Taming Load Balancing in Distributed LLM Training

Jiale Xu

Table of Contents

Background: Distributed LLM Serving Paradigm-Memory Footprint

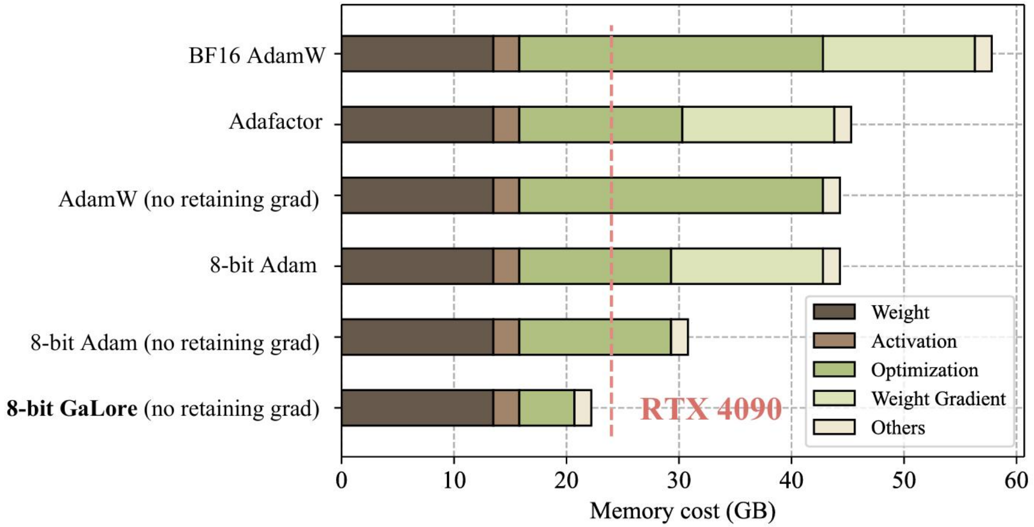


Figure 1: Estimated memory consumption of pre-training a LLaMA 7B model with a token batch size of 256 on a single device, without activation checkpointing and memory offloading[2]. Details refer to Section 5.5.

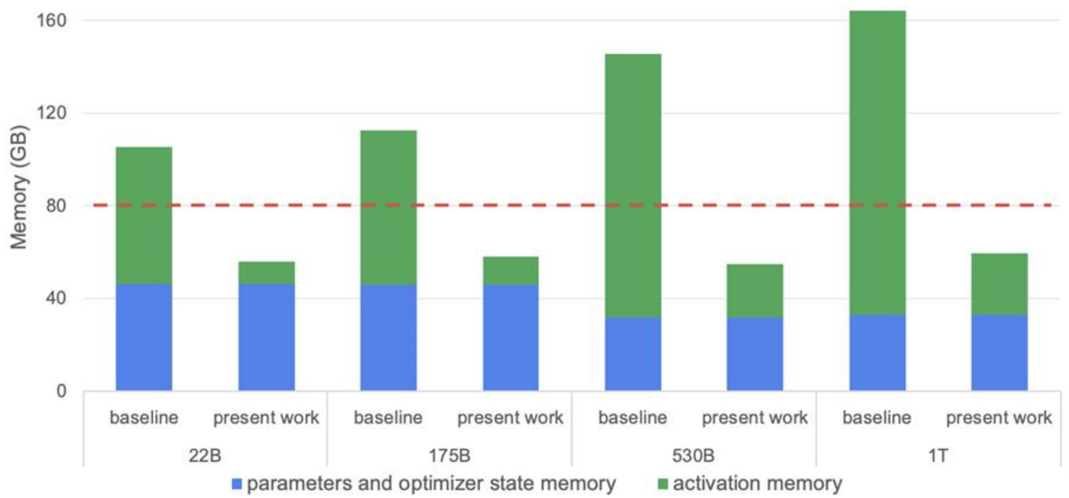


Figure 1: Parameters, optimizer state, and activations memory. The dashed red line represents the memory capacity of an NVIDIA A100 GPU. Present work reduces the activation memory required to fit the model. Details of the model configurations are provided in Table 3.

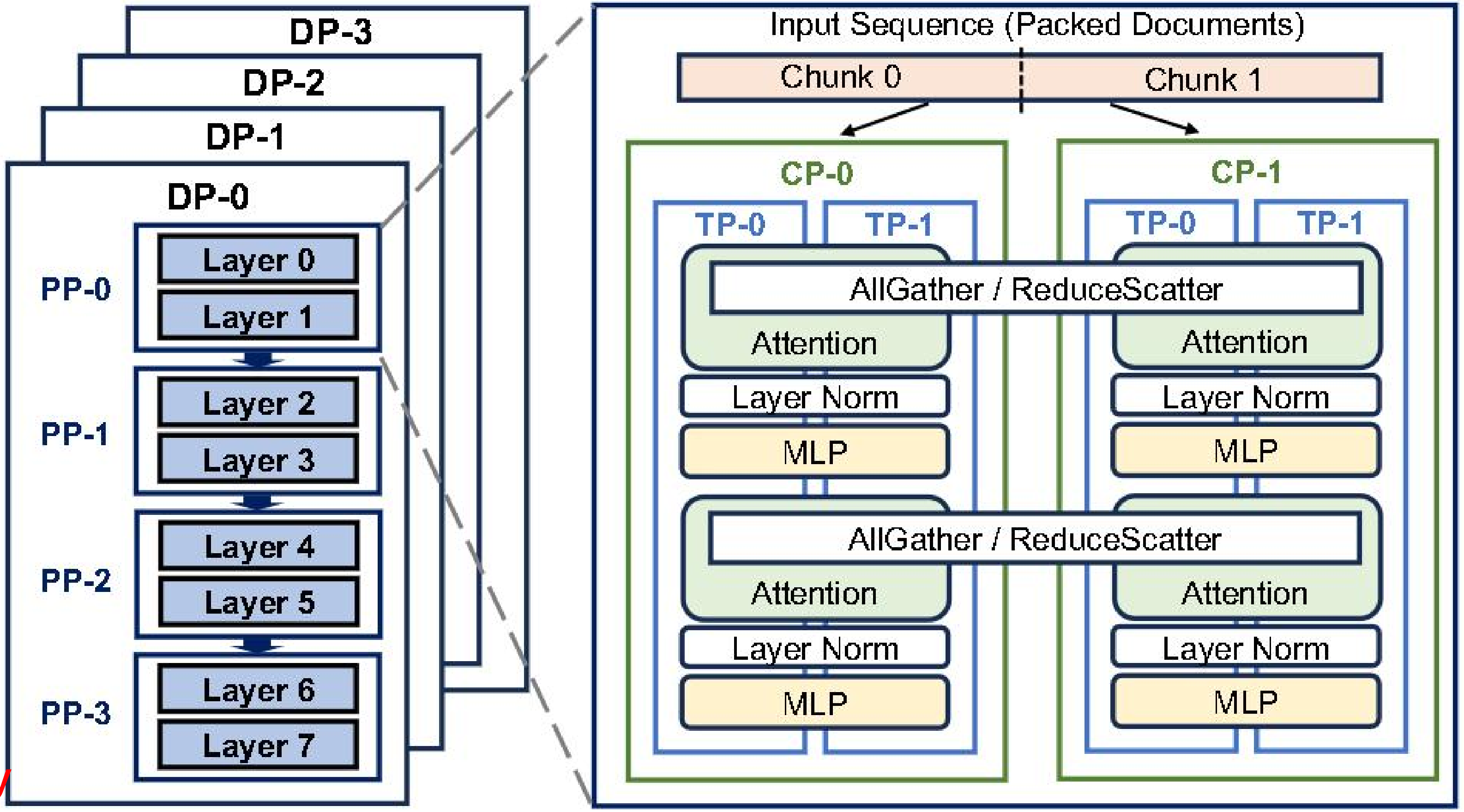
Model weight and Optimizer State is dominating when apply optimizations. → Activation becomes a bottleneck in memory footprint.

After distributed training and model compression techniques.

Background: Distributed LLM Serving Paradigm-4D Parallel

1. Data Parallel
Enlarge input batch Size while duplicating model weight memory

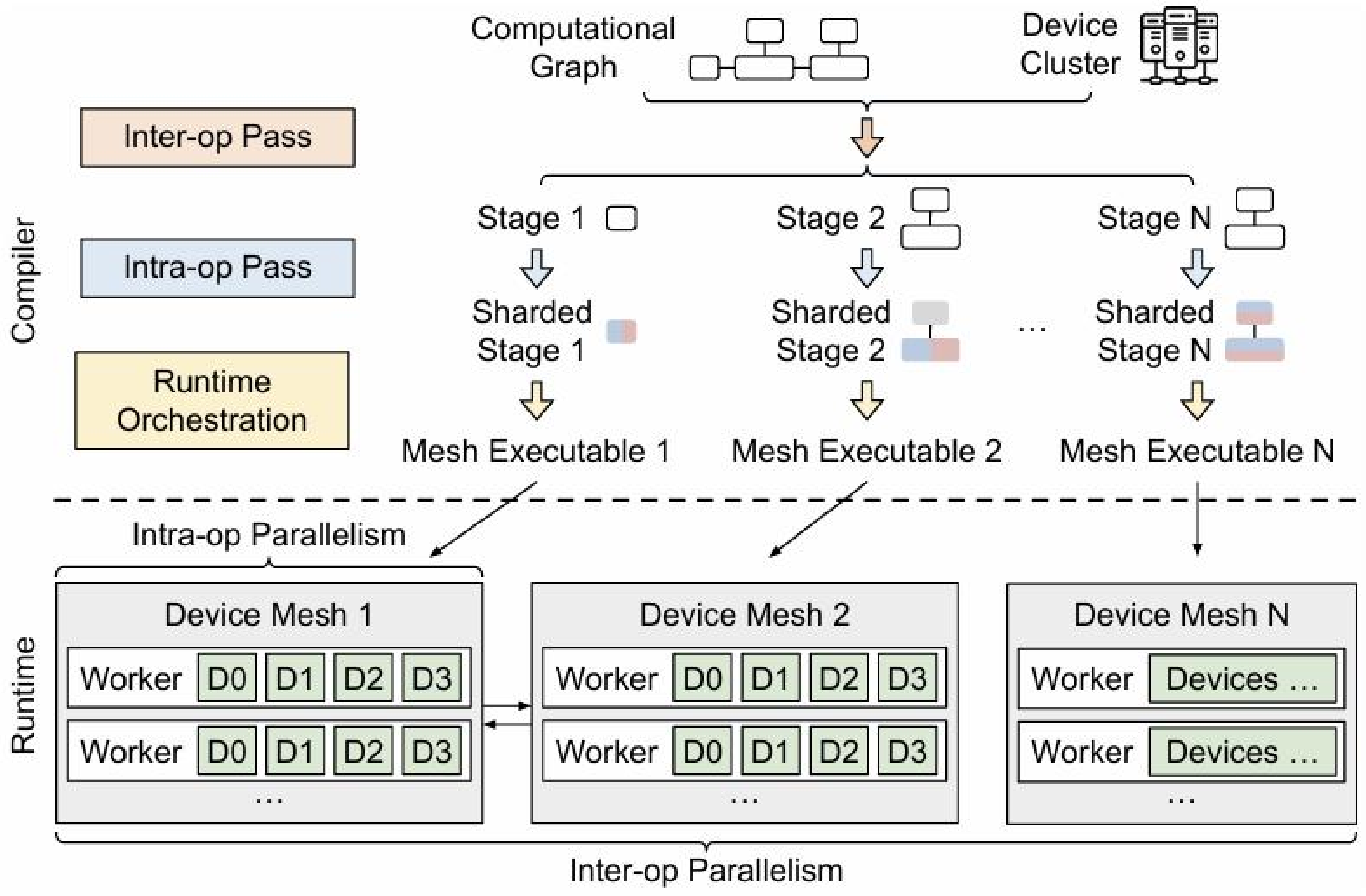
2. Pipeline Parallel
Reducing model weight memory while undermining compute efficiency



3. Context Parallel
Reducing activation memory & increasing compute capability while duplicating model weight memory & communication overhead

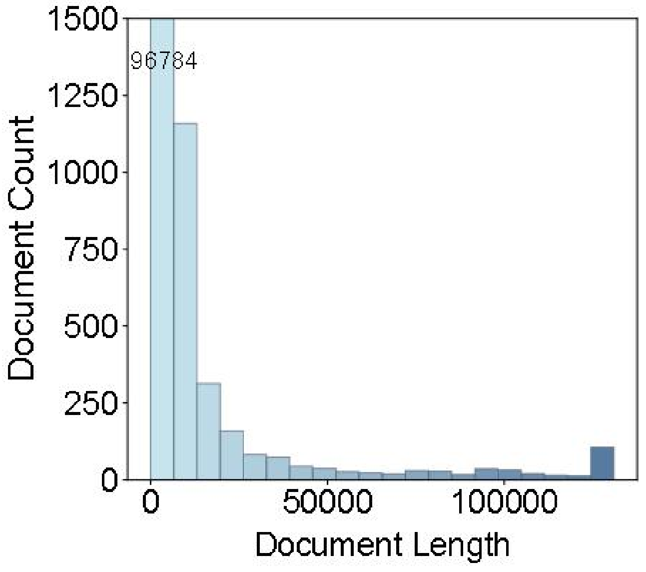
4. Tensor Parallel
Reducing overall memory & increasing compute capability while introducing communication overhead

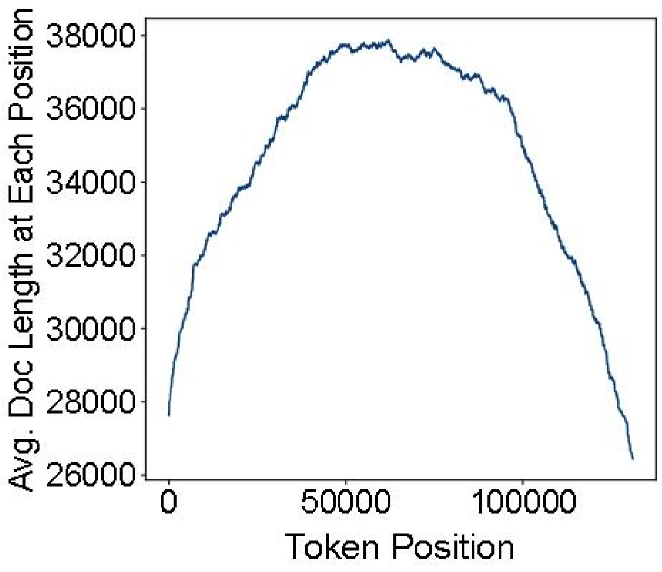
Eternal Goal: Finding a more Efficient Distributed Training Plan



Challenges

Llama2-7b: 4K → Qwen-2.5: 1M





Growing context window size

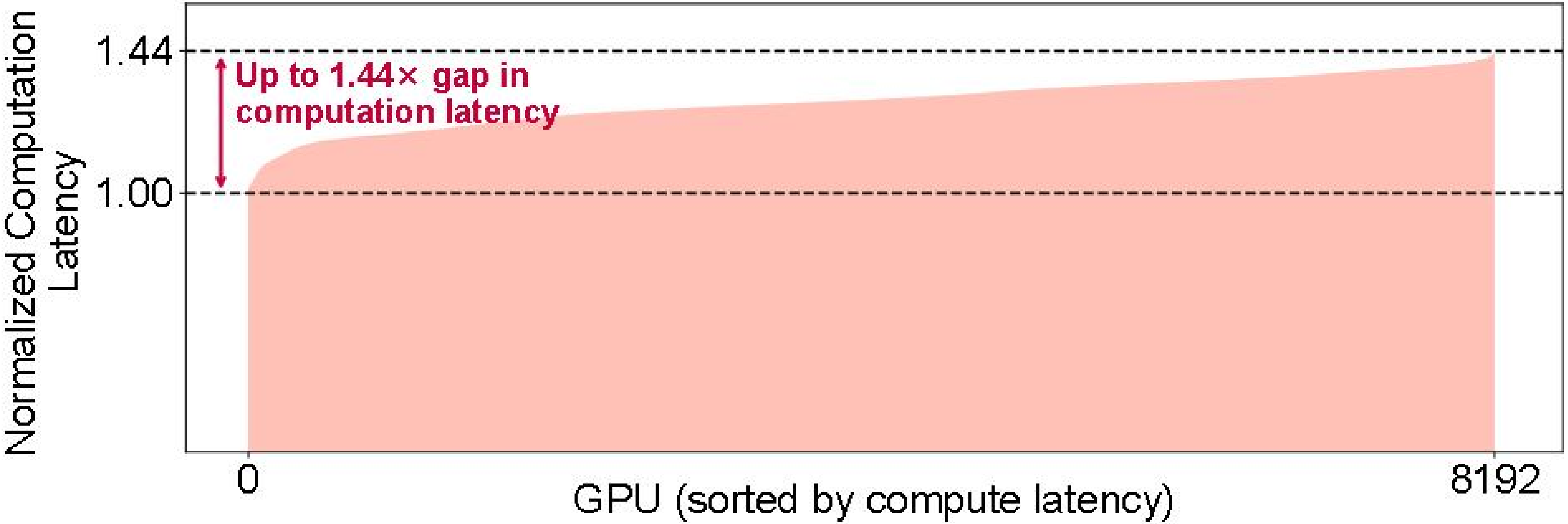
Varied dataset length distribution

OSDI'25 WLB-LLM

Workload-Balanced 4D Parallelism for Large Language Model Training

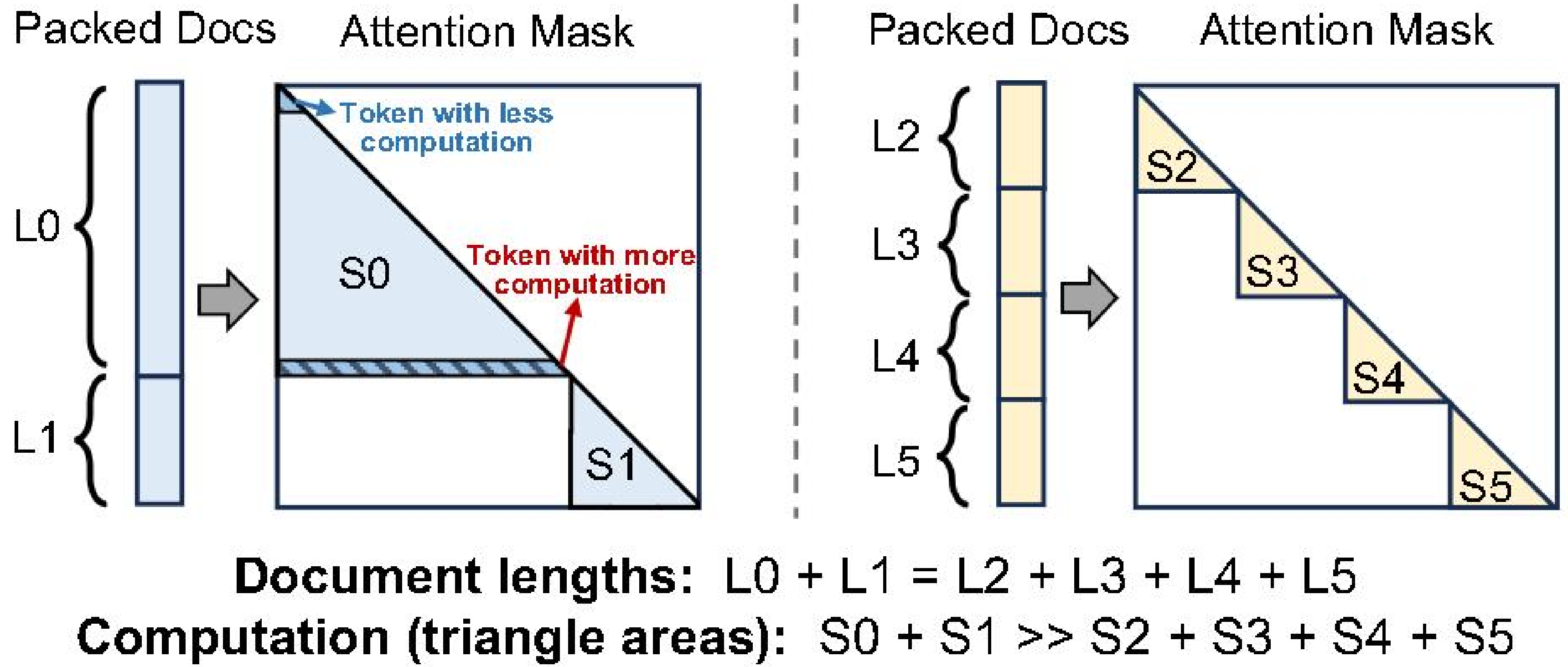
Zheng Wang[1,2], Anna Cai[2], Xinfeng Xie[2], Zaifeng Pan[1], **Yue Guan**[1], Weiwei Chu[2], Jie Wang[2], Shikai Li[2], Jianyu Huang[2], Chris Cai[2], Yuchen Hao[2], Yufei Ding[1,2]
University of California, San Diego[1] Meta[2]

Finding 1: Imbalanced computation latency within large-scale LLM training job.



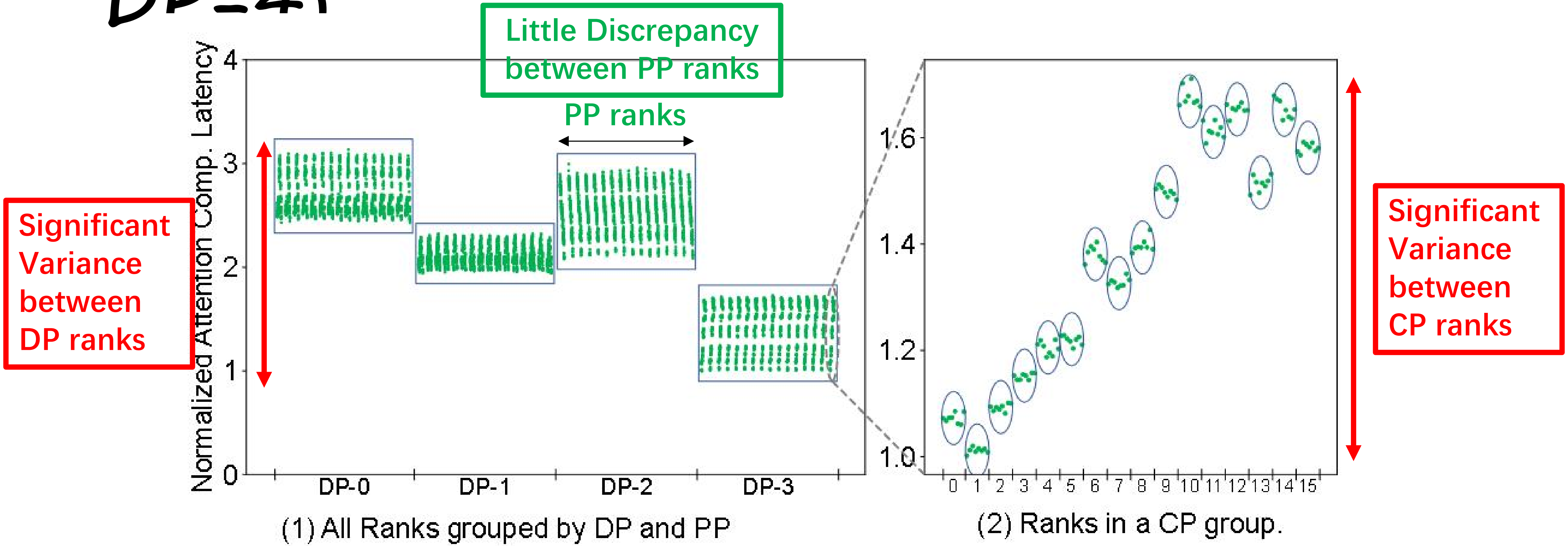
(a) Normalized computation latency in an 8K-GPU LLM training job.

Finding 2: Imbalanced Reason: Attention Computation Nature



(b) Reason of imbalance: input-dependent nature of attention computation and the varying input document length.

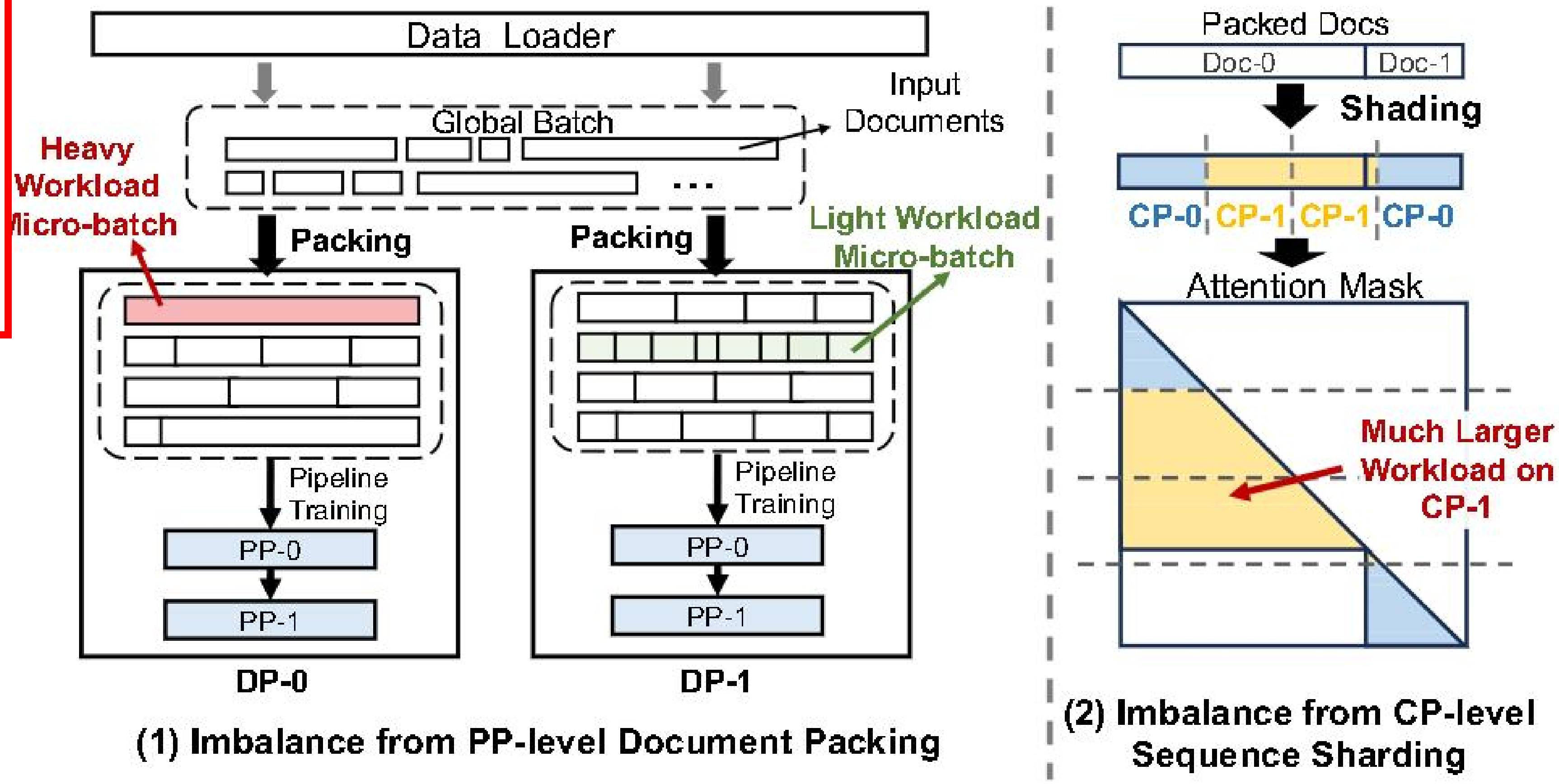
Case Study: (TP=8, CP=16, PP=16, DP=4)



(a) Imbalance Analysis (TP=8, CP=16, PP=16, DP=4): (1) Normalized computation latency (group by DP and PP); (2) Normalized computation latency in a CP group.

Case Study: (TP=8, CP=16, PP=16, DP=4)

DP ranks have different micro batches

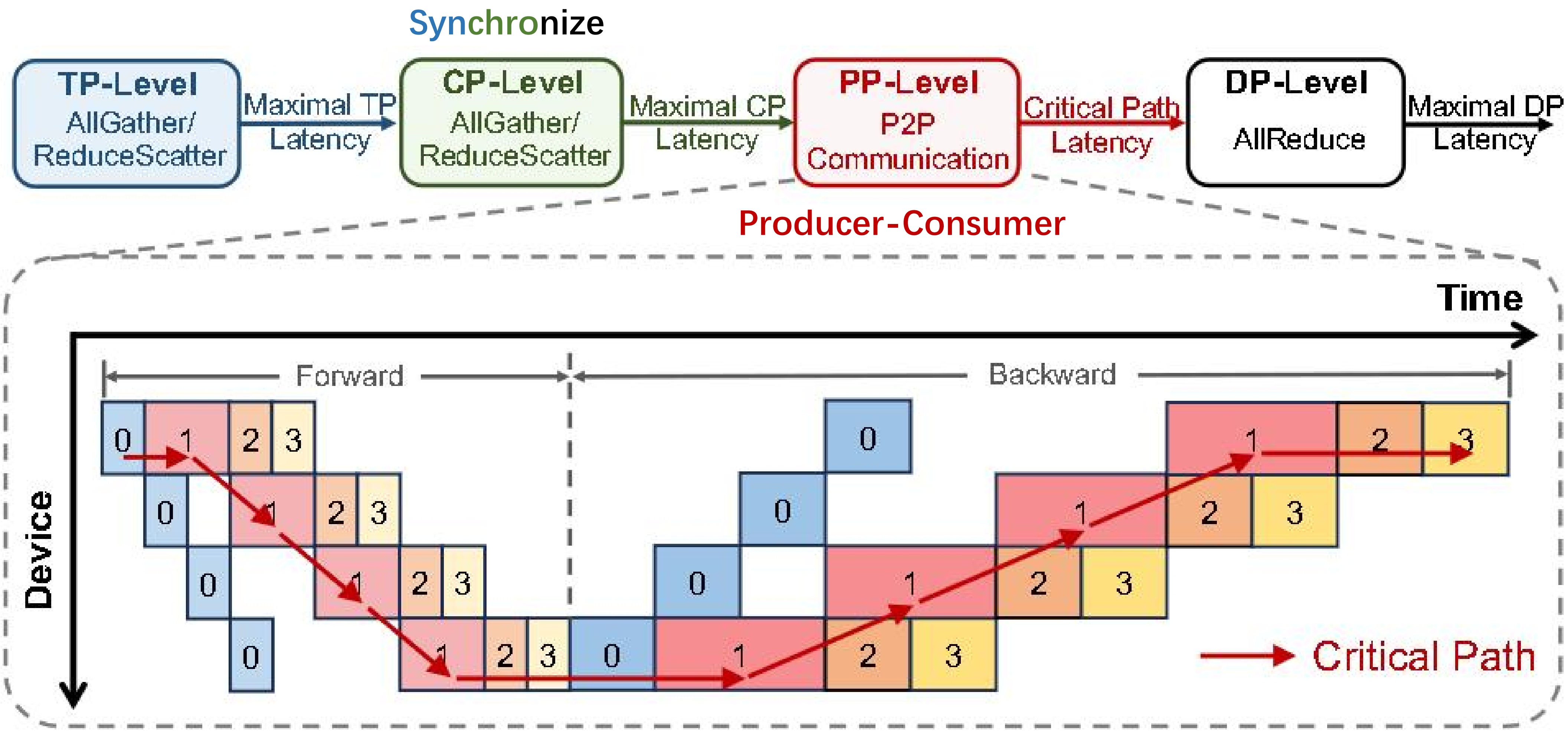


(b) Document packing at PP level and sequence sharding at CP level.

CP ranks have attention load balancing issue

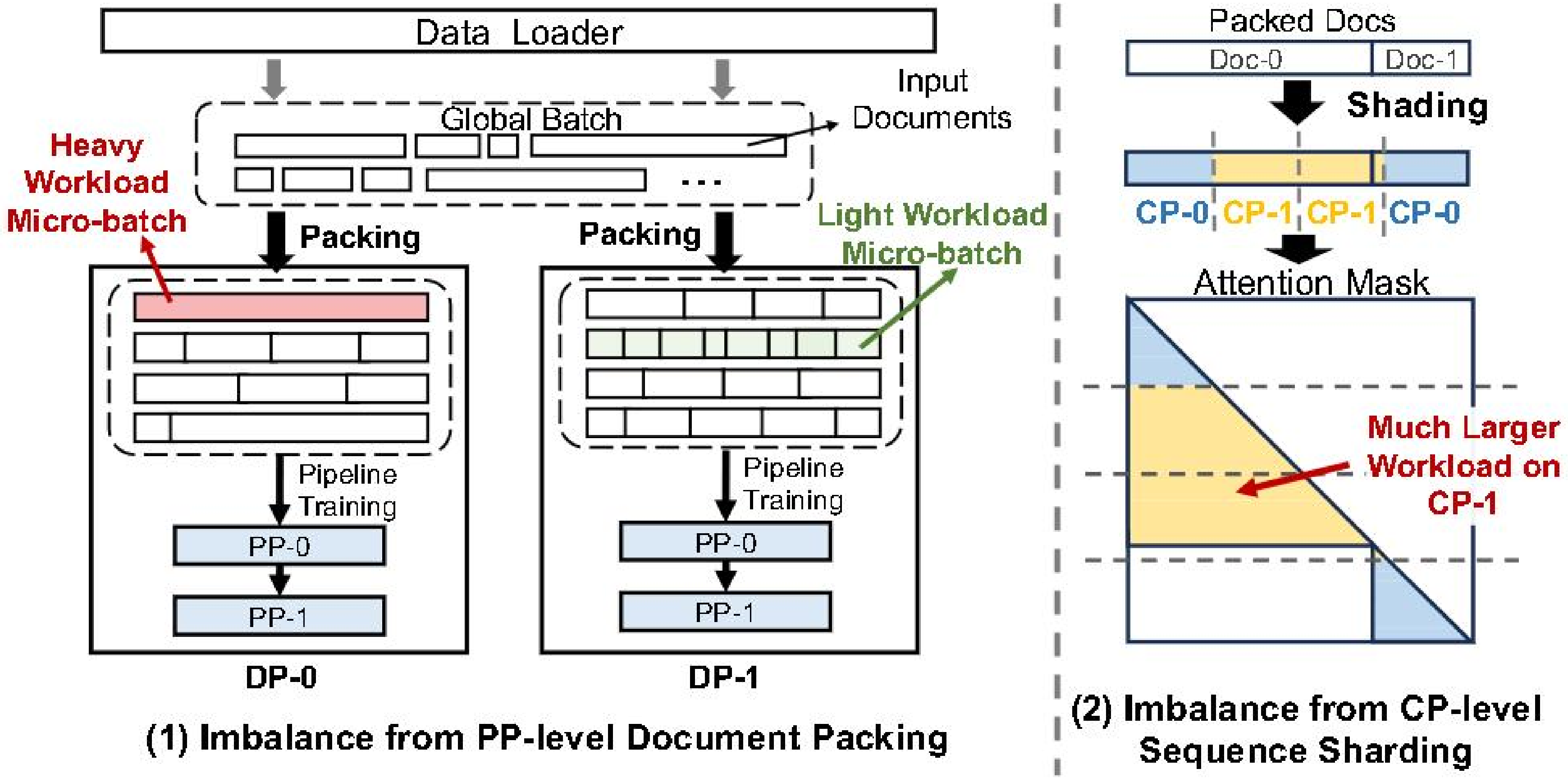
PP ranks share the same micro-batch

Latency Propagation Chain



Two Optimizations

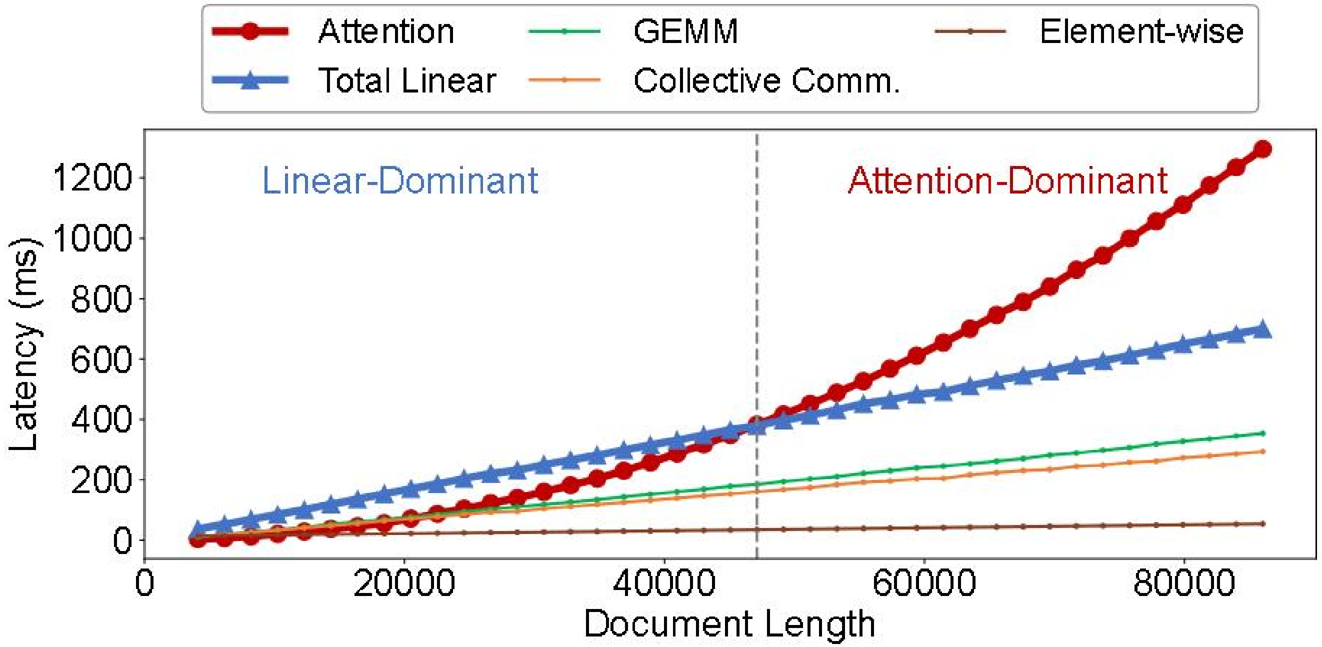
Better PP packing



(b) Document packing at PP level and sequence sharding at CP level.

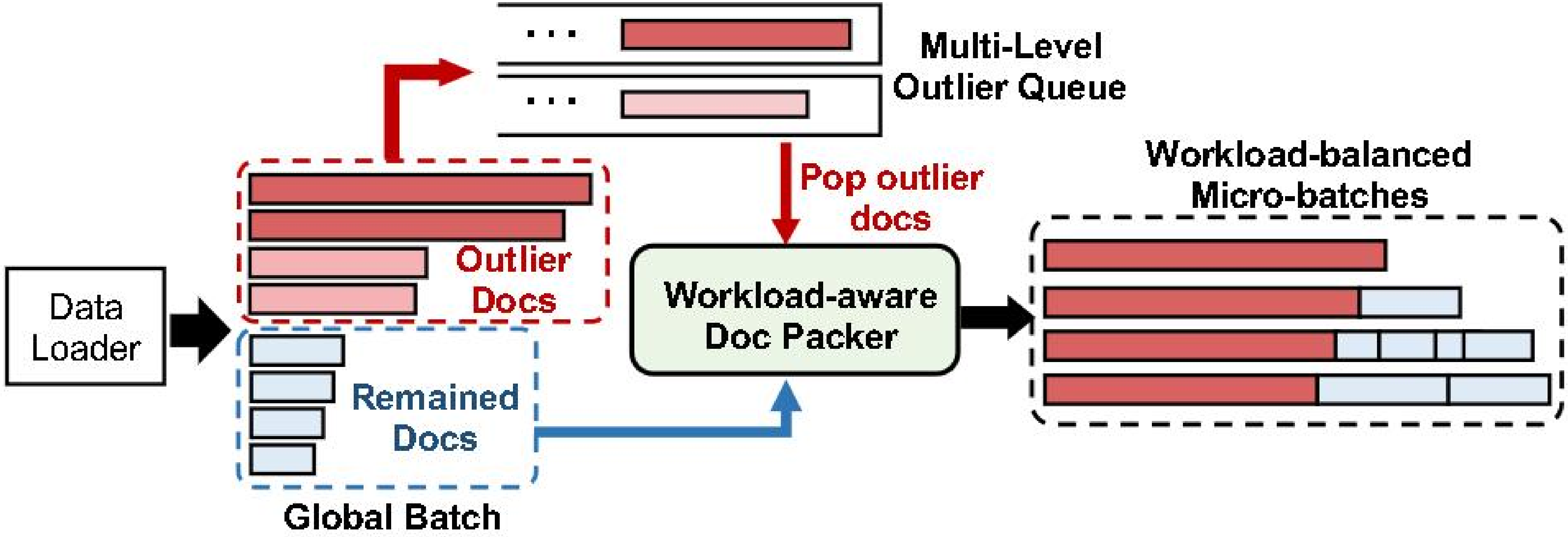
Better Document shard balancing

Better PP Packing

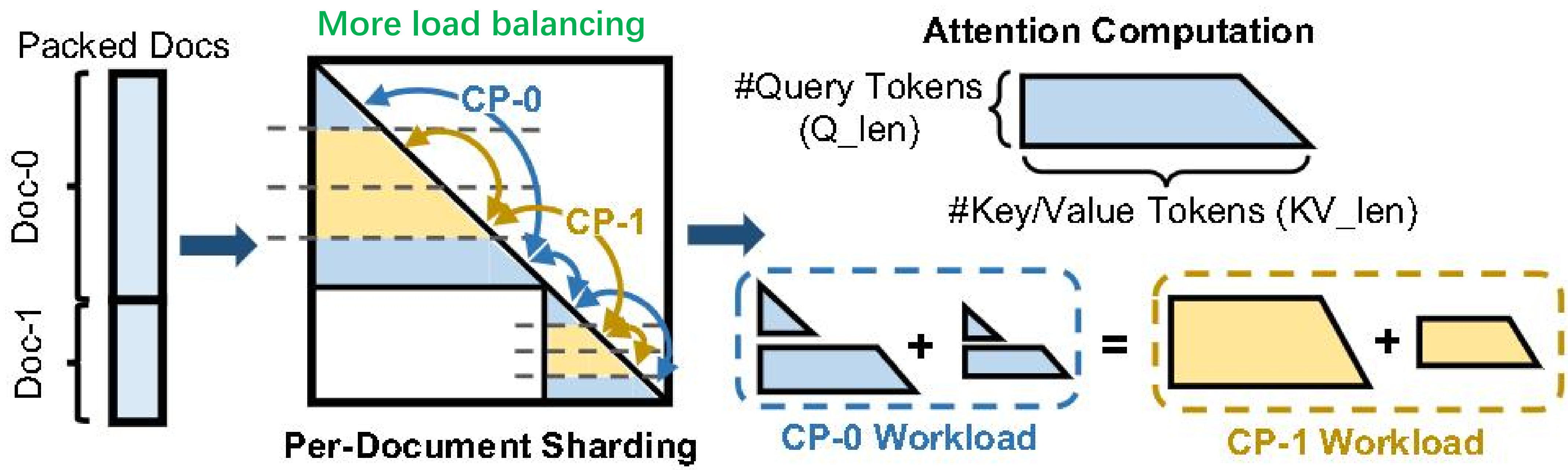


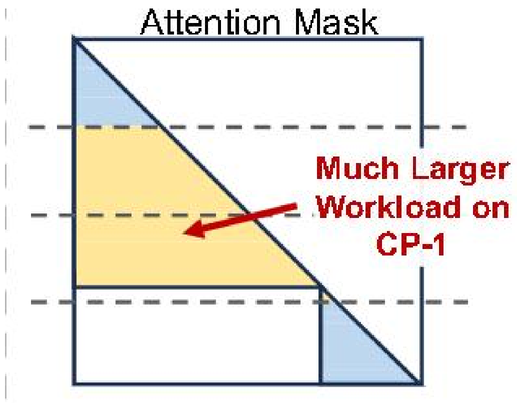
$$
\begin{aligned}
\text{minimize} \quad & \max\Big(\sum_{i=1}^{N}\big(W_a(x_{ij}\cdot d_i)+W_l(x_{ij}\cdot d_i)\big)\Big), & \\
& & j=1,\cdots,M \\
\text{subject to} \quad & \sum_{j=1}^{M} x_{ij}=1, & i=1,\cdots,N \\
& \sum_{i=1}^{N} x_{ij}\cdot d_i \le L_{max}, & j=1,\cdots,M \\
& x_{ij}\in\{0,1\}
\end{aligned}
\qquad (2)
$$

Better PP Packing

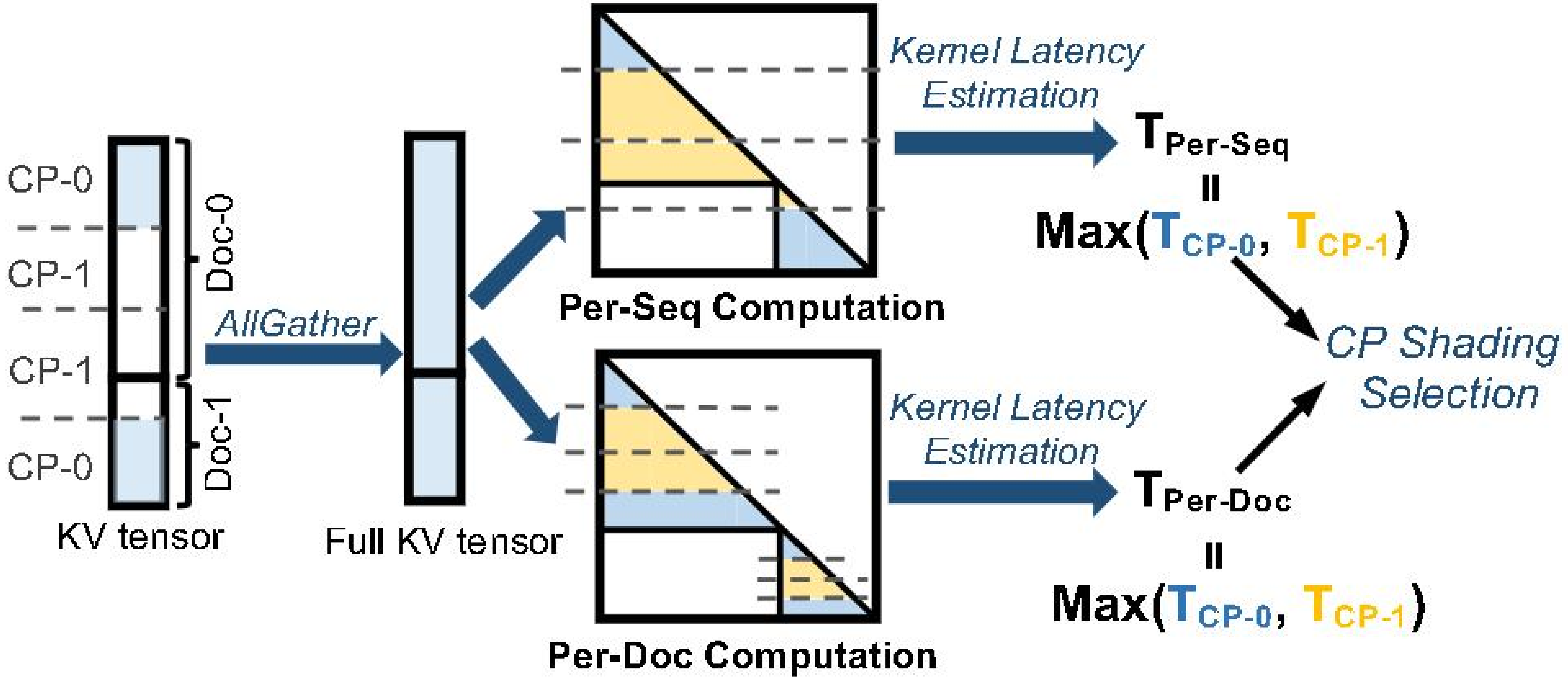


Better Document shard balancing





Better Document shard balancing



Less efficient computation

Better load balancing

Discussion On Kernel Load Balancing

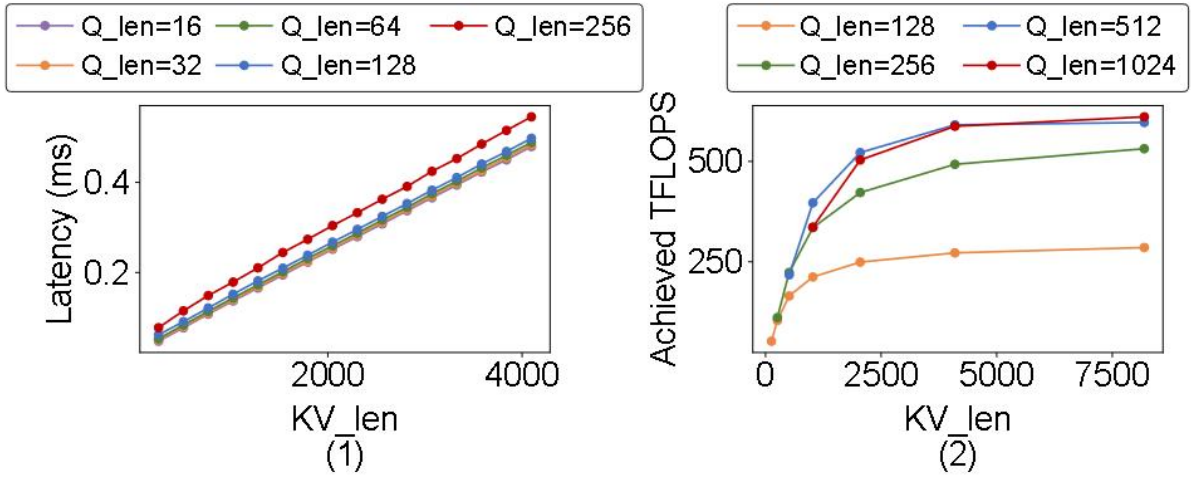
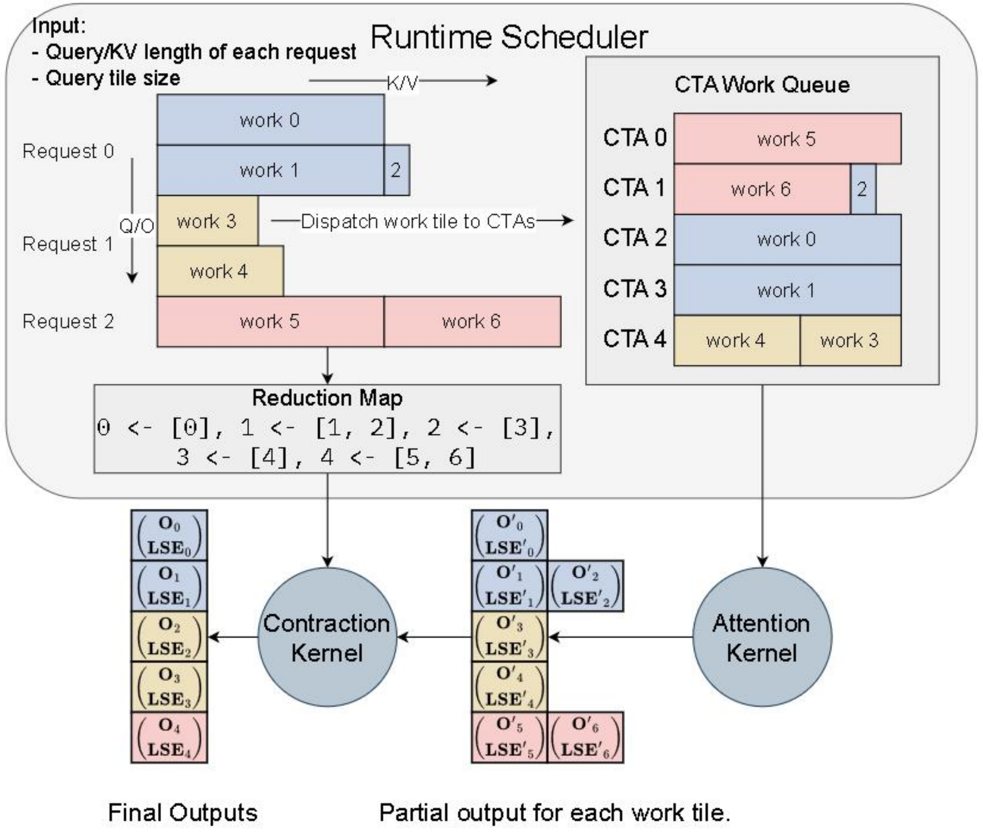


Figure 10: Attention kernel performance profiling: (Left) Attention forward latency; (Right) Achieved TFLOPs of the attention forward kernel.



FlashInfer with load balancing logic

Evaluation - Setup

Model Size	Context Window	#GPU	4D Parallelism Configs (TP, CP, PP, DP)
550M	*64K*	*32*	*(2, 2, 4, 2)*
	128K	*32*	*(2, 4, 4, 1)*
7B	*64K*	*32*	*(4, 2, 4, 1)*
	128K	*64*	*(8, 2, 4, 1)*
30B	*64K*	*64*	*(8, 2, 4, 1)*
	128K	*128*	*(8, 4, 4, 1)*
70B	*64K*	*256*	*(16, 4, 4, 1)*
	128K	*256*	*(16, 4, 4, 1)*

Table 1: Model and 4D parallelism configurations.

Evaluation – E2E

Takeaway #1. Larger Model, Less improve. Reason: Larger communication

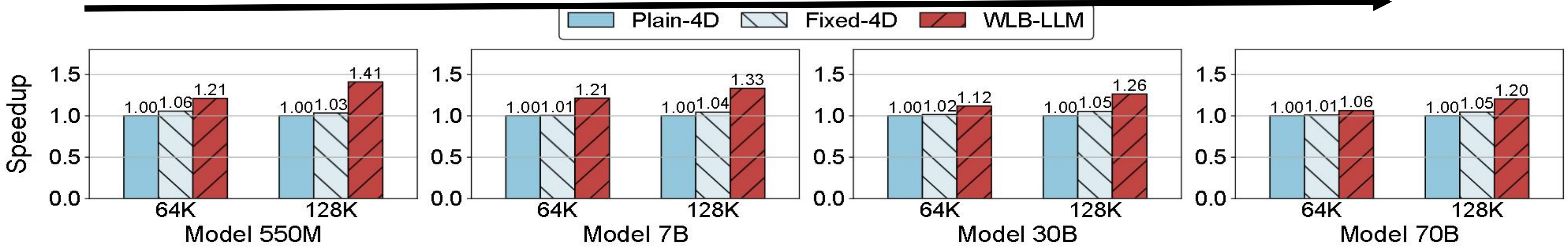


Figure 12: Training performance speedups of *WLB-LLM* and *Fixed-4D* over *Plain-4D* across various configurations.

Figure 13: Performance breakdown of *WLB-LLM* on the 7B model with a 128K context window.

Takeaway #2. PP-Level Load balancing is more important.

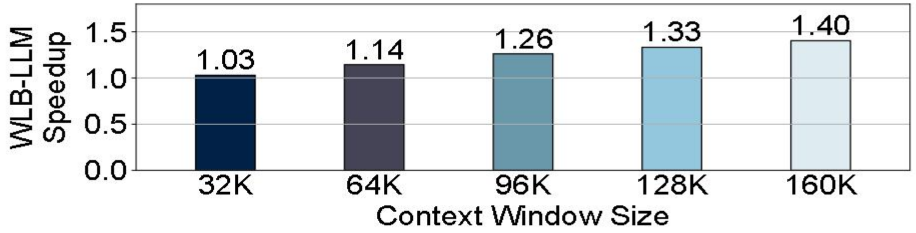


Figure 14: Speedups of *WLB-LLM* on the 7B model across context window sizes.

Takeaway #3. Larger Context lead to more speedup Reason: larger context window raises the likelihood of outlier documents appearing

Related Work(s)

SOSP'24 Enabling Parallelism Hot Switching for Ecient Training of Large Language Models

ASPLOS'25 FlexSP: Accelerating Large Language Model Training via Flexible Sequence Parallelism

	Targeted Parallelism
Hot-switching	TP
FlexSP	SP
WLB-LLM	PP+SP

Comments

Pros:

1. Direct and strong motivation and clear design writing.
2. Comprehensive evaluation with decent speedup, sensible analysis, and progressive breakdown.

Cons:

1. Not a very impressive/novel idea.

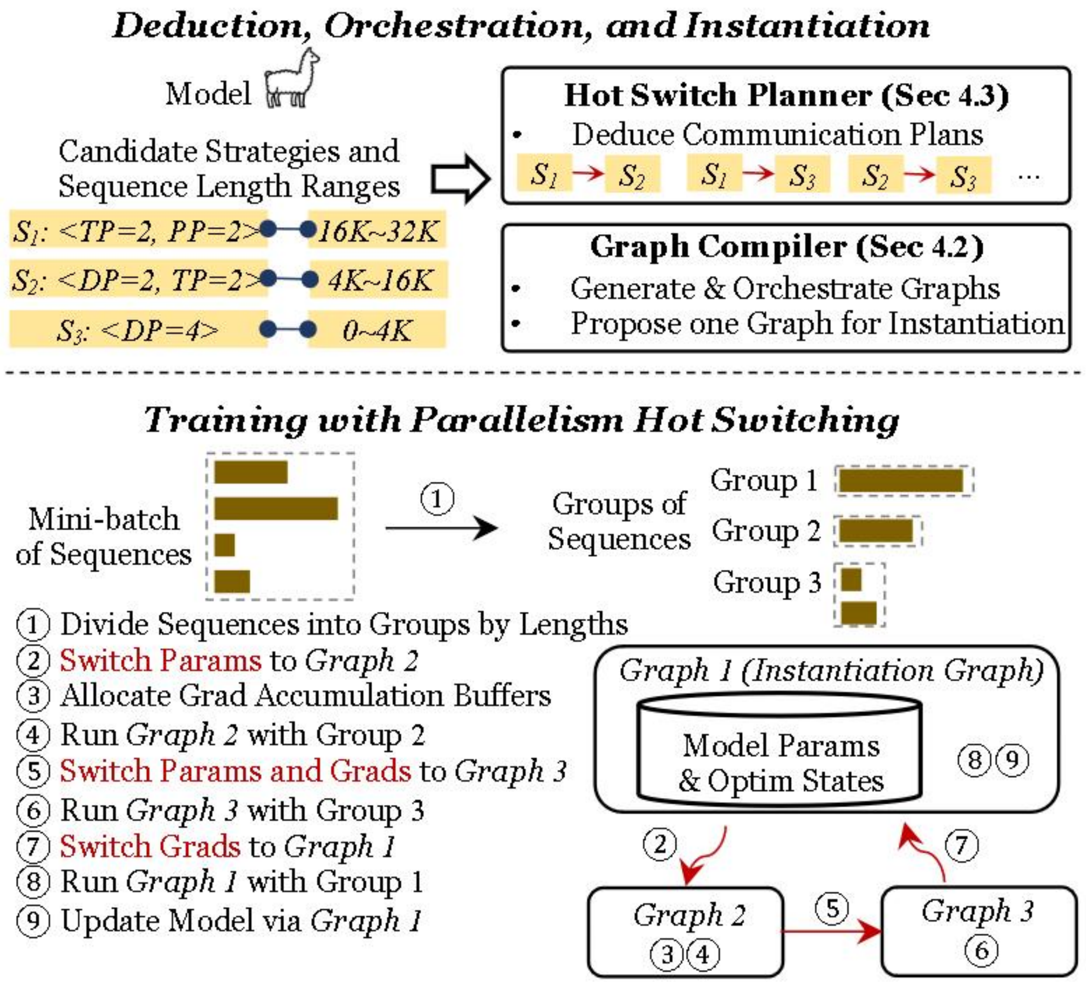
Two ways to achieve load balancing

1. **Request Scheduling**
   Dispatching request evenly
   to different workers.

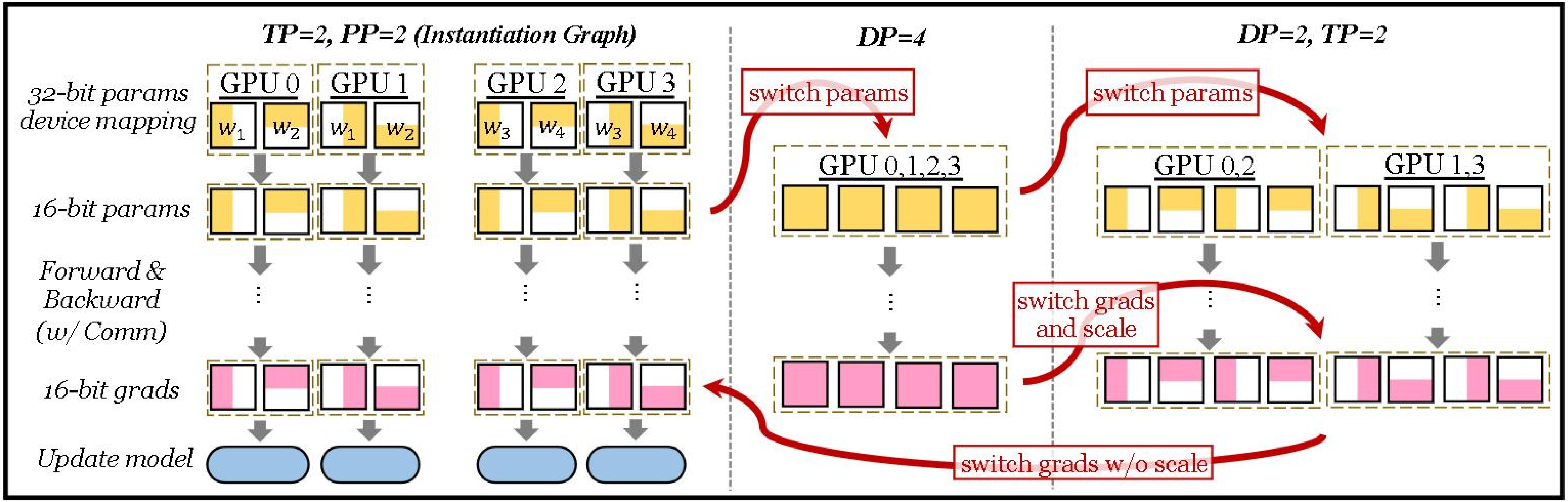
2. **Resource Re-orchestration**

Re-allocate unused resources.

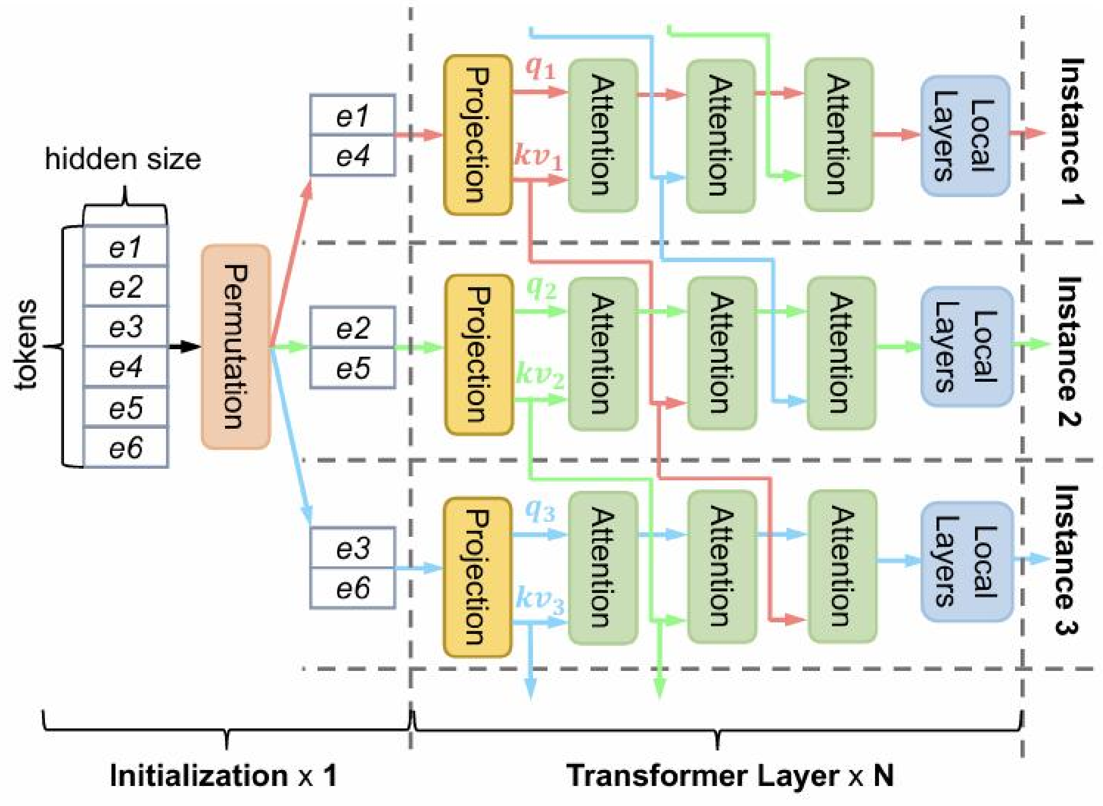
Hot-switching (SOSP'24)



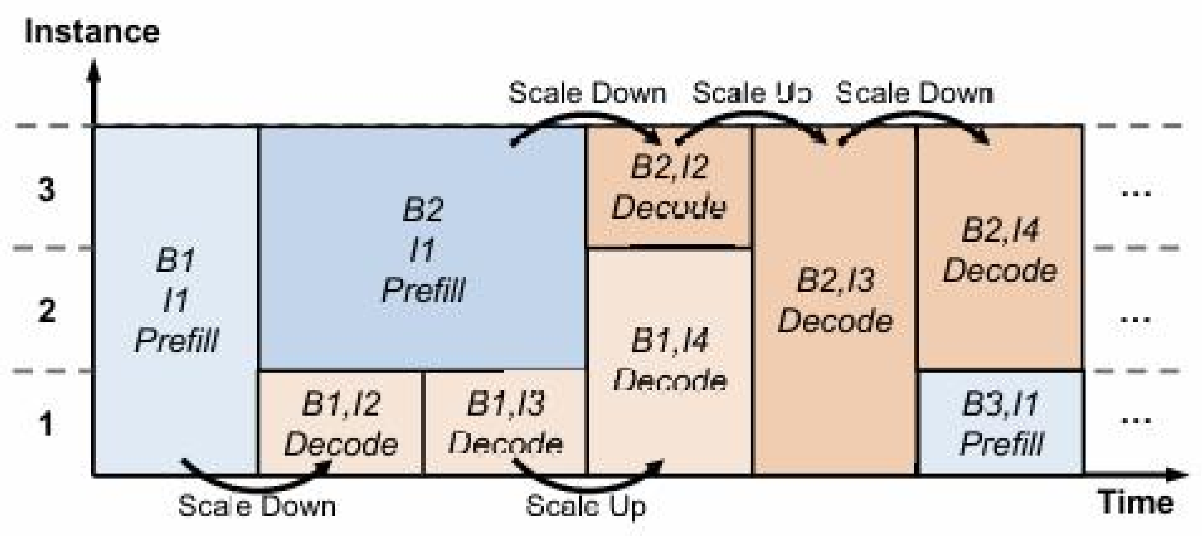
Hot-switching (SOSP'24)



LoongServe (SOSP'24)



Elastic Sequence Parallelism



Runtime Example

DynamoLLM (HPCA'24 Best Paper)

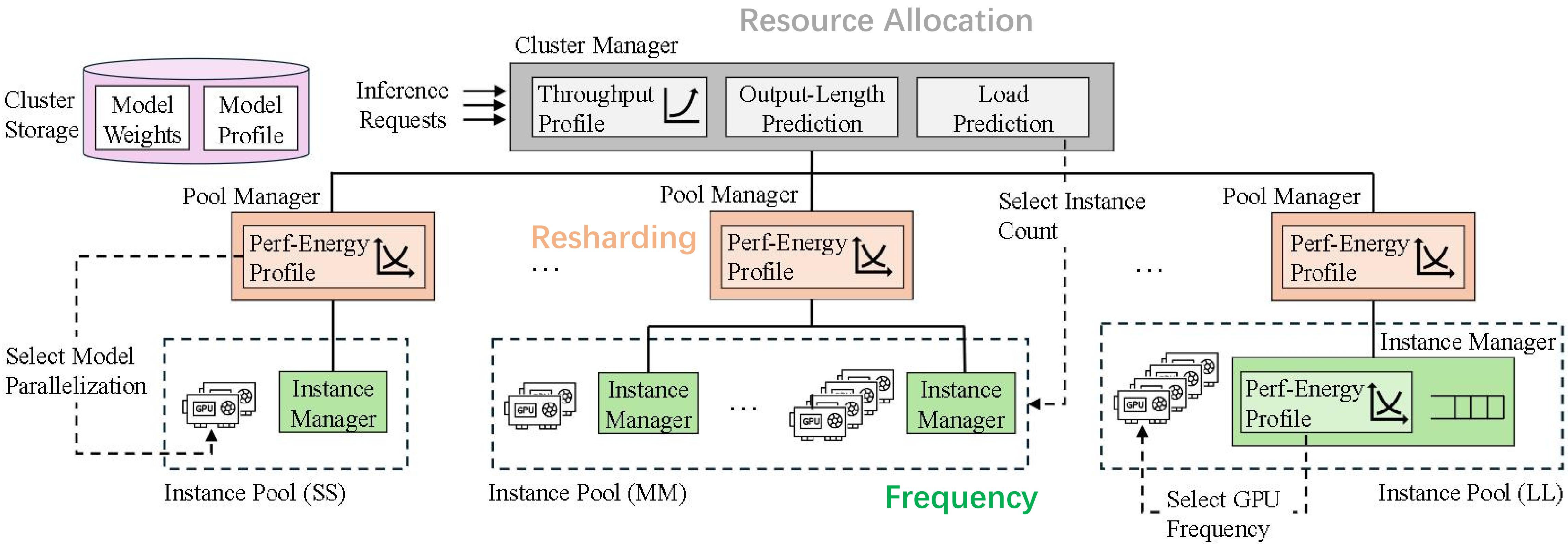


Fig. 4: DynamoLLM architecture: a hierarchy of controllers with cluster resources split into per request-type pools.

DynamoLLM (HPCA'24 Best Paper)

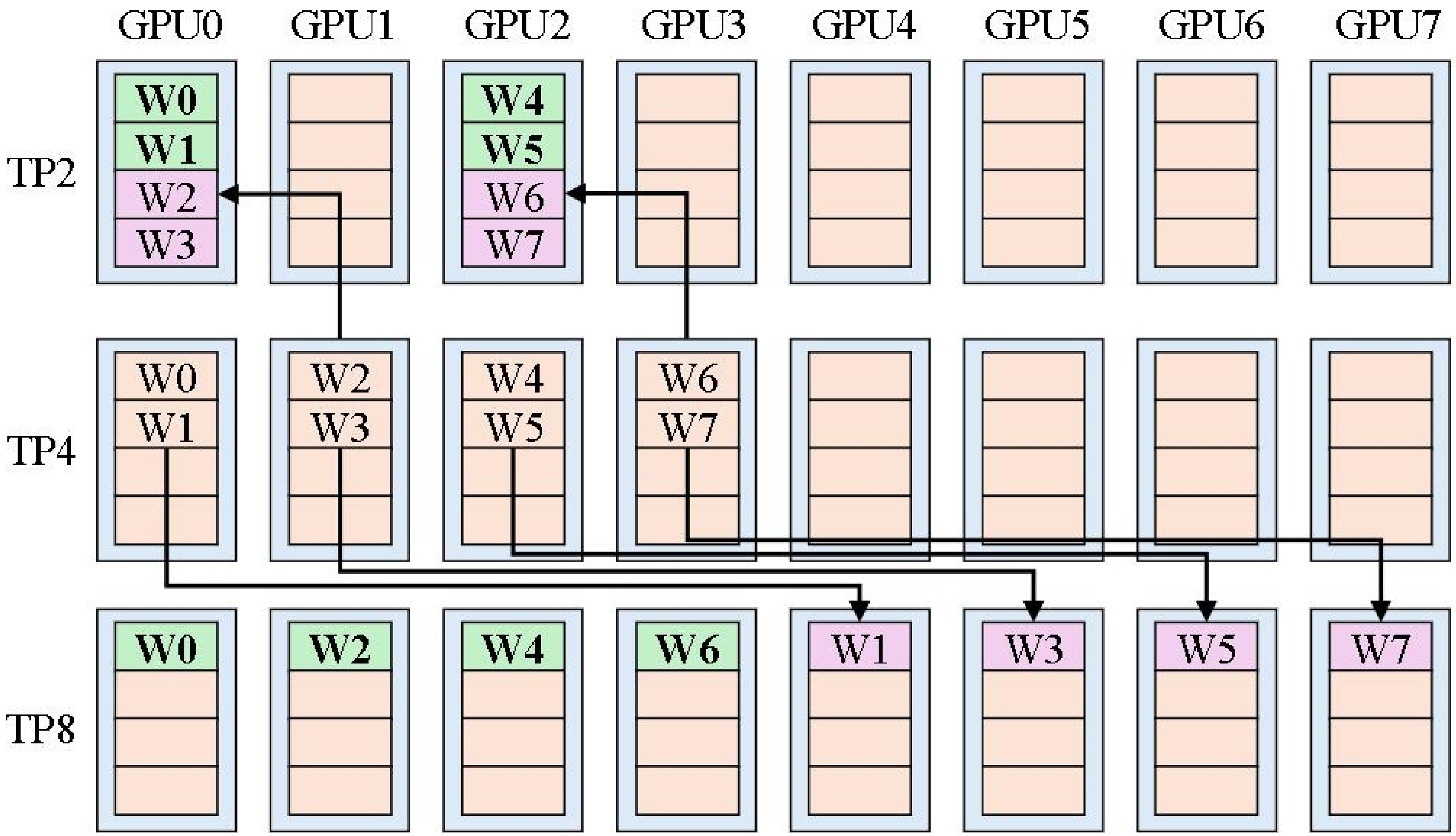
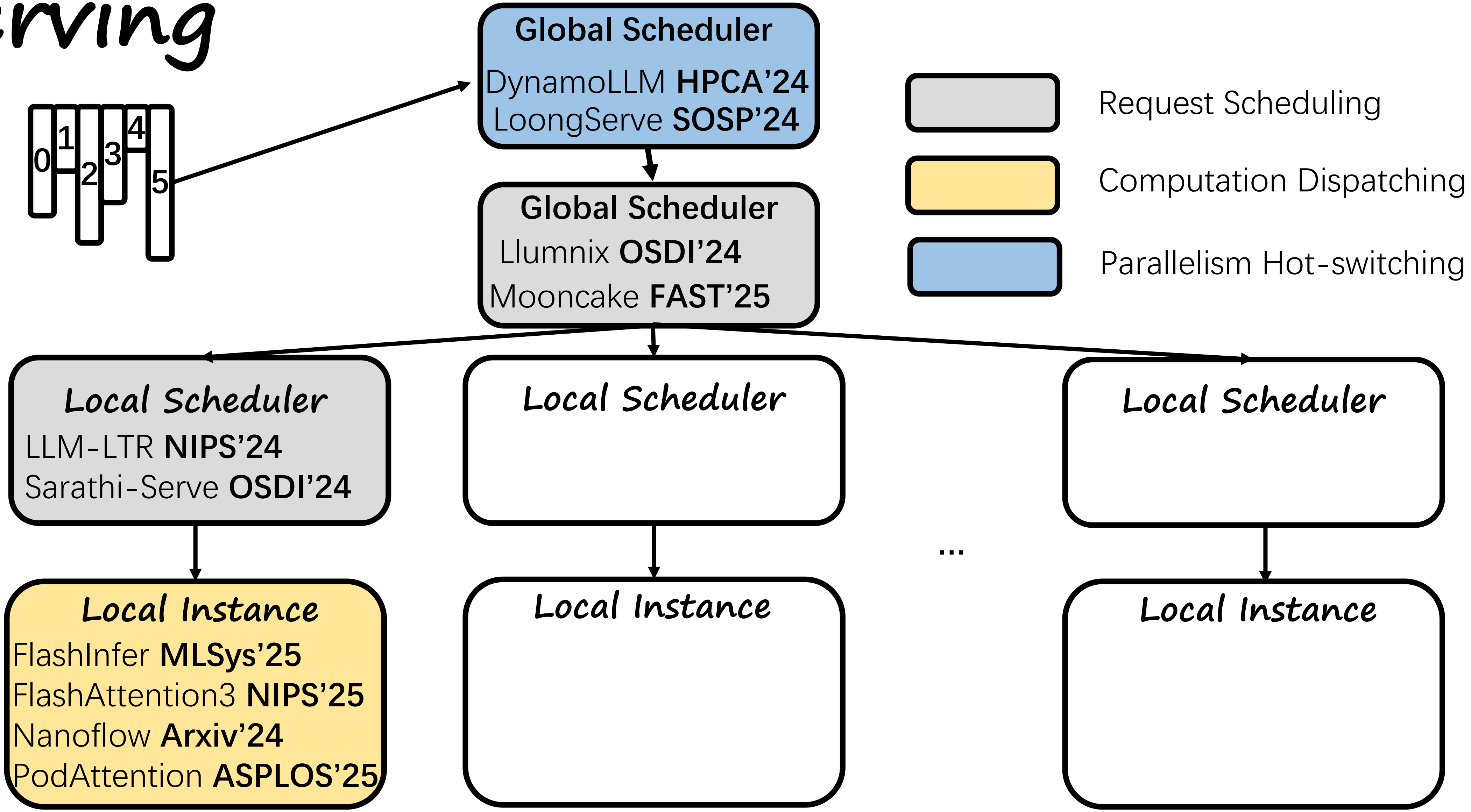


Fig. 5: Example of re-sharding a TP4 model to TP2/TP8.

Discussion: Load Balancing in LLM Serving



Root Cause: Request Length Discrepancy

	Batching	Computation	Objective	
LLM Training	Static	Prefill	Throughput	
LLM Serving	Continuous	Prefill + Decode	SLO	