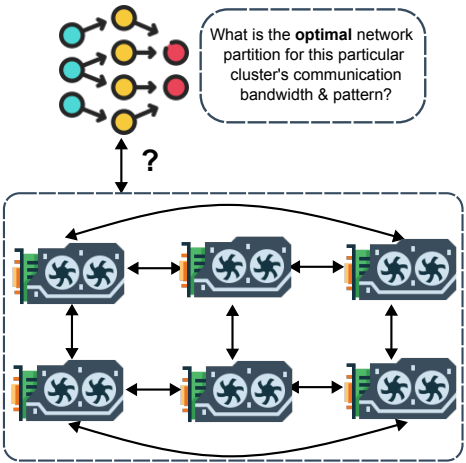



Figure: Envisioned adaptive system responding to network conditions

[a] M. Shoeybi et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019 .

[b] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

# Original Vision: Adaptive Pipeline Parallelism for Multimodal Transformers

**Initial Goal:**

- **Adaptive pipeline parallelism system** for multimodal vision-language models (BLIP)
- **Dynamic adjustment** of pipeline configurations based on:
  - Real-time network conditions
  - Heterogeneous compute patterns
- Address inefficiencies in **static** pipeline implementations
- Extends NVIDIA's Megatron-LM framework[a,b]
- Target: **BLIP** model on **COCO** dataset with throughput optimization

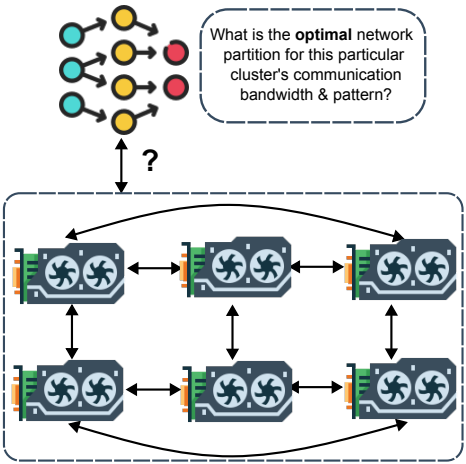



Figure: Envisioned adaptive system responding to network conditions

[a] M. Shoeybi et al., "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019 .

[b] D. Narayanan et al., "Efficient large-scale language model training on gpu clusters using megatron-lm," in *ACM/IEEE SC*, 2021.

# Project Pivot: From Vision to Proof-of-Concept

**Challenges Encountered:**

- BLIP integration with Megatron-LM proved more complex than anticipated
- Megatron-LM multimodal support still under development
- Time constraints for full implementation

# Project Pivot: From Vision to Proof-of-Concept

**Challenges Encountered:**

- BLIP integration with Megatron-LM proved more complex than anticipated
- Megatron-LM multimodal support still under development
- Time constraints for full implementation

**Strategic Pivot:**

- Switched to **Llama 8B** (well-supported in Megatron-LM)
- Focused on **proof-of-concept**: Understanding P2P communication patterns
- Goal: Systematic study of TP/CP/PP/DP configurations

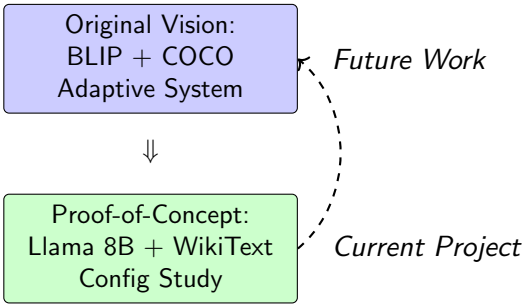



Figure: Project evolution: PoC now, full vision later

# Current Project: Proof-of-Concept Study

**Research Question**

*How do different TP/CP/PP/DP configurations affect P2P communication patterns and training throughput for large language models?*

## Approach

- Grid search of TP/CP/PP/DP configurations
- NCCL P2P communication profiling
- Optimize pipeline stage layout
- Basis for adaptive parallelism systems

## Scope

- **Model:** LLaMA 8B (32 layers, 4096 hidden)
- **Data:** WikiText-103 ($\sim$103M tokens)
- **Compute:** 4–8 GPUs, 1–2 nodes (NERSC Perlmutter)

# Project Progress: What We Built

**Core Infrastructure:**

- Megatron-LM integration with custom P2P profiling
- LLaMA 8B model with TP/CP/PP/DP support
- WikiText-103 dataset preprocessing pipeline
- NCCL communication monitoring + profiling hooks
- Multi-node training scripts (4, 6, 8 GPUs)
- Automated logging + visualization toolkit

# Project Progress: What We Built

**Core Infrastructure:**

- Megatron-LM integration with custom P2P profiling
- LLaMA 8B model with TP/CP/PP/DP support
- WikiText-103 dataset preprocessing pipeline
- NCCL communication monitoring + profiling hooks
- Multi-node training scripts (4, 6, 8 GPUs)
- Automated logging + visualization toolkit

**Experimental Overview:**

- Baseline single-GPU training runs
- TP / PP / DP / CP configuration sweeps
- P2P bandwidth profiling (intra-/inter-node)
- Training convergence + stability verification

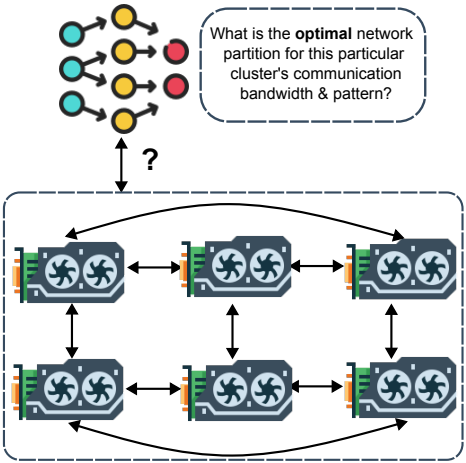
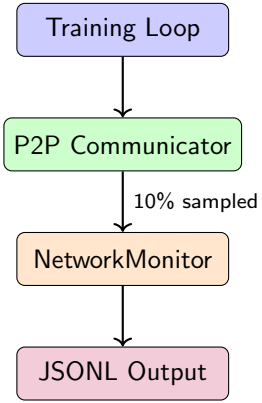



Figure: Pipeline + communication architecture

# P2P Monitoring: Implementation Overview

**Architecture**




**Output**: Per-iteration JSONL with bandwidth, latency, call counts

**Key Implementation Details**

- **Async CUDA Events**
  - Non-blocking timing via `torch.cuda.Event`
  - No synchronization overhead
- **Probabilistic Sampling**
  - 10% of P2P ops monitored
  - Configurable sample rate
- **EMA Smoothing**
  - $\alpha = 0.3$ for stable estimates
  - Handles measurement noise
- **Global Registry**
  - Aggregates stats across all communicators
  - Thread-safe collection

# P2P Monitoring: Why This Works

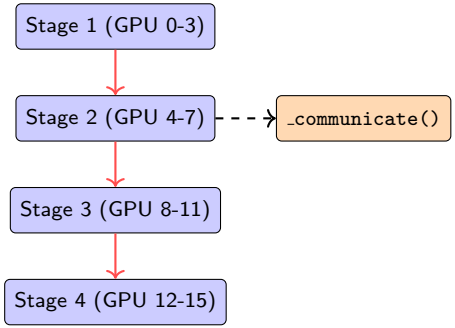
**The Key Insight:** Megatron-LM Has a Single Chokepoint for Pipeline Communication

**How Megatron-LM Moves Data Between Pipeline Stages:**

1. Each pipeline stage needs to send activations
2. **All** this communication goes through one function: `_communicate()`

**Why Only Pipeline Parallelism (PP)?**

- **Tensor Parallelism (TP)**: Uses NCCL all-reduce *within* a node via NVLink — different code path, not P2P
- **Pipeline Parallelism (PP)**: Uses point-to-point sends/recvs *between* nodes — goes through our wrapper




Red arrows = P2P transfers (all go through `_communicate`)

# P2P Monitoring: How We Intercept Communication

**The Wrapper Pattern:** Measure Before and After Each Transfer

## What We Did:

1. Created a new class that *extends* the original communicator
2. Start a timer → call original → stop timer
3. Record: how long it took, how much data moved

## Pseudo-code:

```
def _communicate(self, tensor):
start_timer()
result = super()._communicate(tensor)
stop_timer()
log(time, tensor.size())
return result
```

## Why This Approach?

- **Non-invasive**: We don't modify Megatron's core code
- **Toggle-able**: A config flag switches between normal and monitored mode
- **Low overhead**: Only 10% of calls are timed (random sampling)

## Files We Added:

- `monitored_p2p_communication.py`
- `monitoring_stats_collector.py`

**Total**: ∼400 lines of code, modular design

# Experimental Setup

## Model Configuration

- **Architecture:** LLaMA 8B
  - 32 transformer layers
  - 32 attention heads
  - Hidden dimension: 4096
  - Vocabulary: 32,000 tokens
  - Total parameters: $\sim$8B
- **Dataset:** WikiText-103
  - 103M tokens
  - Long-range dependencies

**Note**: TP = Tensor Parallel, PP = Pipeline Parallel, DP = Data Parallel, CP = Context Parallel

# Experimental Setup

## Model Configuration

- **Architecture:** LLaMA 8B
  - 32 transformer layers
  - 32 attention heads
  - Hidden dimension: 4096
  - Vocabulary: 32,000 tokens
  - Total parameters: ∼8B
- **Dataset:** WikiText-103
  - 103M tokens
  - Long-range dependencies

## 9 Parallelism Configurations Tested

| # | TP | CP | PP | DP | GPUs |
|---|---|---|---|---|---|
| 1 | 4 | 1 | 4 | 1 | 16 |
| 2 | 2 | 2 | 4 | 1 | 16 |
| 3 | 1 | 4 | 4 | 1 | 16 |
| 4 | 1 | 2 | 8 | 1 | 16 |
| 5 | 1 | 1 | 16 | 1 | 16 |
| 6 | 2 | 2 | 2 | 2 | 16 |
| 7 | 4 | 1 | 2 | 2 | 16 |
| 8 | 2 | 1 | 8 | 1 | 16 |
| 9 | 1 | 1 | 8 | 2 | 16 |

**Note**: TP = Tensor Parallel, PP = Pipeline Parallel, DP = Data Parallel, CP = Context Parallel

# Experimental Setup

## Model Configuration

- **Architecture:** LLaMA 8B
  - 32 transformer layers
  - 32 attention heads
  - Hidden dimension: 4096
  - Vocabulary: 32,000 tokens
  - Total parameters: $\sim$8B
- **Dataset:** WikiText-103
  - 103M tokens
  - Long-range dependencies

## 9 Parallelism Configurations Tested

| # | TP | CP | PP | DP | GPUs |
|---|---|---|---|---|---|
| 1 | 4 | 1 | 4 | 1 | 16 |
| 2 | 2 | 2 | 4 | 1 | 16 |
| 3 | 1 | 4 | 4 | 1 | 16 |
| 4 | 1 | 2 | 8 | 1 | 16 |
| 5 | 1 | 1 | 16 | 1 | 16 |
| 6 | 2 | 2 | 2 | 2 | 16 |
| 7 | 4 | 1 | 2 | 2 | 16 |
| 8 | 2 | 1 | 8 | 1 | 16 |
| 9 | 1 | 1 | 8 | 2 | 16 |

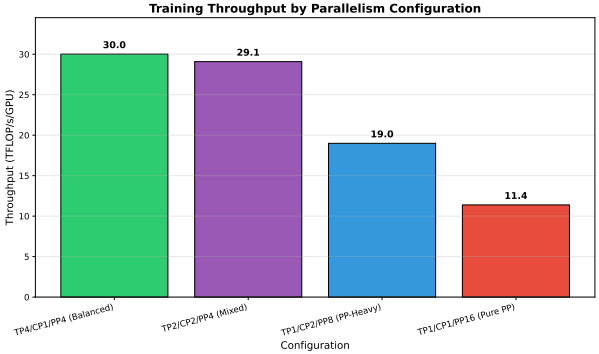
## Training Hyperparameters

- Micro-batch size: 1
- Sequence length: 8192
- Learning rate: 1.5e-4
- Target iterations: 10,000

**Note**: TP = Tensor Parallel, PP = Pipeline Parallel, DP = Data Parallel, CP = Context Parallel

# Results: Training Performance

## Training Throughput by Configuration




## Throughput Comparison

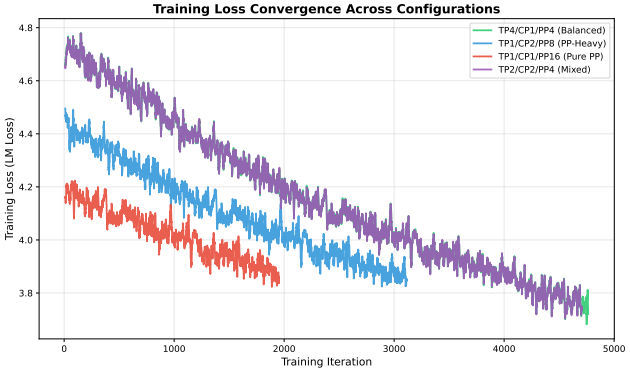
| Config | TFLOP/s/GPU |
|---|---|
| TP4/CP1/PP4 (Balanced) | 30.0 |
| TP2/CP2/PP4 (Mixed) | 29.1 |
| TP1/CP2/PP8 (PP-Heavy) | 19.0 |
| TP1/CP1/PP16 (Pure PP) | 11.4 |

## Key Observations:

- **TP4/CP1/PP4** achieved highest throughput (30 TFLOP/s)
- Pure PP (PP=16) shows 62% lower throughput due to pipeline bubbles
- Balanced TP/PP configs outperform extremes

# Results: Training Convergence

## Training Loss Over Iterations




## Convergence Validation

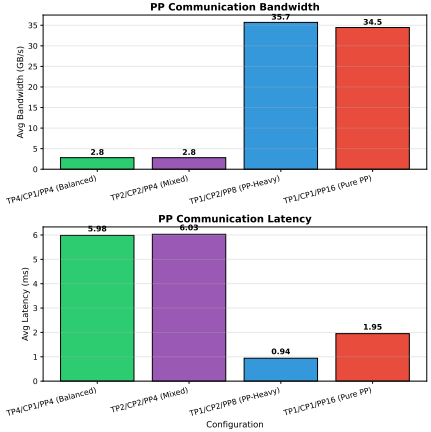
- All configurations converge to similar loss values ($\sim$3.6-4.0)
- **Parallelism strategy does not affect model quality**—only training speed
- Loss curves show expected LLM training behavior

## Final Loss Comparison

| Config | Final Loss |
|---|---|
| TP4/CP1/PP4 | 3.65 |
| TP1/CP1/PP16 | 4.00 |
| TP1/CP2/PP8 | 3.92 |

# Results: Pipeline P2P Communication Patterns

## Pipeline Stage Communication Metrics




*Note: Measures inter-stage PP communication only. TP uses intra-node NVLink (not monitored).*

## Key Findings

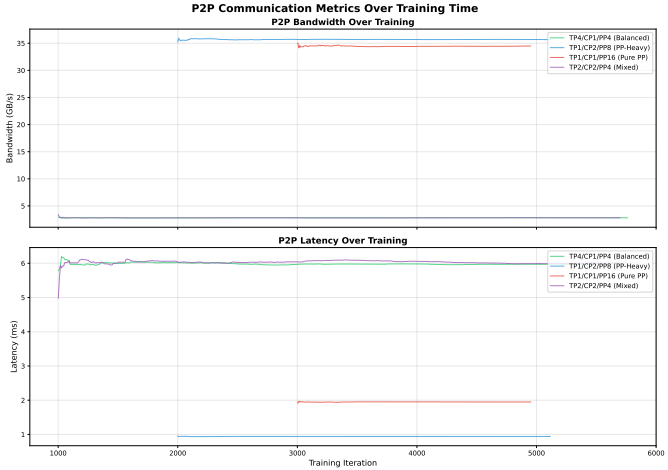
- **High PP** (PP=8, 16): Frequent small transfers → high BW (35 GB/s), low latency (<1ms)
- **Low PP** (PP=2, 4): Fewer, larger transfers → lower BW (2-3 GB/s), higher latency (6ms)

**Trade-off Insight:**

- High PP → more pipeline stages
- But: pipeline bubbles dominate → lower compute efficiency
- Optimal: Balance TP (within-node) + PP (across-node)

# Results: P2P Monitoring Over Time

## Bandwidth & Latency During Training




## Key Findings

- **PP-heavy**: Stable 35 GB/s, ¡1ms latency
- **Balanced**: Lower bandwidth (2-3 GB/s), higher compute utilization
- **Stability**: No degradation over 4000+ iterations

**Implication**: Predictable network behavior enables runtime-adaptive parallelism strategies

# Results: Complete Configuration Comparison

## 9 Valid Configurations (Sorted by Throughput)

| Config | TP | CP | PP | DP | TFLOP/s | Iter (s) | PP BW | PP Lat |
|---|---|---|---|---|---|---|---|---|
| TP4/CP1/PP4 | 4 | 1 | 4 | 1 | 30.0 | 0.89 | 2.8 | 6.0 |
| TP2/CP2/PP4 | 2 | 2 | 4 | 1 | 29.1 | 0.91 | 2.8 | 6.0 |
| TP1/CP4/PP4 | 1 | 4 | 4 | 1 | 29.0 | 0.93 | 2.8 | 5.9 |
| TP1/CP1/PP8 | 1 | 1 | 8 | 2 | 21.2 | 2.43 | 33.5 | 2.0 |
| TP1/CP2/PP8 | 1 | 2 | 8 | 1 | 19.0 | 1.36 | 35.7 | 0.9 |
| TP2/CP1/PP8 | 2 | 1 | 8 | 1 | 18.9 | 1.37 | 35.3 | 1.0 |
| TP4/CP1/PP2 | 4 | 1 | 2 | 2 | 16.7 | 3.07 | 2.8 | 6.1 |
| TP2/CP2/PP2 | 2 | 2 | 2 | 2 | 12.1 | 4.24 | 2.7 | 6.2 |
| TP1/CP1/PP16 | 1 | 1 | 16 | 1 | 11.4 | 2.24 | 34.5 | 1.9 |

*PP BW/Lat = Pipeline parallelism inter-stage communication (GB/s, ms). TP uses NVLink (not monitored).*

**Key Observations:**

- **Best**: TP4/CP1/PP4 balanced within-node TP with cross-node PP
- **Worst**: TP1/CP1/PP16 suffers from pipeline bubble overhead (2.6× slower)
- High PP → frequent small P2P transfers → high BW, low latency (but more bubbles)

# Results: Configuration Analysis

**Key Findings from 9 Experimental Runs**

1. **Best configuration for 16 GPUs**: TP=4, CP=1, PP=4
   - Achieved 30.0 TFLOP/s/GPU throughput
   - Balances tensor parallelism within nodes with pipeline across nodes
2. **Impact of pipeline depth**:
   - PP=4 vs PP=16: 2.6× throughput difference (30.0 vs 11.4 TFLOP/s)
   - Extreme PP causes severe pipeline bubble overhead
3. **Context parallelism (CP) effect**:
   - TP4/CP1/PP4 vs TP2/CP2/PP4: Similar throughput (∼29-30 TFLOP/s)
   - CP provides memory savings without major throughput penalty
4. **Network topology insight**:
   - Intra-node: Prioritize TP (high NVLink bandwidth)
   - Inter-node: Moderate PP acceptable; avoid extreme PP

**General Guideline**: For Perlmutter (4 GPUs/node), use **TP=4 within node, PP across nodes**

# GPU Usage: Resource Allocation

**Detailed GPU Hours Breakdown**

| Activity | Configuration | Duration | GPUs | GPU-Hours |
|---|---|---|---|---|
| Environment Setup | Various | 4 hrs | 4 | 16 |
| Grid Search Runs | 9 configs, 16 GPUs | 1.5 hrs each | 16 | 216 |
| Debug & Validation | Various | 8 hrs | 16 | 128 |
| **TOTAL** | | | | **$\sim$360 GPU-hours** |

**Resource Justification**

- 9 parallelism configurations tested systematically
- Each run: 1.5-hour allocation with P2P monitoring enabled
- Initial profiling adds 30-60 seconds per job for topology discovery
- All runs on NERSC Perlmutter (4 nodes $\times$ 4 A100 GPUs)

# GPU Usage: Utilization Analysis

## Throughput per Configuration

| Config | TFLOP/s | Iter Time |
|---|---|---|
| TP4/CP1/PP4 | 30.0 | 890 ms |
| TP2/CP2/PP4 | 29.1 | 914 ms |
| TP1/CP4/PP4 | 29.0 | 927 ms |
| TP1/CP2/PP8 | 19.0 | 1365 ms |
| TP1/CP1/PP16 | 11.4 | 2240 ms |

## Performance Bottlenecks

- **PP=16**: Pipeline bubbles dominate (2.5× slower)
- **PP=8**: Moderate bubble overhead
- **TP=4, PP=4**: Best balance

## P2P Monitoring Overhead

- Sample rate: 10% of P2P operations
- Measured overhead: $<0.5\%$
- Async CUDA events: Non-blocking

## PP Communication Stats

| Metric | Value |
|---|---|
| Total PP P2P calls monitored | 27,000+ |
| PP BW (low PP=2,4) | 2.8 GB/s |
| PP BW (high PP=8,16) | 35 GB/s |
| PP latency (low PP) | 6.0 ms |
| PP latency (high PP) | 0.9 ms |

**PP = pipeline stage transfers only*

# Lessons Learned

**Key Insights from 9 Experimental Runs**

- **Extreme parallelism hurts throughput** → PP=16 achieved only 11.4 TFLOP/s vs 30.0 for PP=4 (2.6× slower)
- **Pipeline bubbles dominate at high PP** → More stages = more synchronization overhead; diminishing returns beyond PP=4
- **Context Parallelism is "free"** → TP2/CP2/PP4 matched TP4/CP1/PP4 throughput while reducing memory pressure
- **Profiling overhead is negligible** → 10% sampling + async events = <0.5% overhead; enables runtime monitoring
- **Bandwidth vs efficiency trade-off** → PP-heavy configs show 35 GB/s bandwidth but poor compute utilization

**Takeaway**: *Balanced configs (TP=4, PP=4) outperform extremes; profiling reveals hidden bottlenecks.*

# Project Completion: What We Delivered

**Deliverables**

- **Implementation**: Megatron-LM + LLaMA 8B, P2P profiling, multi-config scripts
- **Experiments**: Baseline + multi-GPU runs, bandwidth profiling, config comparison
- **Analysis**: Optimal config, comms bottlenecks, scaling efficiency
- **Documentation**: Open-source code + setup + reproducible training scripts

# Project Completion: What We Delivered

**Deliverables**

- **Implementation**: Megatron-LM + LLaMA 8B, P2P profiling, multi-config scripts
- **Experiments**: Baseline + multi-GPU runs, bandwidth profiling, config comparison
- **Analysis**: Optimal config, comms bottlenecks, scaling efficiency
- **Documentation**: Open-source code + setup + reproducible training scripts

**Alignment with Original Goals**

| Goal | Planned | PoC |
|---|---|---|
| Parallelism framework | BLIP | LLaMA |
| P2P profiling | ✓ | ✓ |
| Config optimization | ✓ | ✓ |
| Dynamic adaptation | ✓ | Future |
| Multi-node scaling | ✓ | ✓ |

# Project Completion: What We Delivered

## Deliverables

- **Implementation**: Megatron-LM + LLaMA 8B, P2P profiling, multi-config scripts
- **Experiments**: Baseline + multi-GPU runs, bandwidth profiling, config comparison
- **Analysis**: Optimal config, comms bottlenecks, scaling efficiency
- **Documentation**: Open-source code + setup + reproducible training scripts

## Alignment with Original Goals

| Goal | Planned | PoC |
|---|---|---|
| Parallelism framework | BLIP | LLaMA |
| P2P profiling | ✓ | ✓ |
| Config optimization | ✓ | ✓ |
| Dynamic adaptation | ✓ | Future |
| Multi-node scaling | ✓ | ✓ |

**Summary**

- Core goals achieved
- Model shift: BLIP → LLaMA (integration-driven)
- PoC forms base for adaptive system

# Limitations & Future Directions

**Limitations**

- **Model scope** — LLaMA only, single size (8B)
- **Limited scale** — up to 16 GPUs / 4 nodes
- **Static configs** — fixed at training start
- **Dataset size** — WikiText-103 only

# Limitations & Future Directions

**Limitations**

- **Model scope** — LLaMA only, single size (8B)
- **Limited scale** — up to 16 GPUs / 4 nodes
- **Static configs** — fixed at training start
- **Dataset size** — WikiText-103 only

**Future Directions**

- Extend to multimodal (BLIP) and multiple model scales (7B–70B)
- Scale deployment to 32–64+ GPUs across large clusters
- Implement **dynamic parallelism switching** during runtime
- Train on large corpora (COCO, RedPajama, The Pile)

**Goal**: *Move from a static single-model PoC to an adaptive, dynamic multimodal system.*

# Future Work: From PoC to Adaptive System

**Next Phase: Turn our static PoC into a data-driven, multimodal adaptive framework**

- **1. Multimodal + model scaling** Integrate BLIP within our Megatron-based pipeline, characterize cross-modal latency patterns, and scale model sizes (7B–70B).
- **2. Large-cluster deployment** Port our training scripts to 32–64+ GPU environments; benchmark scaling limits and train on multi-billion-token corpora (C4, RedPajama).
- **3. Runtime adaptive control** Use our existing P2P hooks to trigger online bandwidth monitoring and dynamically switch TP/CP/PP/DP depending on network load.
- **4. Cost model + auto-configuration** Formalize trade-offs (memory, comms, bubbles) into a predictive cost model that automatically recommends optimal configurations.

**Research objective:** *A scalable, multimodal system that adapts its parallelism strategy in real time.*

# Contributions & Impact

**Research Contributions**

- **P2P profiling infrastructure** NCCL hooks for Megatron-LM; reusable beyond LLaMA
- **Configuration characterization** Systematic TP/CP/PP/DP comparison + cluster-aware guidelines
- **LLaMA 8B benchmarks** Throughput + scaling efficiency as a baseline
- **Foundation for adaptive systems** PoC shows feasibility and reveals key bottlenecks

**Practical Impact**

# Contributions & Impact

## Research Contributions

- **P2P profiling infrastructure** NCCL hooks for Megatron-LM; reusable beyond LLaMA
- **Configuration characterization** Systematic TP/CP/PP/DP comparison + cluster-aware guidelines
- **LLaMA 8B benchmarks** Throughput + scaling efficiency as a baseline
- **Foundation for adaptive systems** PoC shows feasibility and reveals key bottlenecks

## Practical Impact

- **For practitioners** Config selection guide, open-source implementation, profiling tools
- **For researchers** Baseline for adaptive work + communication profiling data

## Open Source

- Code: GitHub Repository
- Documentation: README, examples, setup
- Training scripts: Multi-GPU ready

## Toward Adaptive Multimodal Training

- **Initial objective**: Adaptive parallelism for multimodal transformers
- **Pragmatic PoC**: LLaMA 8B used to examine constraints and scalability
- **Key outcome**: Characterization of communication bottlenecks and optimal TP/CP/PP/DP configurations
- **Research value**: Infrastructure and benchmarks enabling adaptive, large-scale multimodal training

---

**Contact:** Taqiya Ehsan — te137@rutgers.edu; Khizar Anjum — ma1629@rutgers.edu