# Context Parallelism for Scalable Million-Token Inference

Presented by: [Your Name]

[Your Affiliation]

Distributed Machine Learning Systems

*Yang et al., arXiv:2411.01783v3, November 2024*

# Long-Context Inference Challenges

01

## Memory Bottleneck

KV cache grows linearly with sequence length, consuming massive GPU memory

02

## Computational Complexity

Quadratic attention computation—$O(n^2)$ operations for sequence length n

03

## Communication Overhead

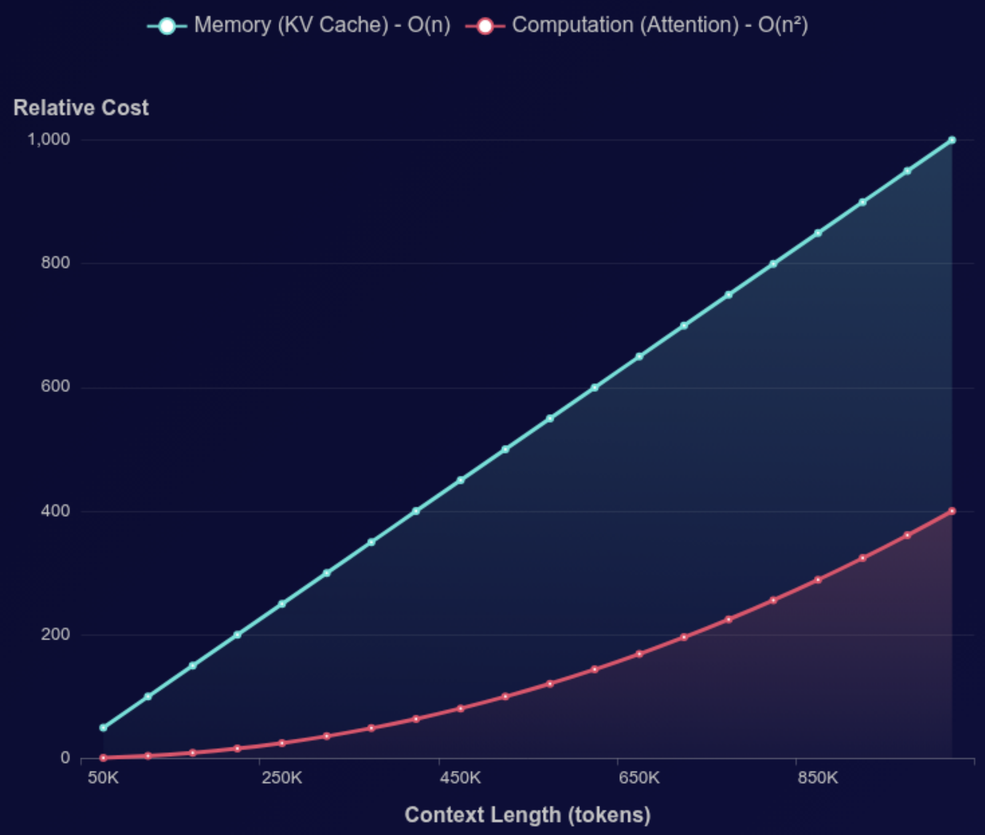
Distributed systems require extensive data movement between devices

04

## Scalability Limitation

Existing parallelism (tensor/pipeline/sequence) struggles with million-token contexts

## Memory & Compute Scaling



# Motivation: Why Context Parallelism?

**LIMITATION**

Existing parallelism techniques (tensor, pipeline, sequence) designed for model/data parallelism, not sequence length scaling

**DEMAND**

Long-context applications require million-token support (128K-1M+ tokens) for real-world use cases

**REQUIREMENT**

Near-linear scaling to 128+ GPUs needed for efficient distributed inference at scale

## Retrieval-Augmented Generation (RAG)

Integrate external knowledge bases with LLMs, requiring processing of extensive retrieved documents alongside queries for accurate, context-aware responses

## Long Document Analysis

Process entire legal contracts, research papers, technical manuals, and reports (100K+ tokens) without truncation or chunking for comprehensive understanding

## Code Understanding & Extended Reasoning

Analyze entire codebases, multi-file projects, and complex reasoning chains requiring long-range dependencies across hundreds of thousands of tokens

# Core Concept: Partitioning Context Across Devices

1

## Context Partitioning

Input context sequence divided across N GPUs—each device handles a distinct context chunk

2

## Local Query Processing

Query tokens processed locally on each GPU using its assigned context partition

3

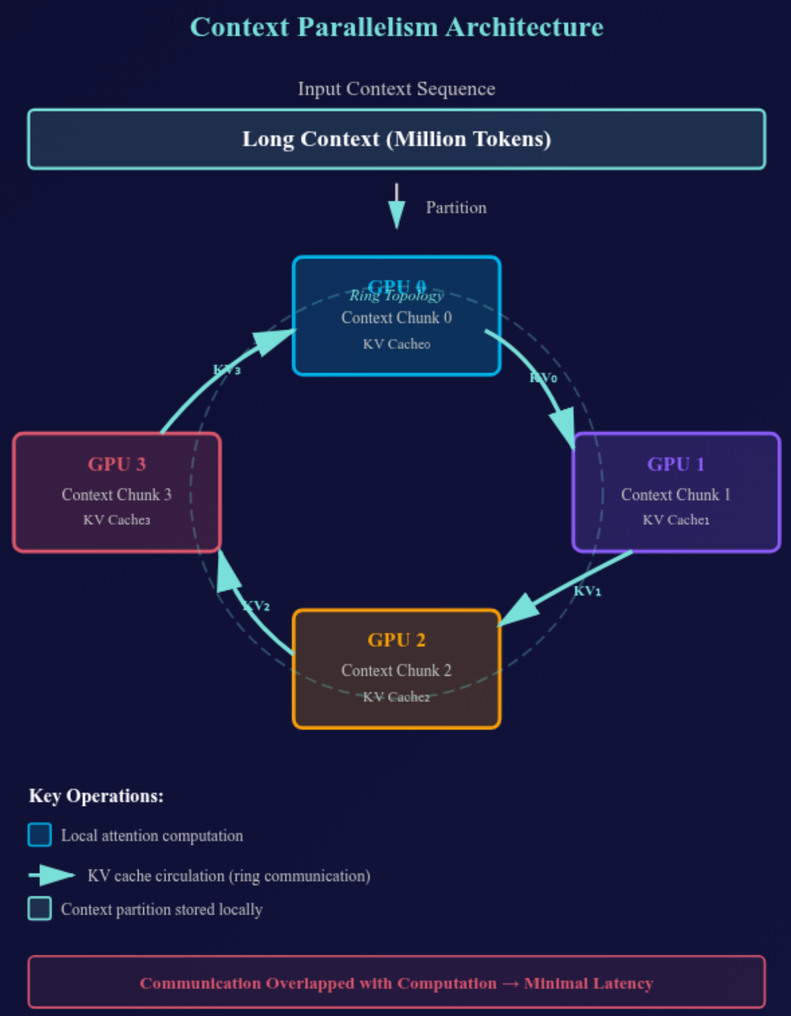
## Ring Attention Mechanism

KV cache circulates in ring topology for cross-device communication—each GPU receives KV from neighbors

4

## Computation-Communication Overlap

Key insight: Communication fully overlapped with attention computation—minimizes latency overhead



# Ring Attention & Communication Optimization

## Pass-KV Strategy

Circulates KV cache in ring topology across N ranks. Each rank executes N partial attention computations. Communication fully overlapped with attention when Q and KV have same length.

**Optimal: Q = KV length (training, low miss rate)**

## Pass-Q Strategy

Alternative approach using All2All communication for different Q/KV ratios. Selected when KV cache miss rate < 3.25%, where pass-KV communication becomes exposed in critical path.

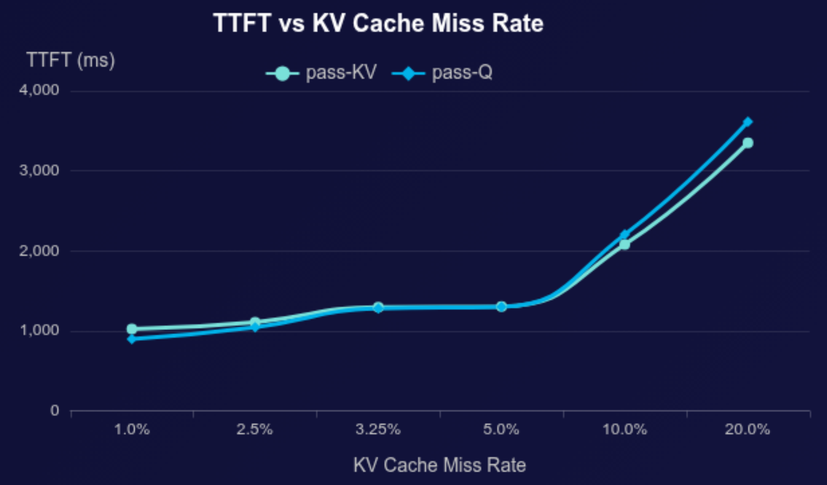
**Optimal: High KV hit rate (< 3.25% miss)**

## Communication-Computation Overlap

Ring-based pass-KV reduces computation granularity and facilitates overlap. SendRecv for $KV_{j-1}$ overlapped with attention compute between $Q_k$ and $KV_j$.

## TTFT Latency Relationship

Time-to-first-token (TTFT) latency is **linearly proportional to KV cache miss rate**. At 2.5% miss rate, pass-Q outperforms pass-KV due to exposed communication in ring loop.



# Parallelism Approaches: Comparative Analysis

## Tensor Parallel

Splits model weights across devices

## Pipeline Parallel

Splits layers across devices

## Sequence Parallel

Splits sequence within attention head

## Context Parallel

Partitions full context in ring topology

| Metric | Tensor | Pipeline | Sequence | Context |
|---|---|---|---|---|
| Scalability | Sublinear | Limited by depth | Moderate | Near-linear |
| Communication Pattern | AllReduce | Point-to-point | AllReduce | Ring (SendRecv) |
| Context Length Support | Limited | Limited | Moderate (128K) | Million-token |

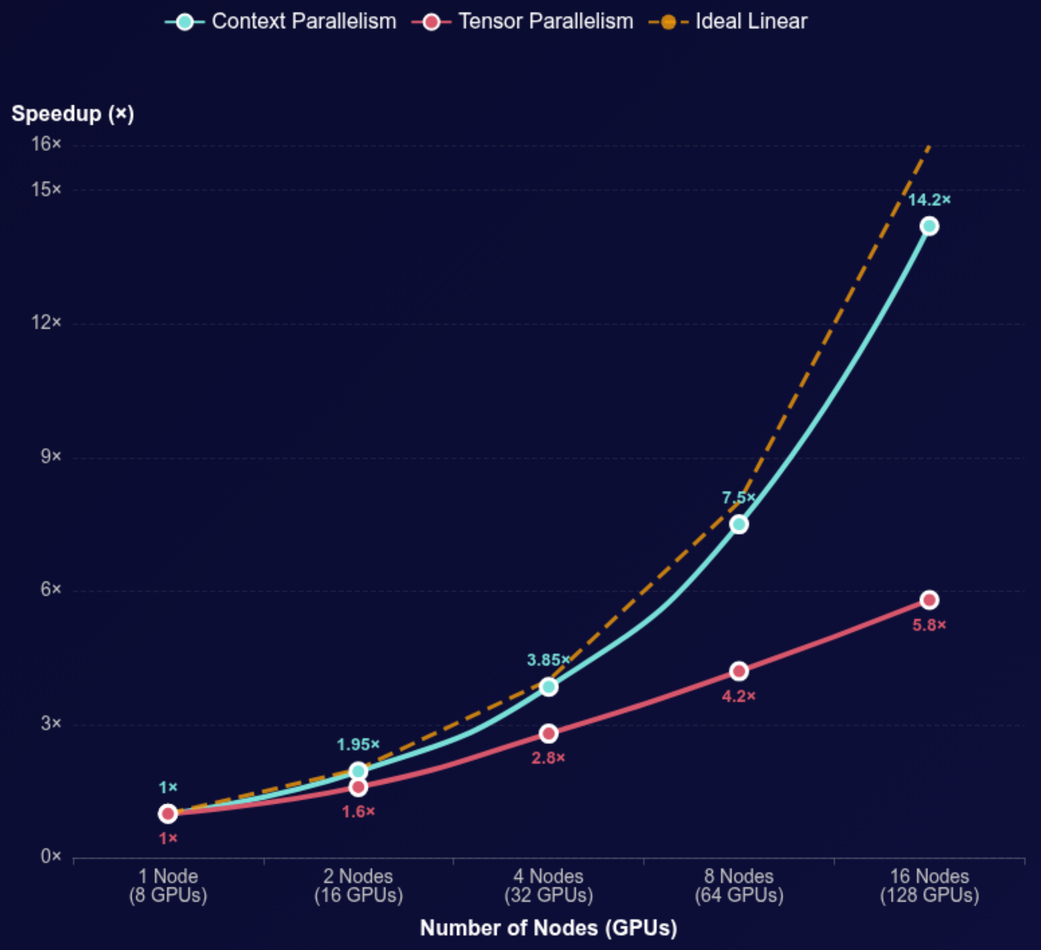
# Why Context Parallelism Scales Better

1. **Near-linear scaling** achieved with up to **128 GPUs** across 16 nodes
2. **Ring communication pattern** has lower bandwidth requirements than all-reduce operations
3. **Computation-communication overlap** reduces effective latency—communication fully overlapped with attention
4. **No pipeline bubbles** unlike pipeline parallelism—continuous computation without idle time
5. **Orthogonal to tensor/pipeline parallelism**—can be combined (e.g., CP4+TP8) for optimal performance

**KEY INSIGHT**

**CP communicates token embeddings on attention layers while TP communicates on linear layers—less communication traffic overall**

## Scaling Performance: CP vs TP



# Implementation Details & Experimental Setup

## Implementation Framework

- **Ring Attention Variants:** Pass-KV (circulates KV cache) and Pass-Q (circulates queries) with runtime heuristic selection
- **Adaptive Strategy:** Pass-Q optimal when KV cache miss rate <5%, Pass-KV preferred when miss rate >5%
- **Scenarios Supported:** Full prefill, partial prefill, and decode stages

## Metrics Tracked

**TTFT Latency**
Time-to-first-token, linearly proportional to KV cache miss rate

**Throughput**
FLOPS utilization (502 TF/sec per H100 at 1M context)

**KV Cache Hit Rate**
Persistent cache efficiency in multi-turn scenarios

**Scaling Behavior**
Latency reduction with increasing CP ranks

## Experimental Configuration

| Parameter | Configuration |
|---|---|
| **Model** | Llama3 405B (8 KV heads, 128K max context) |
| **Context Length** | 2K to 1M tokens (primary: 128K) |
| **GPU Hardware** | NVIDIA H100, 400 Gb/s RDMA backend network |
| **Node Configuration** | 4-16 nodes (GTT infrastructure) |
| **CP Ranks** | CP2, CP4, CP8 (2-8 context parallel ranks) |
| **Batch Size** | Max batch size 1 (full prefill tests) |
| **Baseline Comparisons** | **TP8/TP16** (Tensor Parallelism), **Standard Inference** |

# Key Results: Scaling & Performance

## ① SCALING

### Near-Linear Scaling

Context parallelism achieves **near-linear scaling** up to **128 GPUs** (16 nodes × 8 GPUs), significantly outperforming tensor parallelism which becomes bottlenecked by inter-host communication

## ② LATENCY

### TTFT Proportionality

TTFT latency is **linearly proportional** to KV cache miss rate—from **1.0s at 1%** miss rate to **3.4s at 20%** miss rate for 128K context

## ③ STRATEGY

### Pass-KV vs Pass-Q

**Pass-KV optimal** for equal Q/KV lengths; **tipping point at 5%** cache miss rate (T=6400). Pass-Q preferred when miss rate exceeds 3.25%

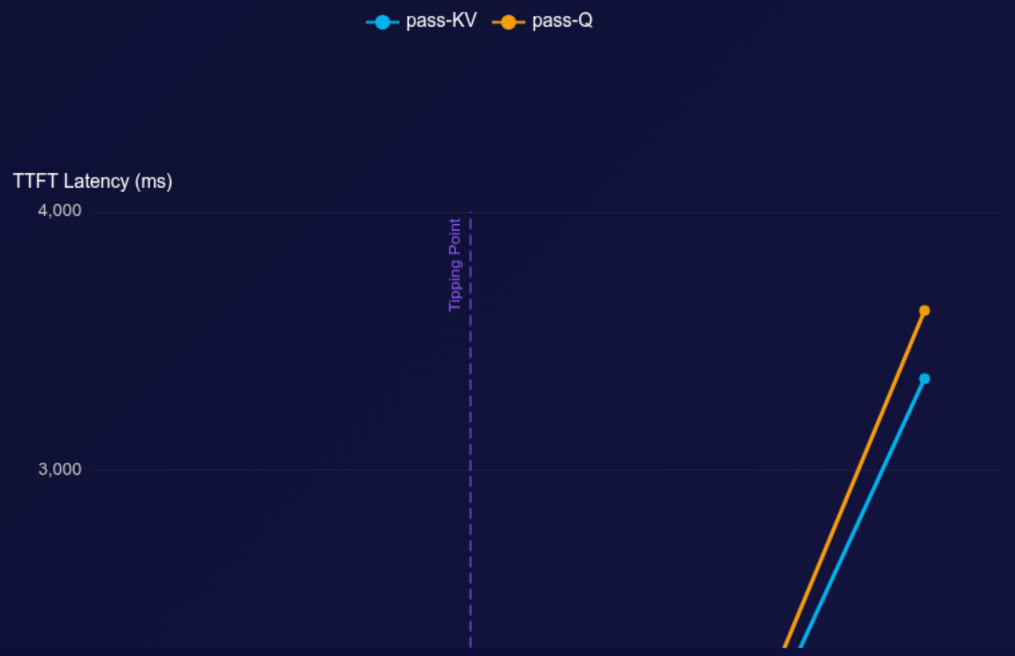
## ④ THROUGHPUT

### Performance Gains

Achieves **540 TF/sec per H100** for 8K context, scaling to **1M tokens over 128 GPUs**. CP reduces communication traffic compared to TP

### Scaling Efficiency: CP vs TP


### TTFT vs KV Cache Miss Rate



# Key Contributions & Future Directions

## Main Contributions

### ① Novel Technique

**Context parallelism** enables **million-token inference** by partitioning context across devices with ring attention mechanism

### ② Ring Attention

**Communication-computation overlap** with pass-KV and pass-Q variants optimized for different context/KV ratios

### ③ Near-Linear Scaling

Demonstrated on **up to 128 GPUs** with significantly reduced latency and improved scalability

### ④ Practical Performance

**2-4× speedup** on long-context workloads with efficient KV cache management across distributed systems

## Future Research Directions

- **Hybrid Parallelism:** Combining context parallelism with tensor and pipeline parallelism for optimal resource utilization
- **Multi-Modal Contexts:** Extending to handle mixed text, image, and video data in long-context scenarios
- **Heterogeneous Hardware:** Optimizing for diverse GPU architectures and network topologies in production systems

### KEY TAKEAWAY

**Context parallelism fills a critical gap in distributed inference for long-context LLMs, enabling scalable million-token processing**