# Optimizing parallelization and overlap to increase training efficiency using Megatron-Core

Xue Huang, Solution Architect, NVIDIA
xueh@nvidia.com

Yuliang Liu, Foundation Model Training Director, Kuaishou Technology
liuyuliang@kuaishou.com

# Agenda

- Overview
- Backgroud
- NVIDIA Megatron Core
- Hotspots Evolution, Solutions, and Achievements in KuaiShou LLM Training
- Summary and Future work

# OVERVIEW

快手

| Evolution of LLM | Hotspots in LLM training | Solutions | Achievements |
| --- | --- | --- | --- |
| more GPUs; GPU upgrade | Data Parallel time increase | Overlapping in Data Parallel | 15% acceleration |
| model size grows; sequence length increase | Tensor Parallel time increase | Overlapping in Tensor Parallel | 5-10% acceleration Depend on model size |
| sequence length increase to 1M | Tensor Parallel needs to cross nodes to meet the increasing memory demands | • Context parallel<br>• GEMM-last Recomputing<br>• Pipeline-aware offloading | 30% acceleration |

GPT 175B with 4096 sequence length achieved **up to 40%** acceleration on 2K Hopper cluster.

# Background

# Large Language Model

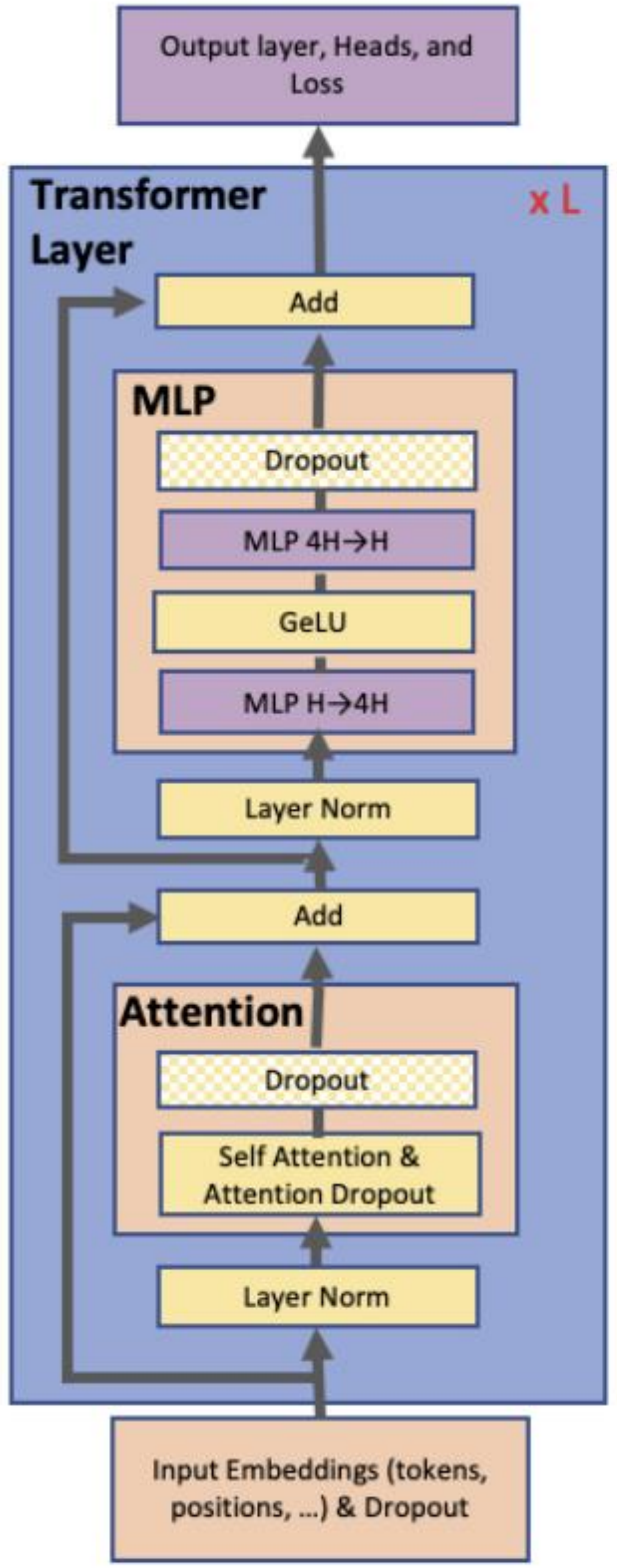
快手

- Transformer-based Large Language Models
  - encoder-only architecture: Bert
  - **decoder-only architecture: GPT/LLaMa**
  - encoder-decoder architecture: T5

- Decoder-only architecture:
  - Input Embedding
  - Decoder layers
    - Attention: Multi-head attention for GPT, Grouped-Query Attention for Llama
    - MLP
    - LayerNorm
    - Residual connection
  - Output layer

- GPT 175B Model configuration:
  - Number of layers: 96
  - Sequence length: 2048
  - Hidden size: 12288
  - Vocabulary size: 51200
  - Total parameters: 175B

GPT




NVIDIA

# Challenges

## GPT 175B

- Memory Challenge: (BF16+FP32)
  - The model could not fit in the memory of a single GPU or a sing node
  - **Model parallelism is the MUST have and it MUST across multi-nodes**

| Parameters (2 bytes) | Gradients (4 bytes) | Master Parameters (4 bytes) | Adam Optimizer state (4 bytes * 2) | Total |
|---|---|---|---|---|
| 350 GB | 700 GB | 700 GB | 1400 GB | **3.15 TB** |

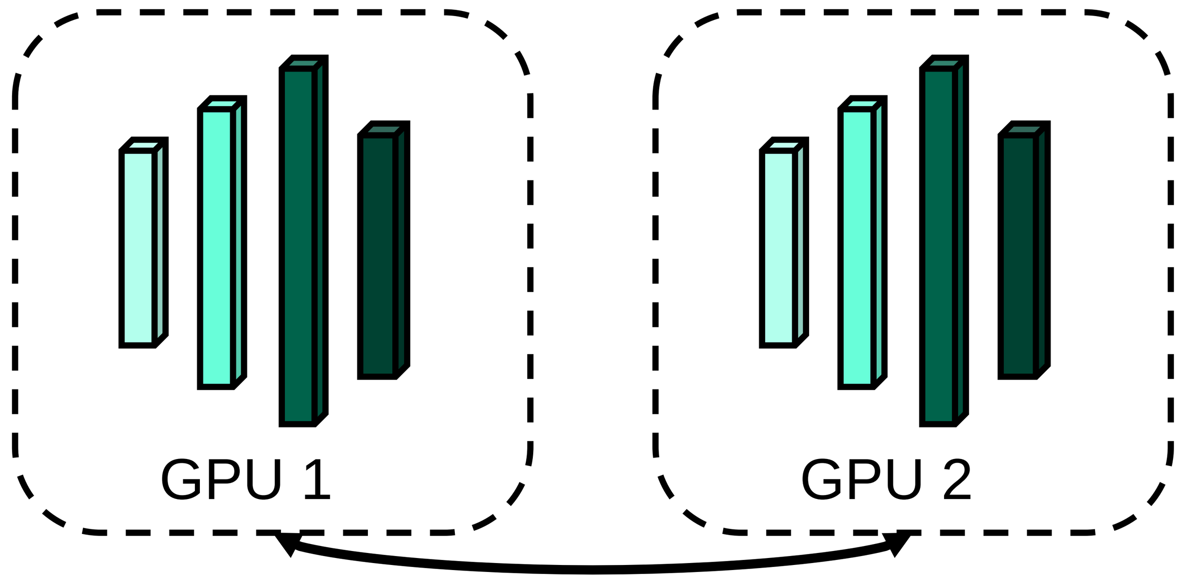
- Compute Challenge:
  - model FLOPs of each iteration: $72BLsh^2\left(1+\frac{s}{6h}+\frac{V}{12hL}\right)$
    - Please refer to https://arxiv.org/abs/2205.05198
    - B: batch size, S: sequence length, l: num of transformer layers, h: hidden size, V: vocabulary size
  - Each iteration: 4.5 ExaFLOPs (if B=2048)
  - Total 300B tokens(75K iterations): 340 ZettaFLOPS
  - **Extremely huge computing requirements: ~35 years with single A100 computing (Not considering efficiency).**
- Communicate Challenge:
  - Model parallelism and data parallelism will introduce communication across GPUs.
  - **As the model size expands and the GPU number increases, communication will become the bottleneck**

**Need an efficient implementation framework to achieve high training efficiency**

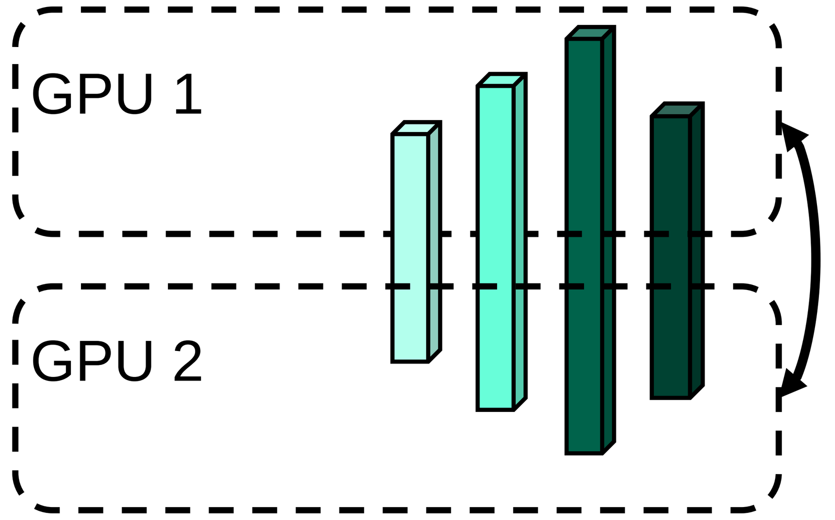
https://arxiv.org/abs/2205.05198

# Parallel Training

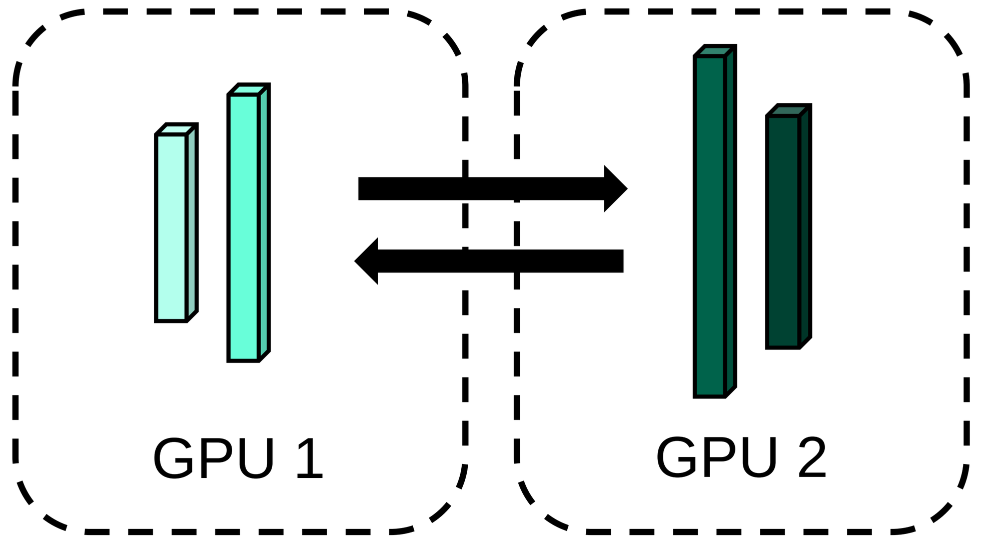
**Data Parallelism**




**Model Parallelism**




Tensor(intra-layer) parallelism

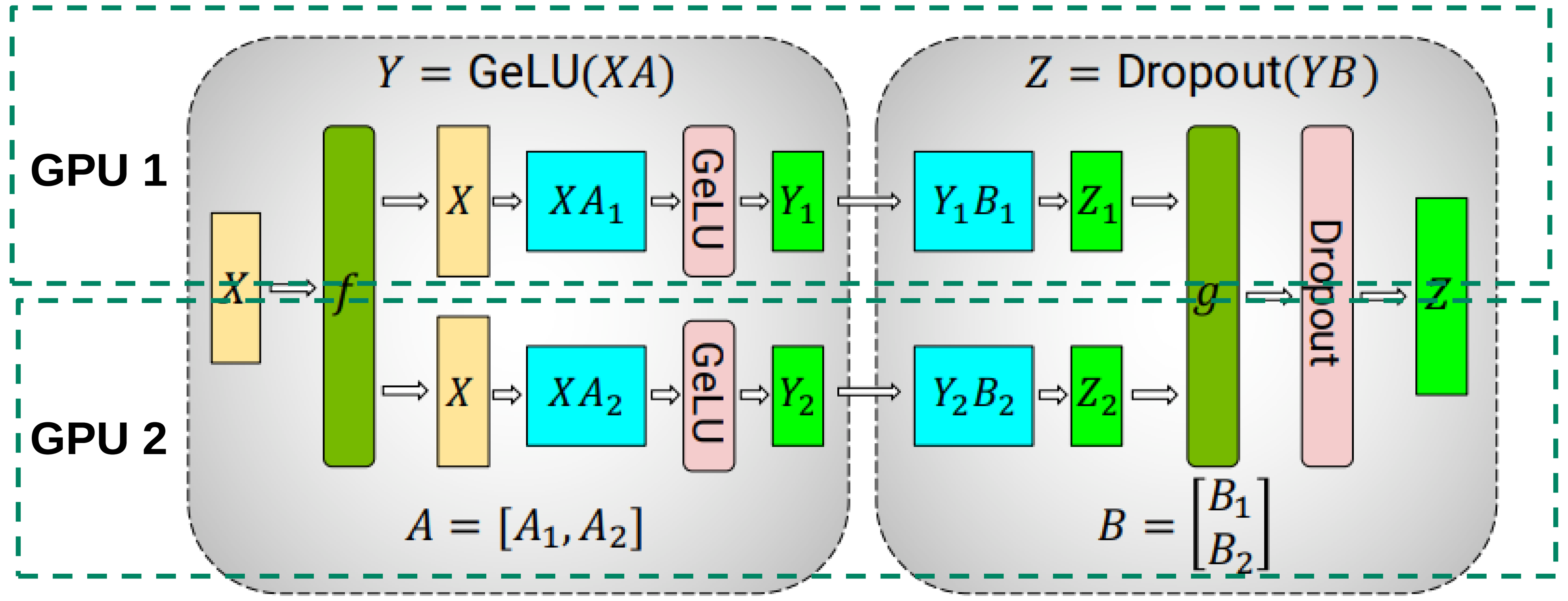



Pipeline(inter-layer) parallelism

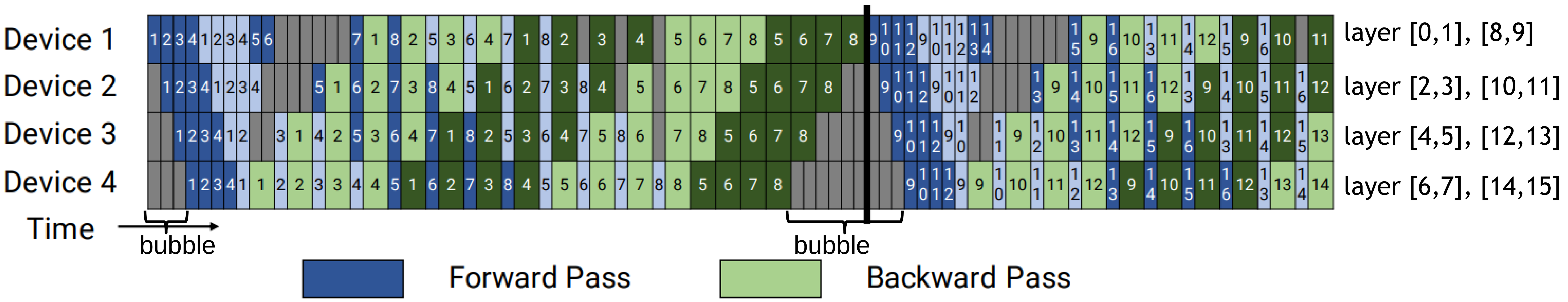
- Data Parallelism:
  - Each device has a copy of the model parameters
  - Split input data across multiple GPUs and allreduce of weight gradients after every iteration.
- Model Parallelism:
  - Split the model parameters across multiple GPUs
  - Tensor Parallelism: Split individual layers across multiple GPUs. Devices compute different parts of Layers 0,1,2,3
  - Pipeline Parallelism: Split layers across multiple GPUs. Layers 0,1 and layers 2,3 are on different GPUs

# Model Parallelism

- **Tensor Parallelism**
  - Splits all GEMMs and self-attention blocks in each transformer layer across GPUs
  - Requires 4 all-reduces in a single forward and backward pass of each layer.
  - **Communication intensive, better only intra-node**
- **Pipeline Parallelism**
  - Split layers across multiple GPUs. Pipeline parallelism will introduce the pipeline bubble
  - **Interleaved Pipeline:**
    - Each GPU performs computation for multiple subsets of layers (called a model chunk) instead of a single contiguous set of layers. Layers 0,1,4,5 and layers 2,3,6,7 are on different GPUs
    - Reduce the bubble time fraction by v (the number of model chunks)
  - Need point-to-point communication between stages, which is much cheaper than all-reduce communication. We can use it inter-nodes.




tensor parallelism in MLP

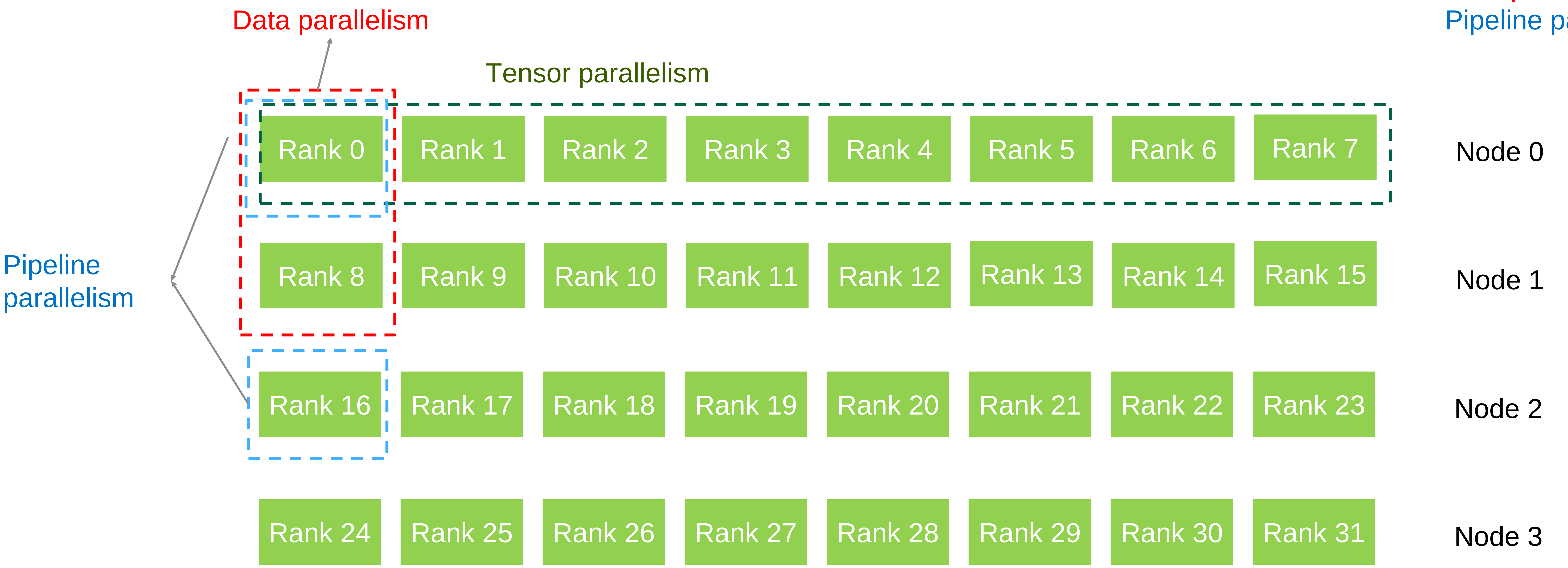



interleaved pipeline

# 3D Parallelism

Data parallelism + Tensor parallelism + Pipeline parallelism

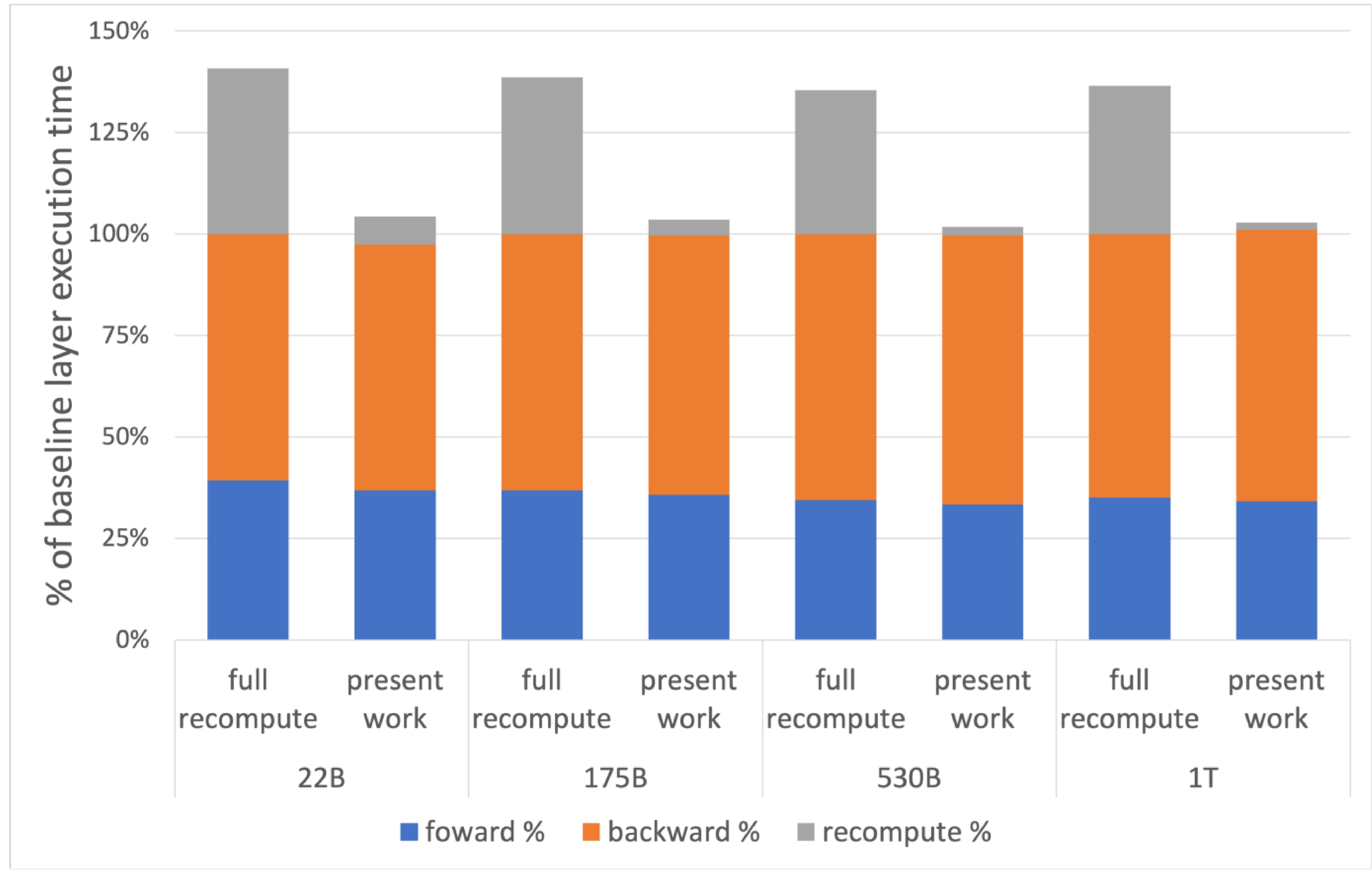
Tensor parallel size 8
Data parallel size: 2
Pipeline parallel size: 2




- 4 Tensor Parallelism group: [0,1,2,3,4,5,6,7], [8,9,10,11,12,13,14,15], [16,17,18,19,20,21,22,23],[24,25,26,27,28,29,30,31]
- 16 Data Parallelism group: [0,8], [1,9], [2,10], [3,11], [4,12], [5,13], [6,14], [7,15], [16,24], [17,25], [18,26], [19,27], [20,28], [21,29], [22,30], [23,31]
- 16 Pipeline Parallelism group: [0,16],[1,17],[2,18],[3,19],[4,20],[5,21],[6,22],[7,23],[8,24],[9,25],[10,26],[11,27],[12,28],[13,29],[14,30],[15,31], [16,32]

# Memory Saving

- **Sequence parallelism**
  - Partition the layernorm and dropout along the sequence dimension to reduce the activation memory
  - 4 reduce-scatter + 4 allgather( ~ 4 allreduce)  in a single forward and backward, **no extra communication overhead**
- **Activation Recomputation**
  - Recompute activations during the backward pass instead of saving them in memory.
  - Full activation recomputation:
    - Recompute all activations in transformer layers
    - Significantly reduce the required memory, but with **30%-40%** computing overhead.
  - Selective activation recomputation:
    - Choose activations to recompute based on compute-memory tradeoff
    - Lower memory footprint of activations and slightly recompute overhead



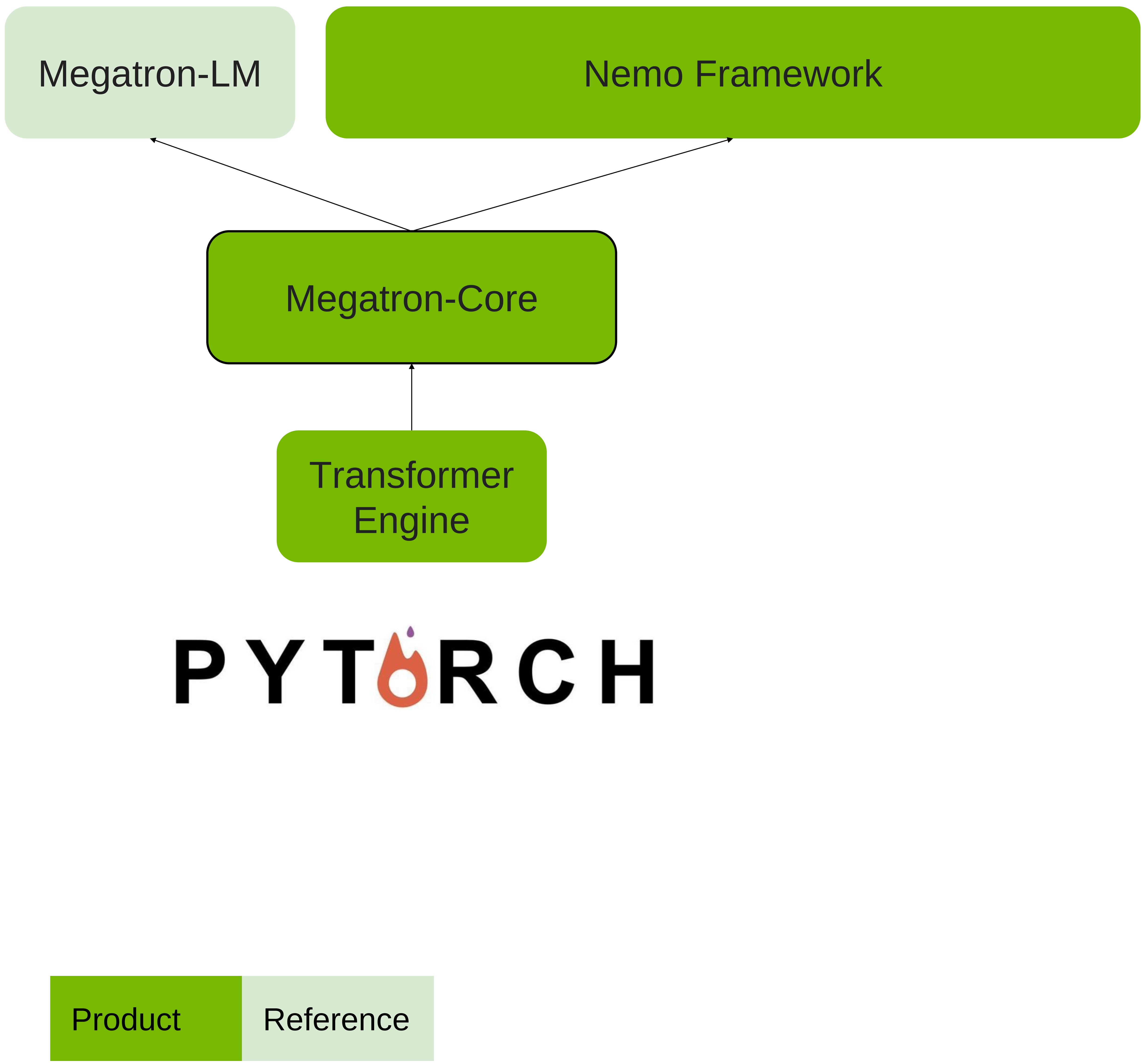

- **Distributed Optimizer/ZeRO**
  - Share the model states(model parameters, gradients, optimizer states) across data parallel GPUs
  - Zero2 and Zero3 may introduce more communication
- **Offloading**
  - Reduce GPU memory requirement by exploiting CPU memory but with GPU-CPU-GPU transfer overhead
    - Offload the gradients, parameters and optimizer states to the CPU
    - Offload activations from GPU to CPU after forward pass and prefetch back from CPU to GPU before backward pass

| | Parameters (2 bytes) | Gradients (4 bytes) | Master Parameters (4 bytes) | Adam Optimizer state (4 bytes * 2) | Total |
|---|---|---|---|---|---|
| baseline | 2M | 4M | 4M | 8M | **18M** |
| Zero1 | 2M | 4M | 4M / dp | 8M / dp | **(6+12/dp)*M** |
| Zero2 | 2M | 4M / dp | 4M / dp | 8M / dp | **(2+16/dp)*M** |
| Zero3 | 2M / dp | 4M / dp | 4M / dp | 8M / dp | **(18/dp)*M** |

# LLM Training – NVIDIA Megatron Core

# Overview of NVIDIA’s Large Language Model offerings

快手




## Core value Proposition

**Nemo Framework:** **Easy to use OOTB FW with large model collections** for Enterprise users to experiment, train, and deploy.

**Megatron-Core:** **Library for GPU optimized techniques for LLM training.** For customers to build custom LLM framework.

**Megatron-LM:** A **lightweight framework reference** for using Megatron-Core to build your own LLM framework.

**Transformer Engine:** **Hopper accelerated Transformer models.** Specific acceleration library, including FP8 training.

megatron core: https://github.com/NVIDIA/Megatron-LM/tree/main/megatron/core

# Megatron-Core

快手

## Performance at Scale

**Memory, Compute, and Communication Optimization**

**Parallelism Techniques**

- Data - Parallelism
- Tensor - Parallelism
- Pipeline - Parallelism
- Sequence - Parallelism
- Expert - Parallelism
- Context - Parallelism

**Memory Saving Techniques**

- Selective Activation Recompute
- CPU offloading (Activation, Weights)
- Attention: FA1, FA2, FA-cuDNN, GQA, MQA, SWA

**Distributed Optimizers (WIP)**

- Zero-1, Zero-2, Zero-3
- Precision aware optimizers (BF16, FP8)

Hopper FP8 via Transformer Engine
Communication Overlap Optimizations
MLPerf Optimizations
TRT-LLM based Inference

## Flexibility

Optimized transformer blocks and techniques for LLM frameworks

- Architecture
  - Decoder only (GPT/Llama)
  - Enc-Dec (T5)
  - Encoder (BERT)
  - RAG (RERTO)
  - MoE
  - ViT
- Dataloaders for different architecture
- Distributed Checkpointing
- Spec options for customizing Transformer layer
- PyT programmability interface

## Formalized Product Support

Latest research and performance optimizations

Regular release, open-source on GitHub and pip wheels

Versioned APIs and Documentation

Open source - welcome PRs from the community


NVIDIA.

# Hotspots Evolution, Solutions and Achievements in KuaiShou LLM Training

快手



NVIDIA.

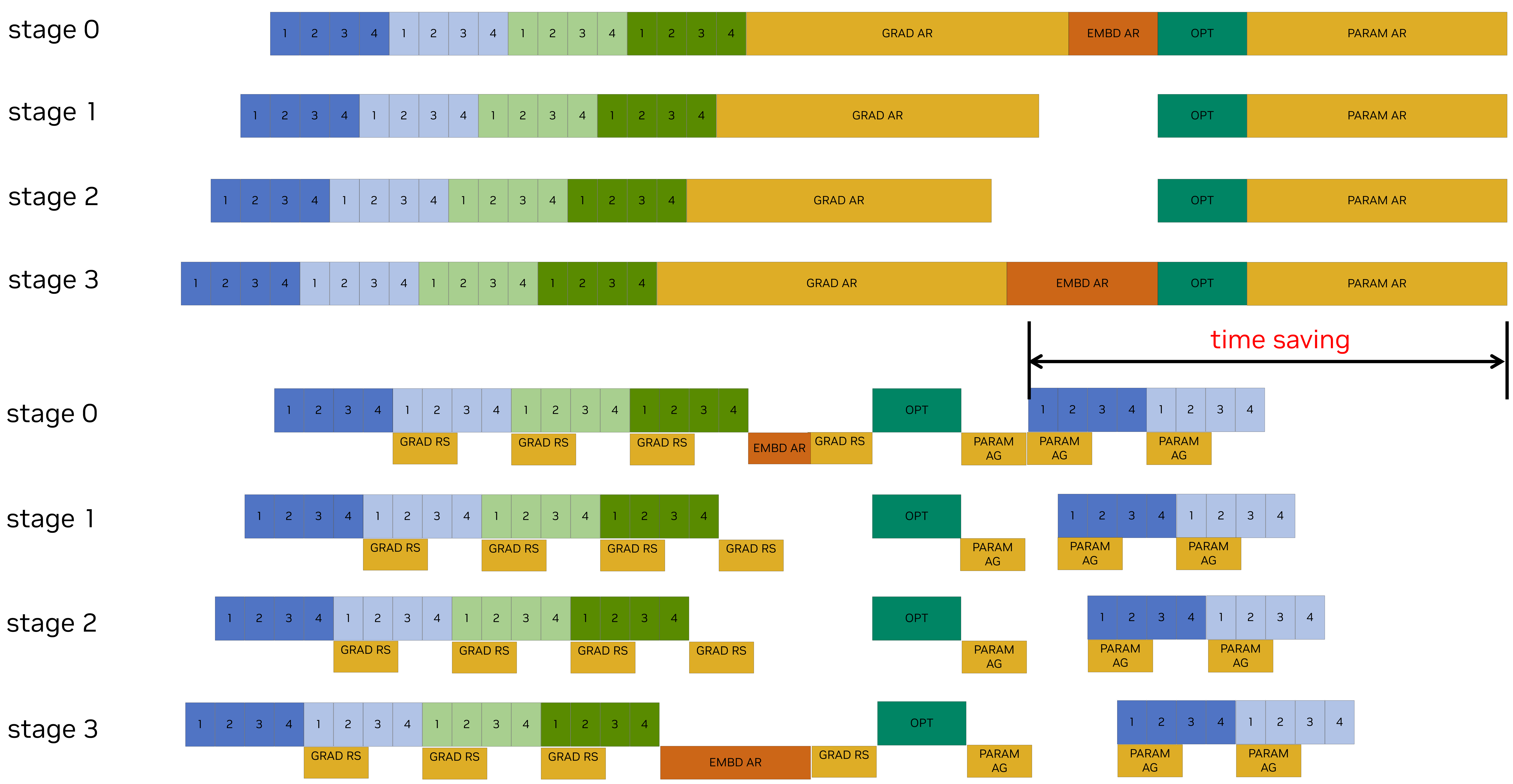
# Overlap optimization - Data Parallel

## challenges

- In the training process of Large Language Models (LLMs), the problem size is generally fixed
- In actual training, we have observed that as the cluster scale expands and computing devices are enhanced (from Ampere to Hopper), the communication-to-computation ratio increases, with the proportion of DP (Data Parallel) communication time exceeding **20%**.

# Overlap optimization - Data Parallel

## Solution




快手

NVIDIA

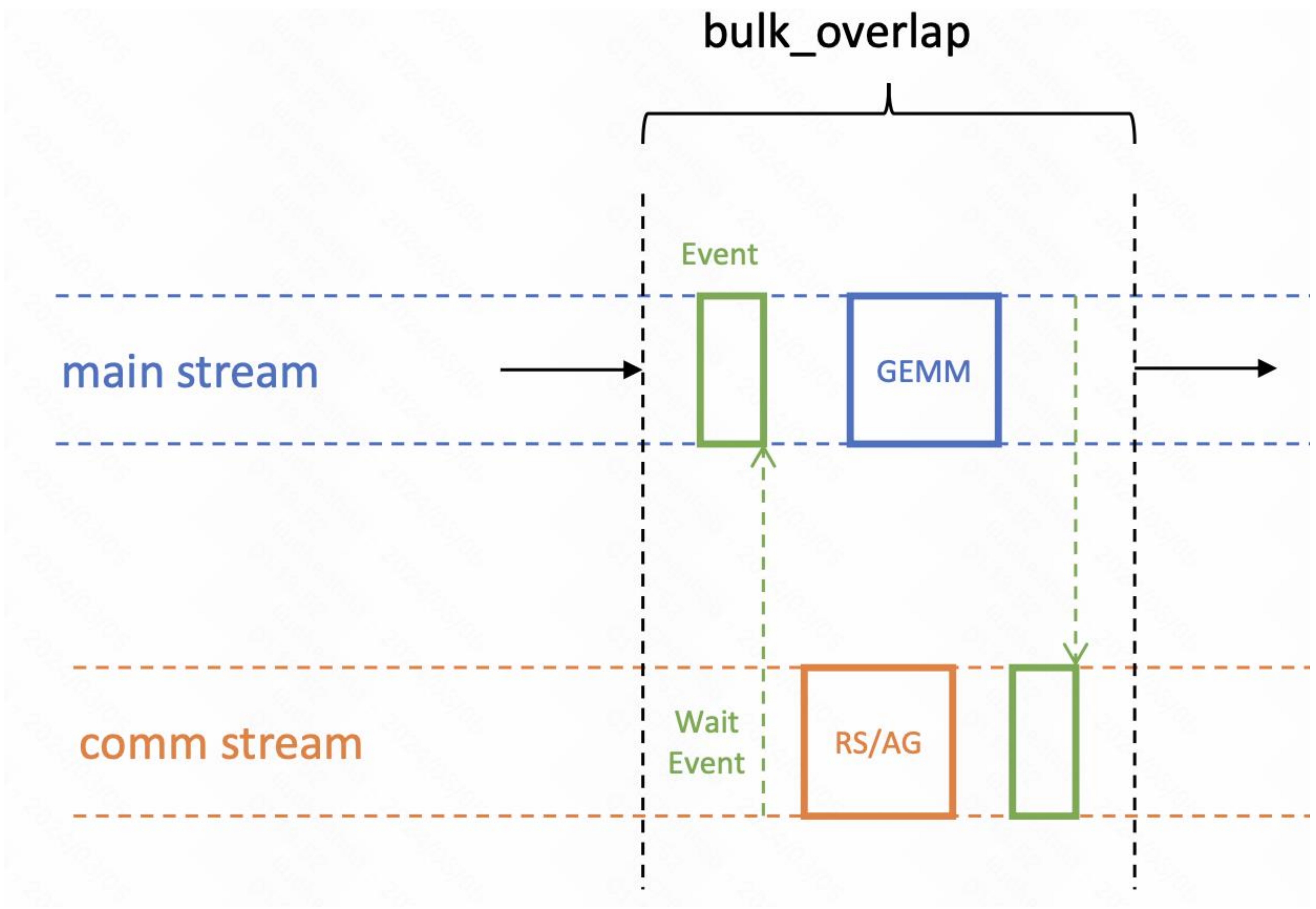
# Overlap optimization - Tensor Parallel

## Challenges

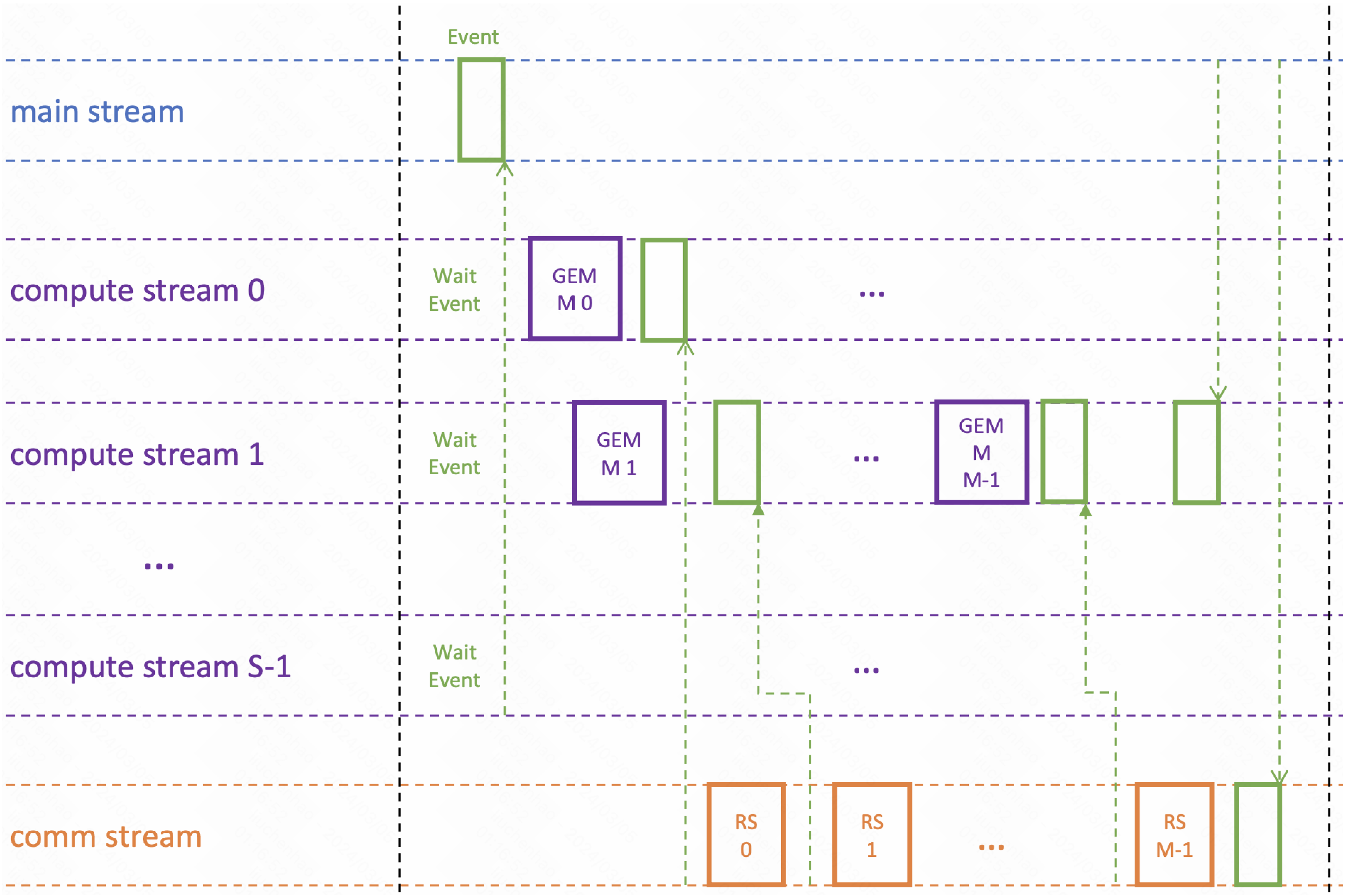
- As the model size expands and the sequence length increases, The increase in the Tensor Parallel size from 2 to 8, along with the expansion of sequence length, both contribute to a more significant burden on TP communication. This heightened communication load can lead to performance bottlenecks.
- In 175b 4k sequence length training， the communication overhead in Tensor Parallelism reaches 40%.

# Overlap optimization - Tensor Parallel

## Solution



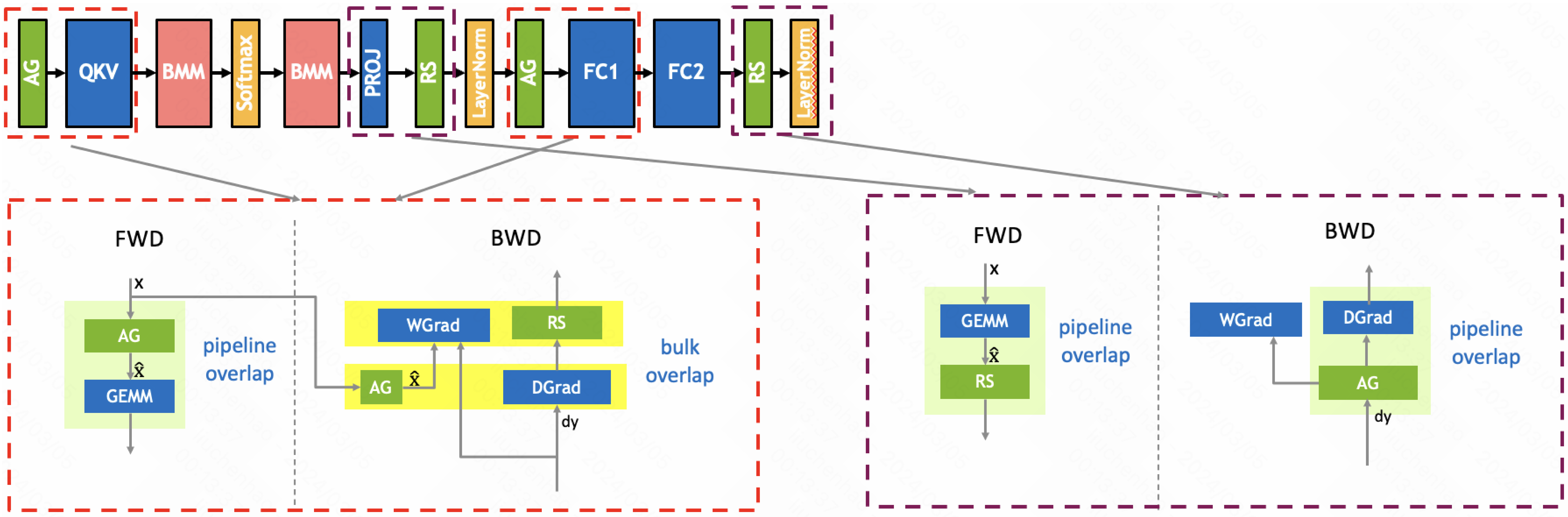

Bulk overlap




Pipeline split overlap

# Overlap optimization - Tensor Parallel

## Solution



# Overlap optimization - Tensor Parallel

## achievement


On a 2K-scale cluster, we have achieved a 30% acceleration in the forward pass and a 10% end-to-end acceleration by TP overlap.

快手

NVIDIA

# Long context window

快手

## Motivation

- Major players in the realm of large models have expanded their context windows to over 100k, with models like Claude 3 and Gemini 1.5 Pro supporting a context window size over 1M.
- Moreover, the capability to train with long contexts is a prerequisite for replicating Sora, which handles individual video inputs equivalent to approximately 1M tokens.

## Challenges

For a 175B model with a 32k context window, even when employing TP with a size of 8 and PP with a size of 8

$$\text{Activatifon} = 37.3bs\frac{n}{TP}L = 180GB > 80GB$$

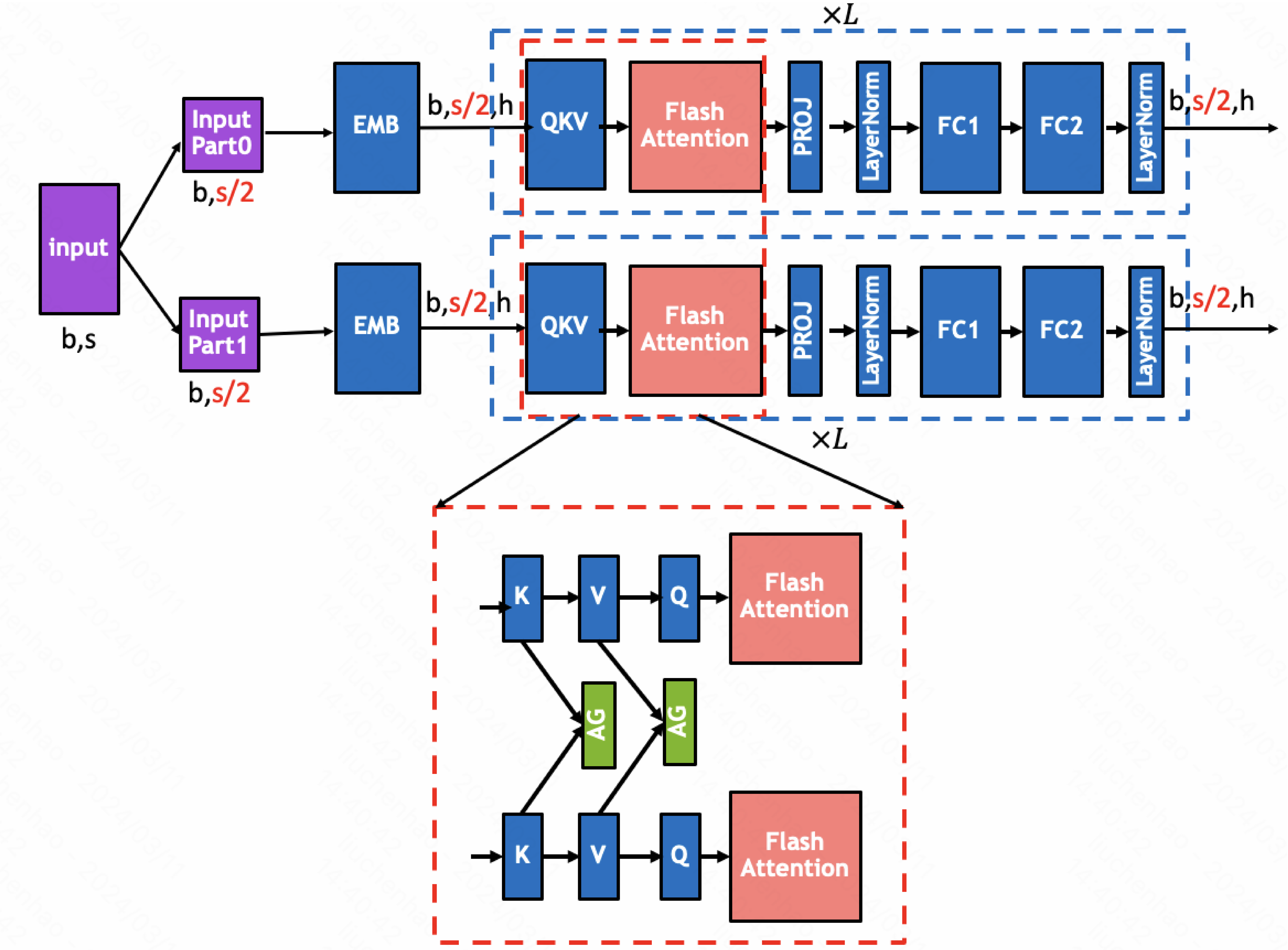
I will introduce three ways to address the significant memory demands for activation:

- Communication for Memory – Context Parallel
- Computation for Memory – Gemm-last Recomputing
- Main memory for Memory – Pipeline-aware offload

# Long context window

## Context parallel

- **Why not TP**
  - As Tensor Parallelism (TP) scales up and exceeds the NVLink domain, increased communication overhead is unacceptable.
  - Since TP splits the model dimensions at the head level, in extreme cases, it becomes infeasible not only from a performance standpoint but also in terms of meeting the requirements, regardless of performance considerations.




## Effect

**Memory overhead**

- The model 's activations are consistently partitioned along the s-dimension throughout the entire process.

**Scalability**

- The splitting dimension is on sequence. Therefore, theoretically, given a sufficient number of machines, we can address issues with any context window size.

**Communication overhead:**

- The amount of communication is 50% compared with TP. (10BshL VS 20BshL)
- In addition, the amount of communication introduced by CP is directly proportion to KV size, which could be partitioned by TP.
- By altering the computation order of the Q, K and V, it is possible to overlap the communication with computation.

# Long context window

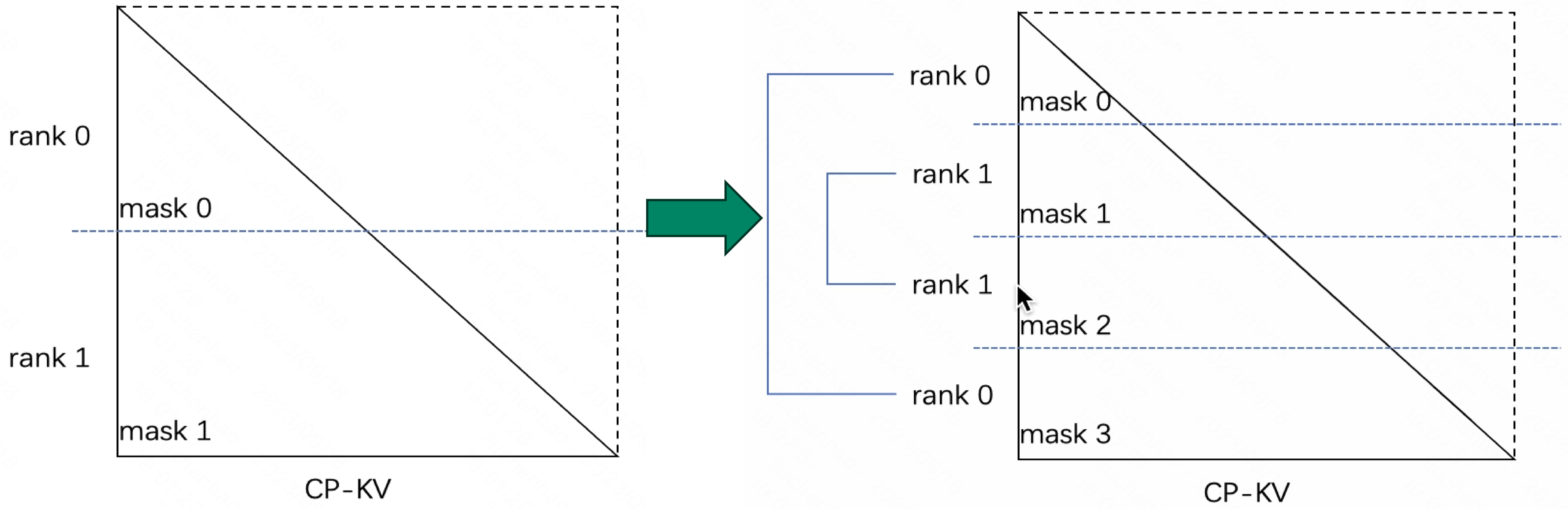
## Context parallel

### Computing load balance

**background**

Large Language Models (LLMs) that employ a decoder-only architecture typically use a causal mask within the attention mechanism. However, this causal masking leads to an uneven distribution of computational load in context parallelism (CP), as illustrated in the lower left figure.

**solution**




### Combining with GQA

**Background**

In long-context scenarios, GQA is almost a mandatory technology. GQA is an optimization technique that has been gaining traction in the development of LLMs designed to handle long sequences effectively.

**Effect**

GQA reduces the size of KV by grouping queries and using shared key and value pairs for each group. Since the communication volume in context parallelism is directly proportional to the size of the KV activations, GQA effectively decreases the amount of communication required in the context parallel.

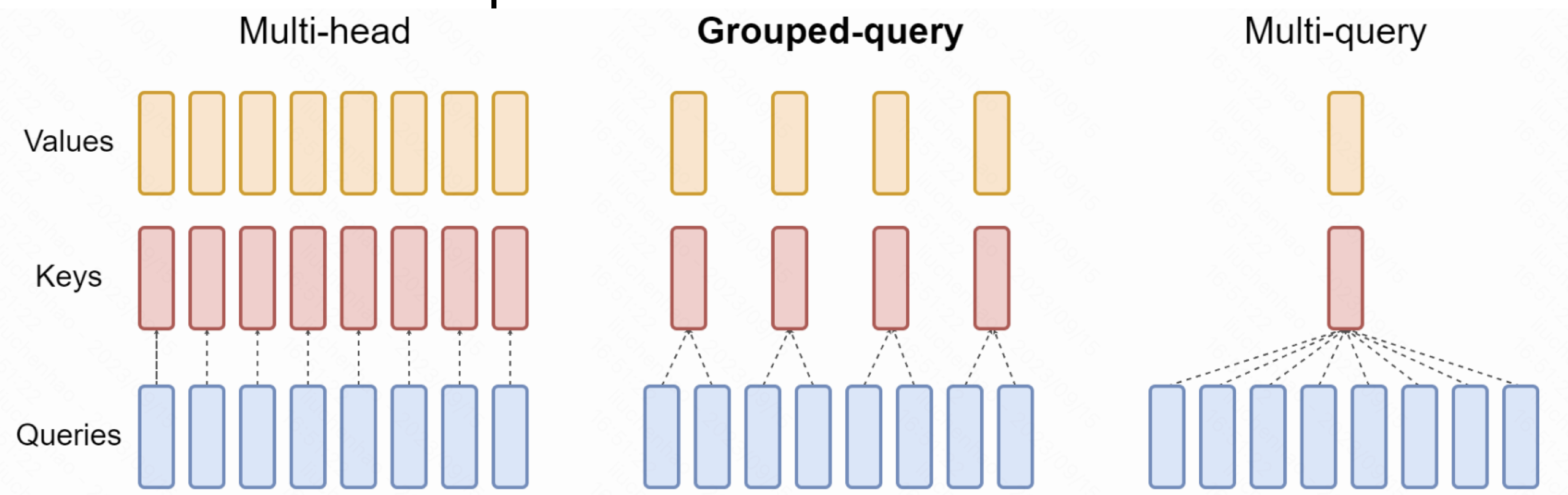
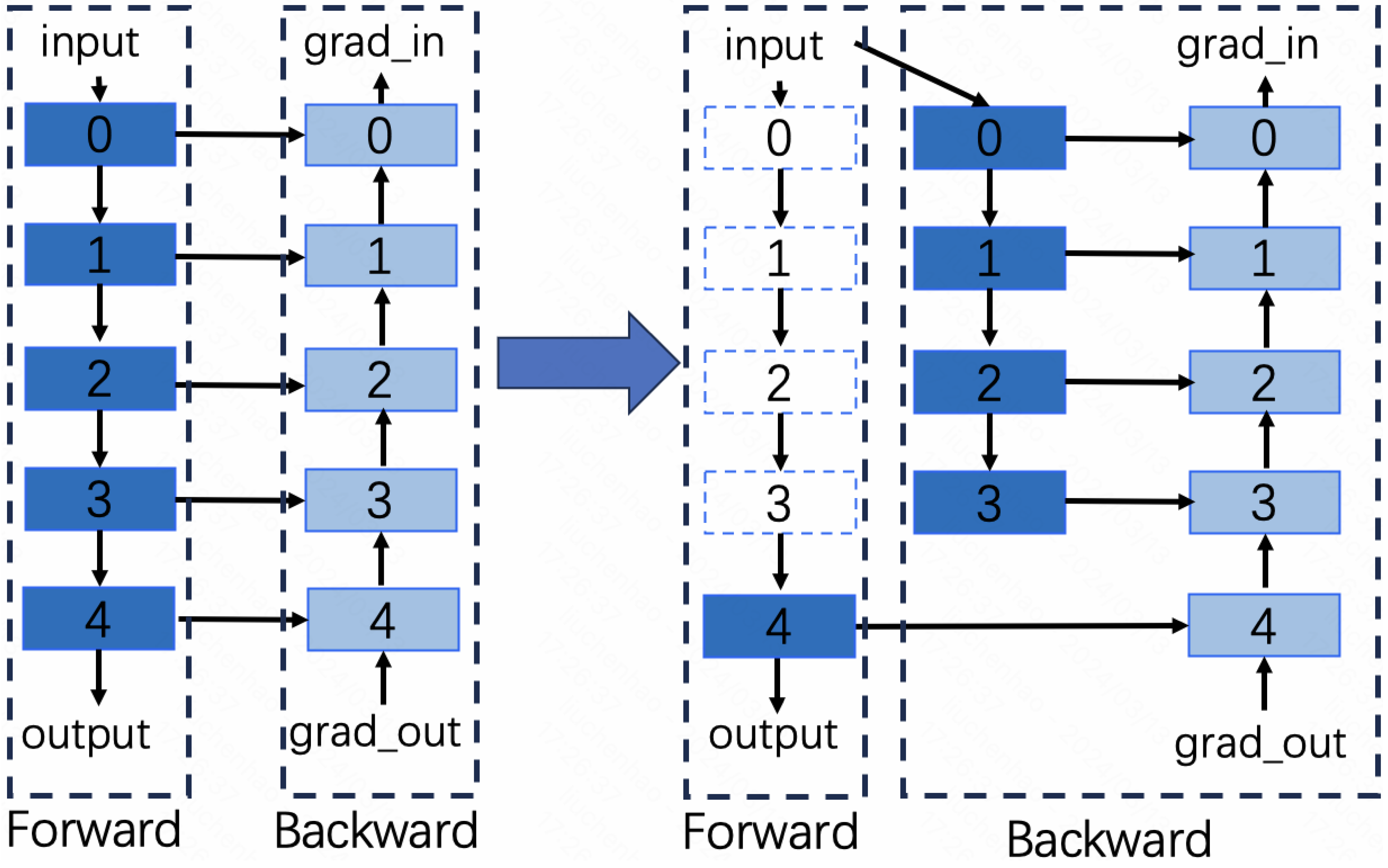



Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

# Long context window

## Gemm-last recomputing

### Recomputing Introduce




Non-recomputing

recomputing

### Recomputing limitation

Mainstream frameworks typically employ full recomputing, which leads to a complete recomputing of the forward pass during each backward pass, introducing approximately 30% of redundant computation.

Megatron-LM implemented selective recomputing for the attention part, but with the widespread adoption of flash attention,, where the memory cost of the attention part is no longer a hot topic, this solution is no longer as efficient.

# Long context window

## Gemm-last recomputing

### Recomputing optimization

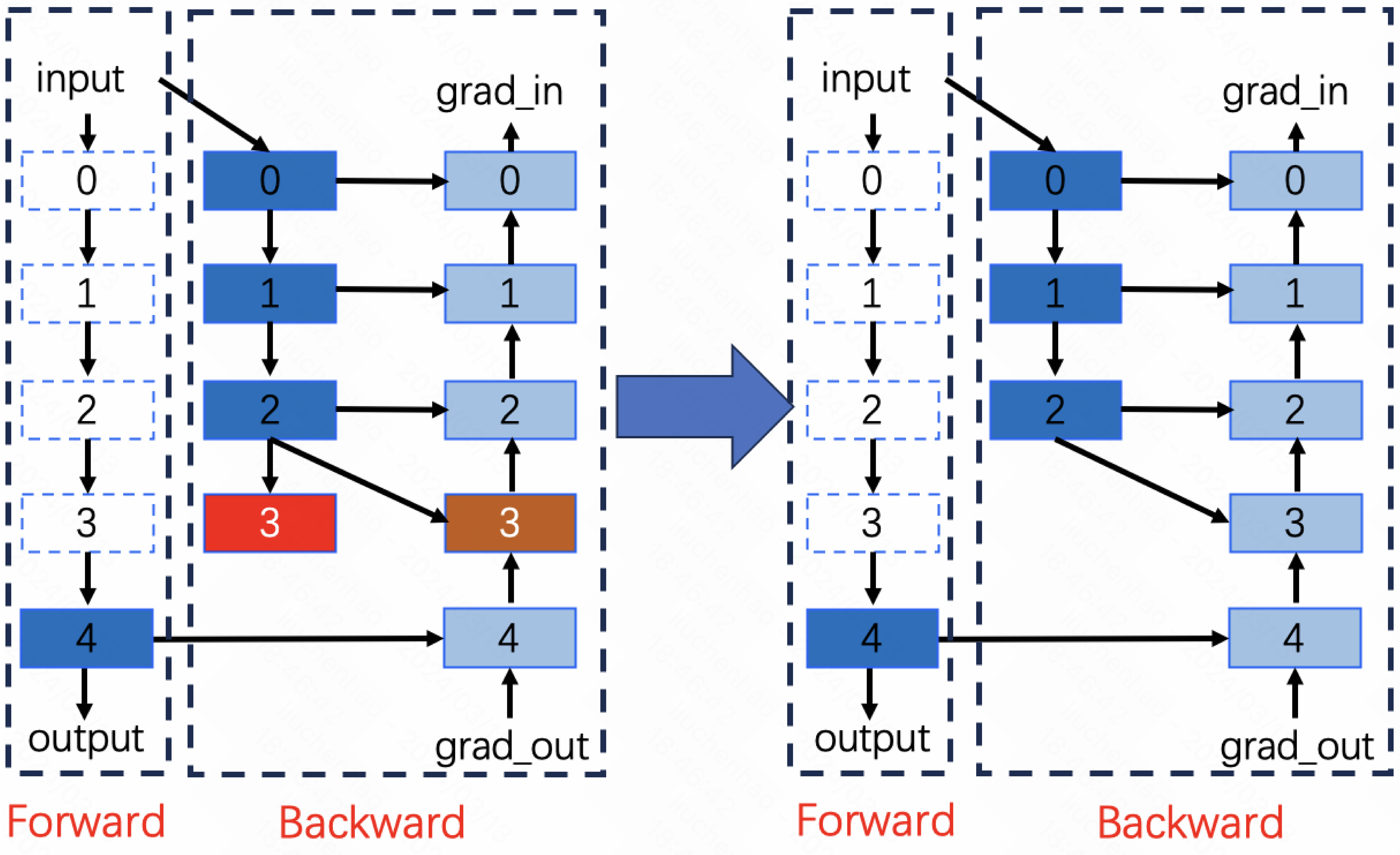
- Certain kernels, such as GEMM, do not depend on the forward output results for their backward computation, as illustrated in the figure on the right.
- When such operators are the last operator in a recomputing block, there is no need to recompute them.
- I refer to this recomputation strategy as "gemm-last recomputing."

### Achievement

- Only operators with low computational load, such as residual-add and layer norm, need to be recomputed.
- Compared to a no-recomputing approach, our strategy reduces the memory footprint by 40% while increasing the computational load by less than 1%.

In summary, we have derived the backpropagation expressions for the matrix-matrix product $Y = XW$:

$$\boxed{\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} W^T} \qquad \boxed{\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}} \tag{26}$$



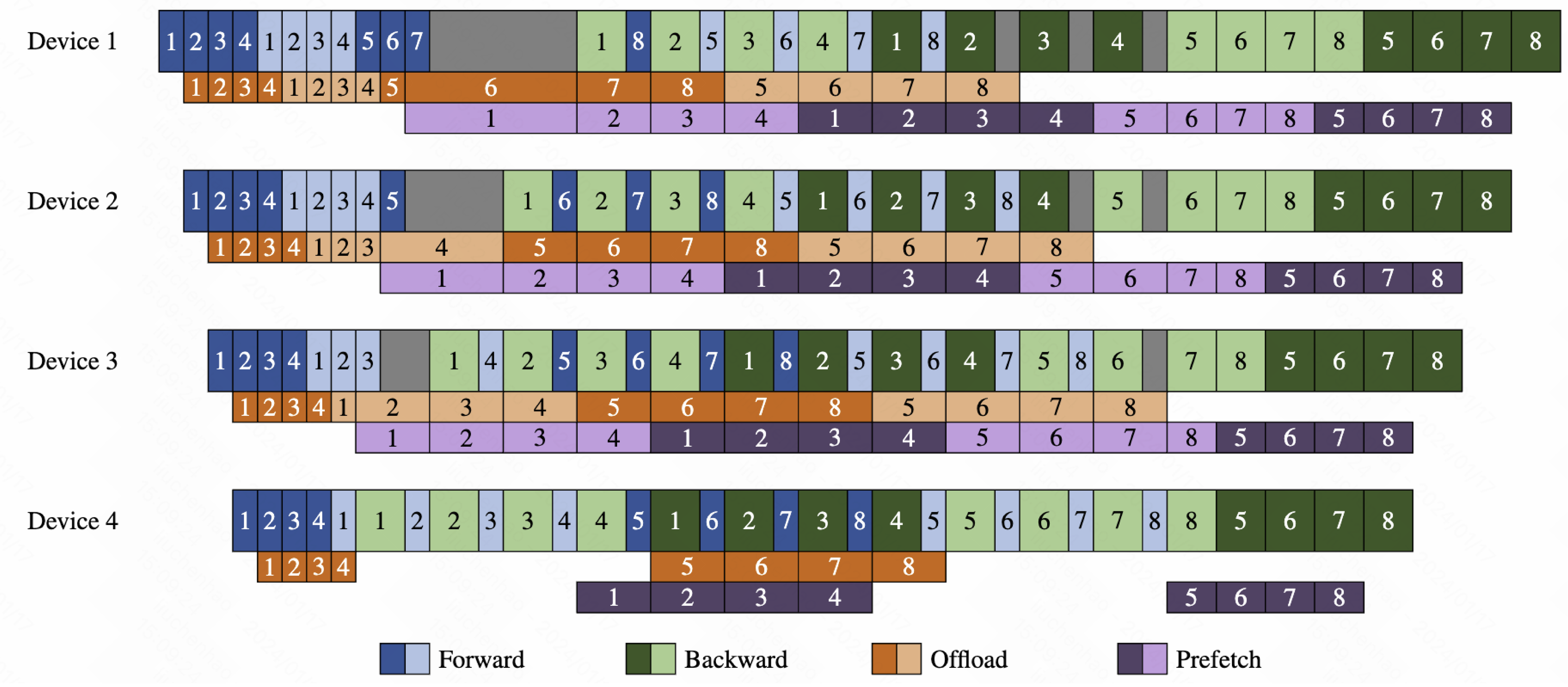
# Long context window

## pipeline aware offloading

### Motivation

- Host memory resources are often underutilized during the training process, while GPU memory (VRAM) resources are in short supply.
- On the Hopper, the upgrade to PCIe generation 5 provides each card with x16 bandwidth of 64GB/s; Additionally, host-to-device (H2D) and device-to-host (D2H) transfers have negligible impact on computation.
- In hybrid parallel scenarios, activations generated from forward computations are not immediately used but are separated by at least one complete virtual pipeline computation.

### Solution




### Achievement

- By employing an offloading strategy that minimally impacts computation, we achieve a memory-for-VRAM swap, effectively increasing the context window size by 2.5 times.

# Long context window
## achievement

| Model | Context window | Before Optimization | After Optimization |
|---|---|---|---|
| 175b | 4k | 37% | 44.9% |
| | 8k | 31% | 44.6% |
| | 16k | 33.3% | 45.1% |
| | 32k | 30.2% | 42.7% |
| 66b | 4k | 35.6% | 42% |
| | 8k | 34.2% | 42.5% |
| | 16k | 34.6% | 41.8% |
| | 32k | 28.9% | 38.9% |
| | 64k | 21.9% | 35.6% |

*The table data was acquired in a 512-GPU cluster with H800 GPUs equipping 8 x 100Gb/s of bandwidth.*

### Throughput

- By implementing context parallelism with a lower communication cost, it reduces the overhead associated with mitigating GPU memory (VRAM) issues.
- Additionally, by adopting two high cost-effectiveness solutions for alleviating VRAM constraints—gemm-last recomputing and pipeline-aware offloading—we decrease the reliance on communication in exchange for memory, thereby further enhancing the training throughput.
- Compared to SOTA open-source solutions, there is more than a 30% improvement in throughput across any context window size, as shown in the left figure.

### Context window size

- By employing strategies that trade memory for VRAM, it significantly increases the context window size that can be supported by a single computational device.
- The solution is highly scalable, and given sufficient computational resources, it can theoretically support an unlimited context window size.

# Summary

# Summary

- Training large language models needs large memory, communication volume and huge computing resources.
- Model parallelism is a MUST have, we introduce Tensor parallelism and pipeline Parallelism.
- There are several memory-saving technical methods such as Sequence parallelism, activation recomputation, distributed optimizer, and offloading.
- **Megatron core** is a library for GPU-optimized techniques for LLM training, which has integrated most technical points.
- Shared Kuaishou LLM training best practice, GPT 175B on 2K Hopper clusters achieves E2E up to 4**0%** acceleration improvement

| Evolution of LLM | Hotspots in LLM training | Solutions | Achievements |
| --- | --- | --- | --- |
| more GPUs<br>GPU upgrade | Data Parallel time increase | Overlapping in Data Parallel | 15% acceleration |
| model size grows<br>sequence length increase | Tensor Parallel time increase | Overlapping in Tensor Parallel | 5-10% acceleration<br>Depend on model size |
| sequence length increase to 1M | Tensor Parallel needs to cross nodes to meet the increasing memory demands | • Context parallel<br>• Gemm-last Recomputing<br>• Pipeline-aware offloading | 30% acceleration |

# Future Work

# Future Work

- Performance optimizations for Mixture-of-experts models
- Efficient CUDA Kernels implementation on Hopper or the next-gen GPU.
- MultiModal models(text-to-image, text-to-video, image-to-video) training and optimizations

Thank you!

Yuliang Liu: liuyuliang@kuaishou.com
Xue Huang: xueh@nvidia.com