WLB-LLM: Workload-Balanced 4D Parallelism for Large Language Model Training

Zheng Wang, Anna Cai, Xinfeng Xie, Zaifeng Pan, Yue Guan, Weiwei Chu, Jie Wang, Shikai Li, Jianyu Huang, Chris Cai, Yuchen Hao, Yufei Ding

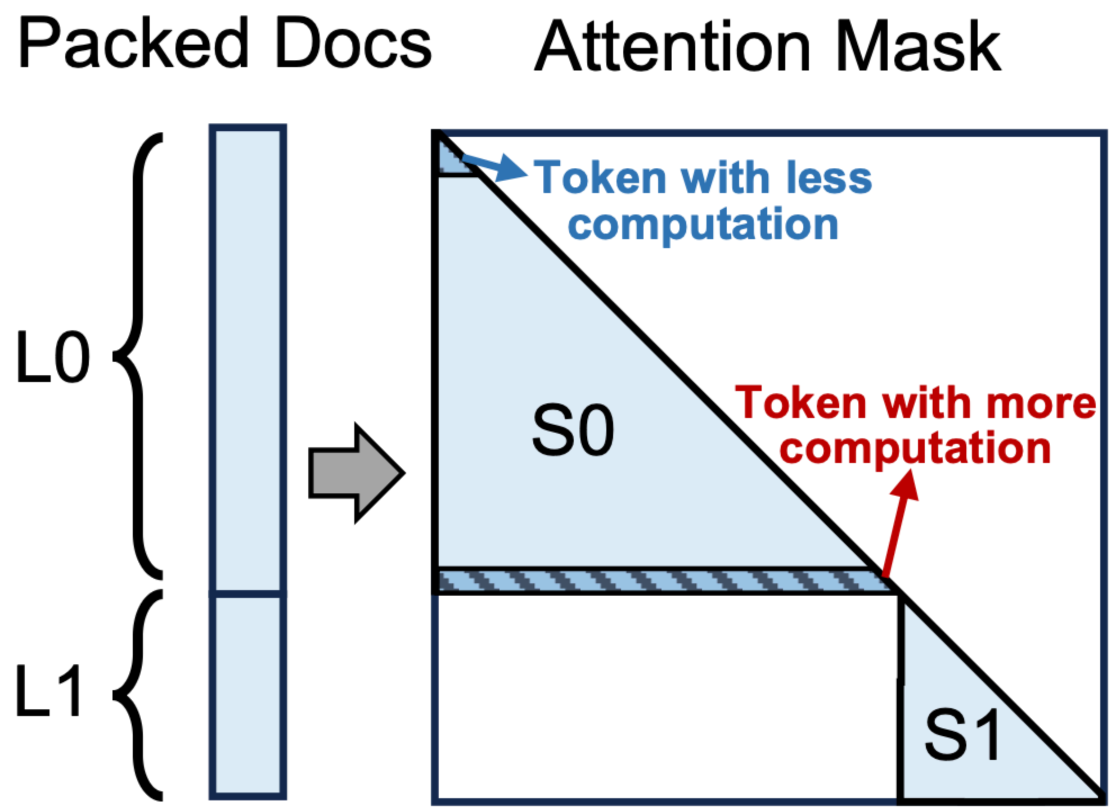
Kartik Ramesh

Introduction

- Llama 3.1 405B trained over 16k GPUs for 30M H100 hours.
- Using AWS H100 pricing **$212M**.
- A 1.2x increase in training speed would have saved **$36M** in cloud costs.
- WLLB-LLM is a work from Meta and UCSD, released ~6 months after their training Llama 3 that achieves this.

Self Attention

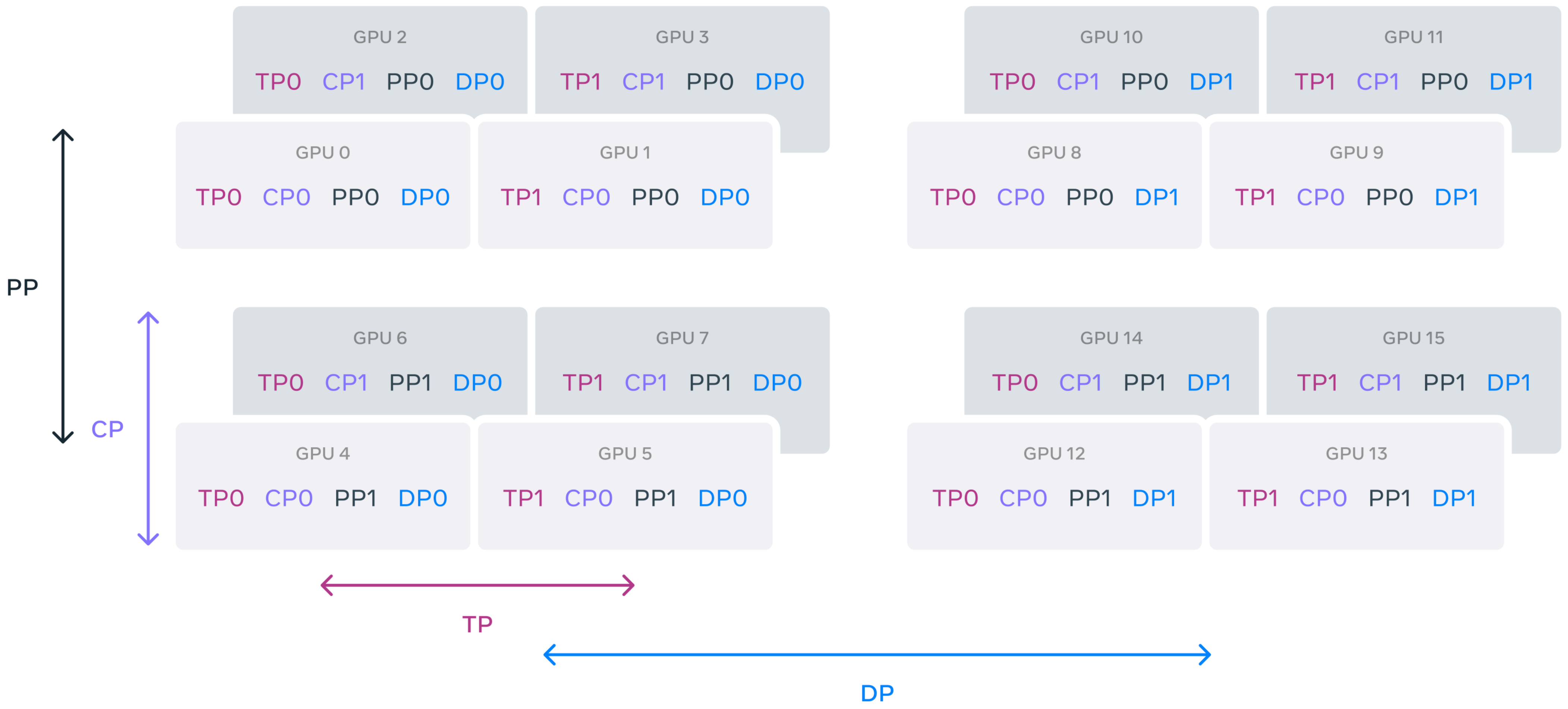
- Attention scales quadratically with the number of past tokens $O(T^2)$
- Not all tokens are equal: Processing later tokens in a document require more computation than earlier.
- When packing multiple documents for training, control attention span using masks.



4D Parallelism

- Huge models are trained on huge clusters because of scale and necessity.
- Tensor shape $[B, T, H]$ across several layers.
- 4D Parallelism splits this shape in various ways.
    - Data - Split B across replicated model.
    - Pipeline - Split model into groups of layers.
    - Context - Split along sequence length T
    - Tensor - Split across the H.

4D Parallelism



$$DP > PP > CP > TP$$

Key Takeaways

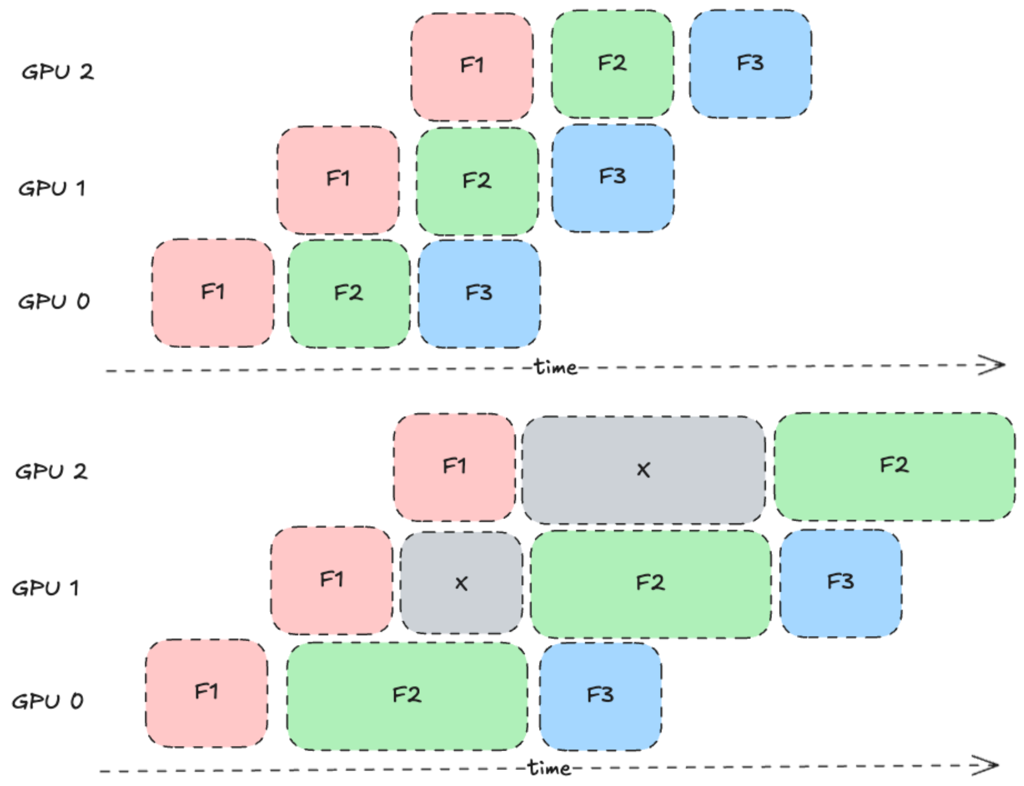
Problem: In long-context 4D training, token count is a weak proxy for compute; attention cost is highly non-uniform.

- **Core idea #1 (PP)**: Reduce PP imbalance via attention-aware micro-batch packing.
- **Core idea #2 (CP)**: Reduce CP imbalance via fine-grained per-document sharding.

Bottom line: WLB-LLM improves training throughput ($\approx 1.23x$) without hurting convergence.

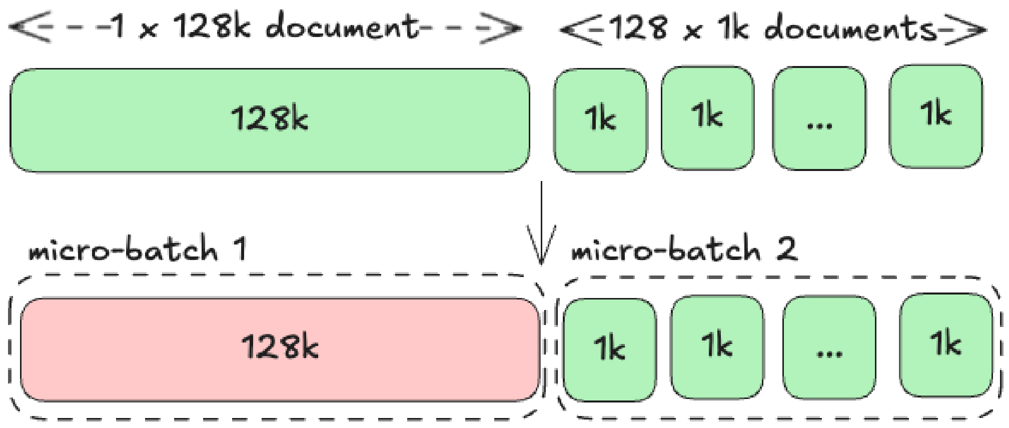
Motivation #1 - Pipeline Imbalance

- Pipeline Parallelism splits B into N μB to hide delays.
- Balanced Batches $\rightarrow$ Higher throughput



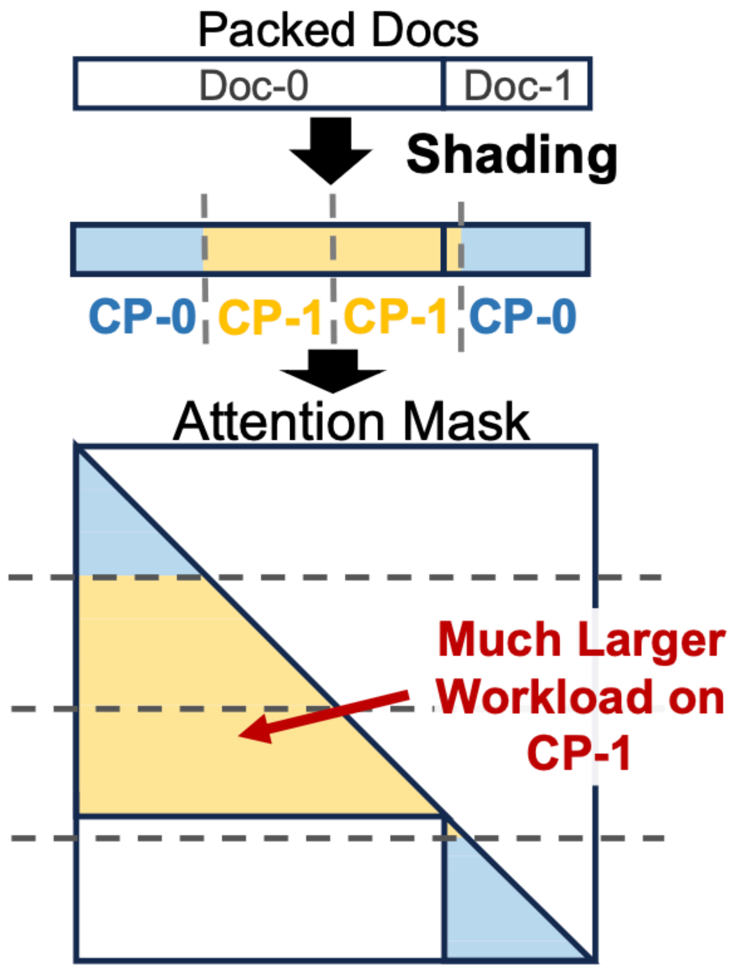
Motivation #1 - Pipeline Imbalance

- Naive approach $\longrightarrow$ split by tokens.
- Each μB has uniform sequence length = `CONTEXT_WINDOW_SIZE`
- Does this balance LLM workload?
- $k.128^2 >> k.128 * 1^2$



Motivation #2 - Context Imbalance

- Workload across CP workers should be balanced.
- After packing, split the sequence into $2.CP$ parts and assign one from front and one from back to balance attention.
- Good heuristic, fails for multiple packed documents. Common in long context training.
- Every small delay adds up to higher-order delays.



Baseline: Attention-Aware packing.

Idea: Divide B into μB by estimating d_i^2 as attention cost for each document.

- It works, but limited balancing improvements $\rightarrow$ limited speedup.
- Higher balancing across μB requires balancing across multiple global B. This disturbs the random order of training and loss convergence.
- It might be impossible to come up with such a μB construction, if there are no candidates.

$$\begin{aligned} \text{minimize} \quad & \max(\boxed{\sum_{i=1}^{N} x_{ij} \cdot d_i^2)}, \; j = 1, \cdots, M \\ \text{subject to} \quad & \sum_{j=1}^{M} x_{ij} = 1, \qquad i = 1, \cdots, N \\ & \sum_{i=1}^{N} x_{ij} \cdot d_i \leq L, \qquad j = 1, \cdots, M \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

Attention Cost

Each Document is present in only one batch

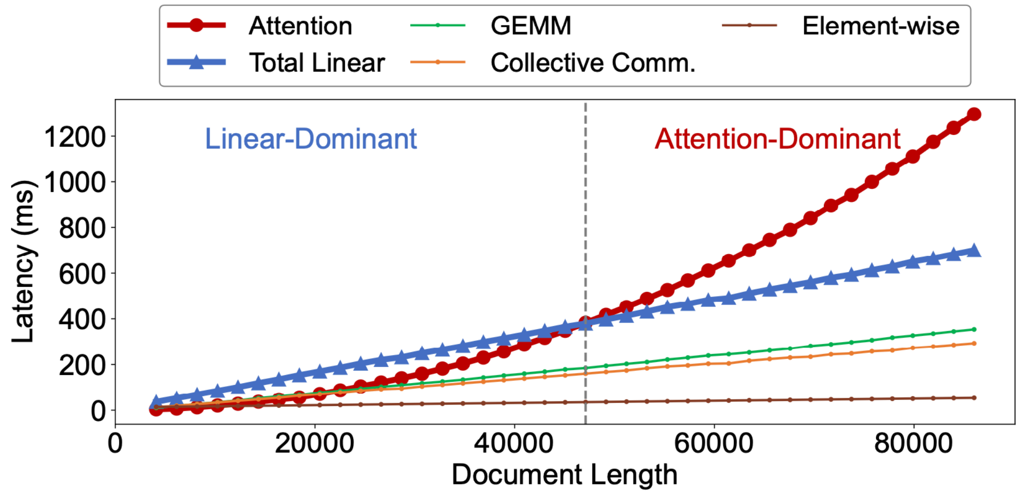
L = Context Size

Variable-Length Packing

Idea: Allow $len(\mu B) > \texttt{CONTEXT\_SIZE}$ for weaker μB

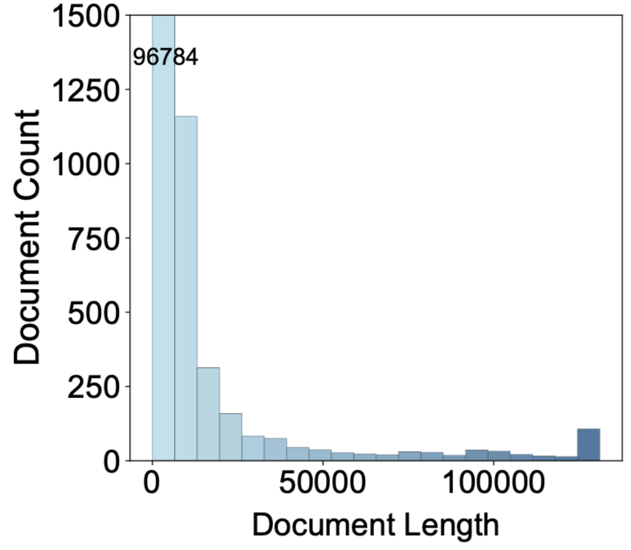
- Attention is Quadratic, but other operations are linear (feed-forward, comms etc.)
- Balance the total workload, not just attention.
- Balance long documents against many shorter documents.

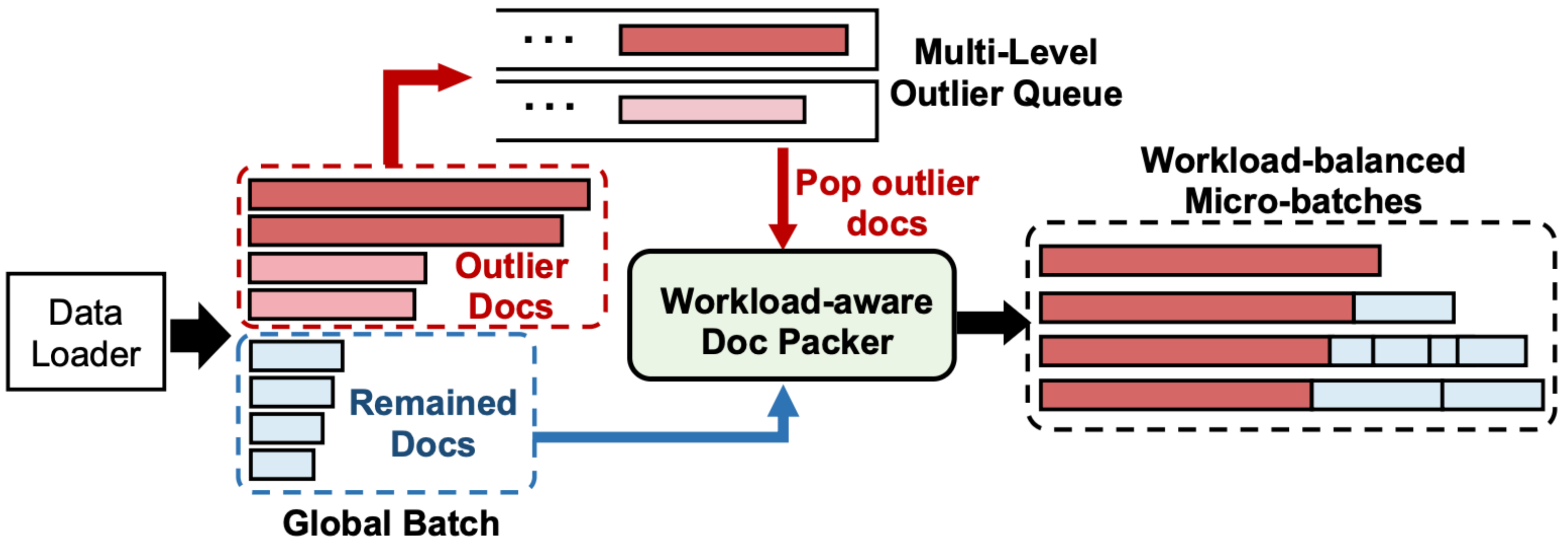
$$\min\left(\max\left(\sum_{i=1}^{N}\left(W_a(x_{ij}\cdot d_i)+W_l(x_{ij}\cdot d_i)\right)\right)\right)$$



Outlier document detection

- Still, you might not have sufficient smaller documents to balance the load of a long document.
- Observe: there aren't that many ultra-long documents.
- Instead of balancing across multiple batches, delay the few long documents.
- Model convergence should not hurt significantly.

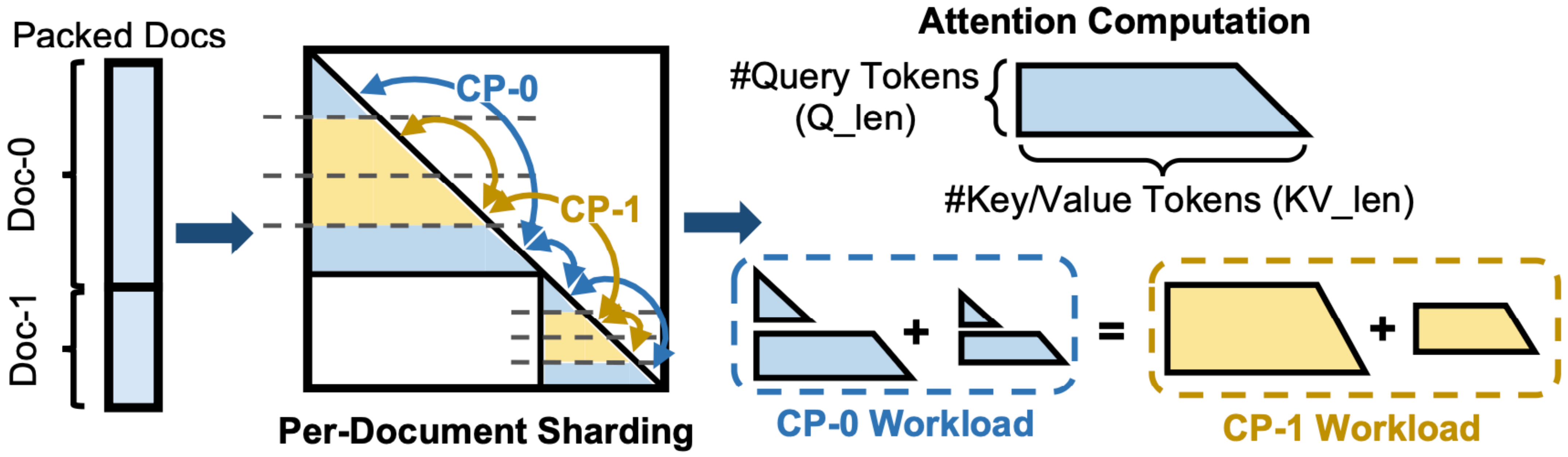



Multi-Level Outlier Queue
Pop outlier docs
Workload-balanced Micro-batches
Outlier Docs
Data Loader
Workload-aware Doc Packer
Remained Docs
Global Batch

Improved CP sharding

Idea: Apply CP indexing logic to each individual document.

- This should yield a more balanced workload across multiple CP workers.
- They also implement an optimization to avoid padding tokens.

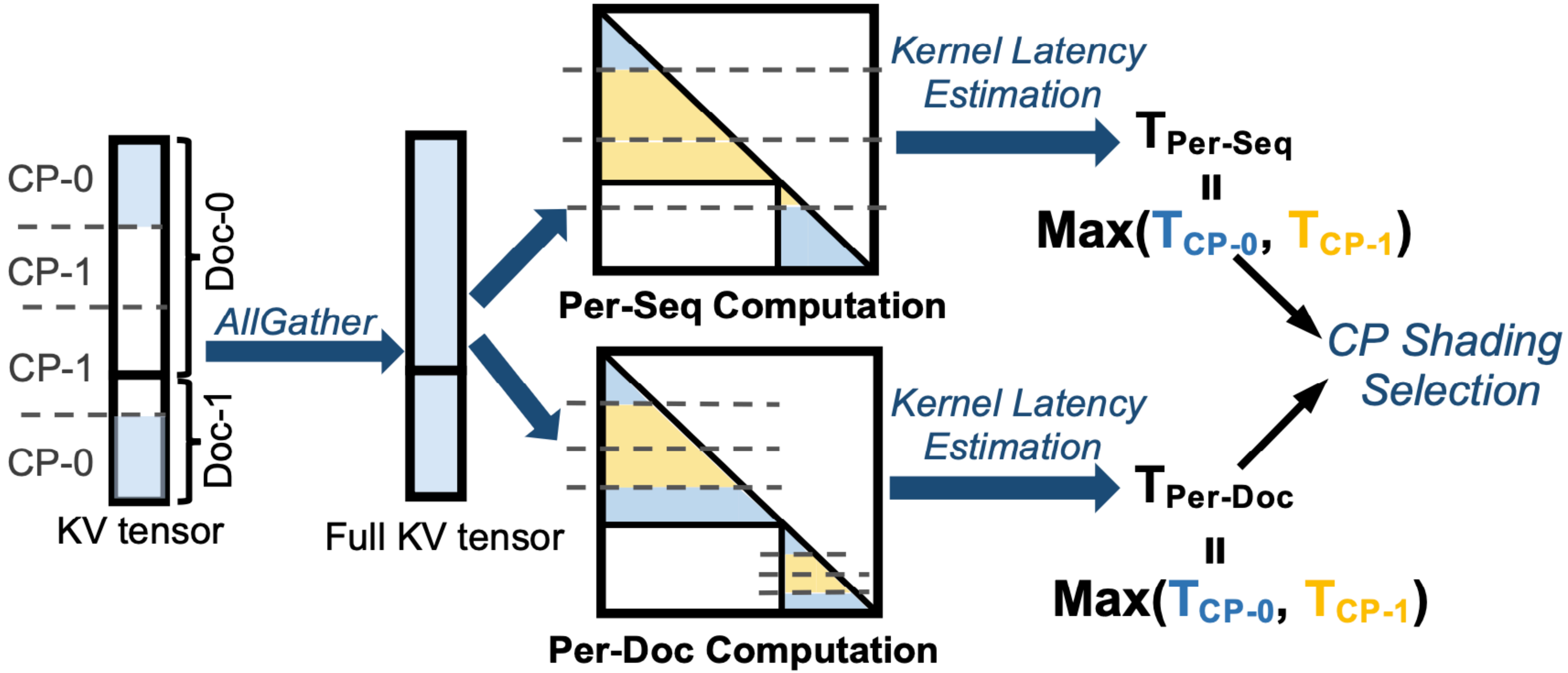


Kernel inefficiencies

- Per-Document sharding achieves better balance, but it does not always guarantee better performance.
- Smaller per-rank attention problems reduce kernel efficiency:
    - Poor tile utilization → padding overhead for short sequences (<128 tokens).
    - Lower effective FLOPs utilization → higher time per token.
    - Reduced KV tile reuse → weaker Hopper TMA multicast benefits

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Kernel Inefficiencies



Experimental Setup

- **Cluster**: 32 nodes, each with 8x NVIDIA H100 SXM 80GB GPUs.
- **Interconnect**: NVLink intra-node, RoCE inter-node.
- **Models**: LLaMA-like 550M, 7B, 30B, 70B; each tested at 64K and 128K context.
- **Training config**: 4D parallelism, global batch size = `PP_size x DP_size`, `bfloat16` precision.
- **Baselines**:
  - `Plain-4D`: default 4D training with per-sequence CP sharding.
  - `Fixed-4D`: fixed-length packing + fixed CP sharding (per-sequence or per-document).

Speedup Breakdown

Which optimization helps us the most?

- `PP-Var-Len` alone $\rightarrow$ 1.28x
- Orthogonal optimizations that combine well.
- Every second counts!

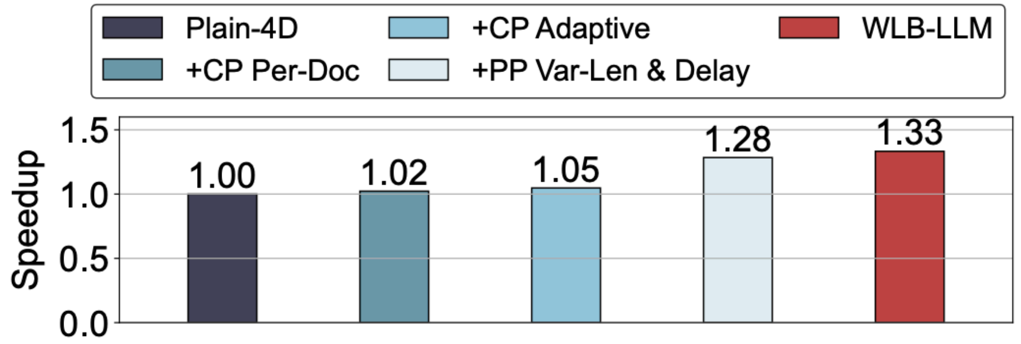


Figure 13: Performance breakdown of *WLB-LLM* on the 7B model with a 128K context window.

Speedup across Model + Context

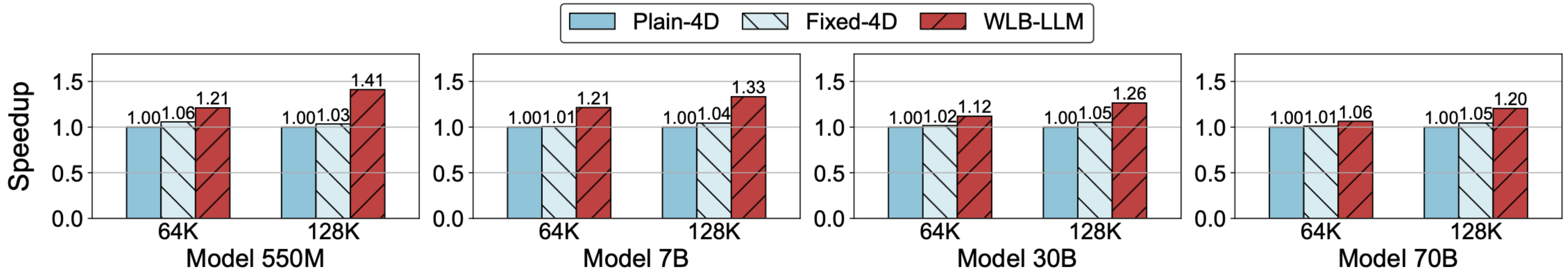


Figure 12: Training performance speedups of *WLB-LLM* and *Fixed-4D* over *Plain-4D* across various configurations.

- WLB-LLM consistently outperforms baseline for all tested configurations.
- Naive attention balancing is insufficient.
- Relative speedup decreases with increased model size.

Other Experiments (Summary)

- **Context sensitivity (Fig. 14)**: Speedup increases with context length (about `1.07x @64K` to `1.40x @160K` on 7B), consistent with worse imbalance at longer contexts.
- **Packing overhead vs balance (Table 2)**: WLB-LLM reaches near-optimal imbalance with low runtime overhead (tens of ms), while solver-based packing can be prohibitively slow.
- **CP sharding ablation (Fig. 15)**: Adaptive sharding consistently outperforms always-per-sequence or always-per-document sharding.
- **Convergence / quality**: Their training-loss curves indicate no clear quality regression from the system optimizations.

Discussion & Critique

Strengths

- Identifies and fixes a bottleneck for training long-context Llama models with 4D parallelism.
- Joint PP+CP optimization gives meaningful real-system gains, especially as context windows grow.
- Engineering is practical: low overhead and no obvious convergence regression in their reported runs.

Weaknesses

- Workload dependence and limited generalization evidence outside their evaluated distribution.
- Strong dependence on a small number of extreme outliers; unclear benefit when length distributions are flatter.
- Heavy use of heuristics (packing + sharding selection) without strong guarantees in worst-case settings.

Related Work

(1) Efficient Long-context Language Model Training by Core Attention Disaggregation (DistCA)

- Split out “core attention” ($\mathrm{softmax}(QK^\top)V$) as a weightless compute service, separate from the rest of the transformer.
- Better than WLB-style baselines at scale: reports ~1.15–1.35× throughput gains over their WLB “ideal” baseline in 4D (with PP), depending on workload.

(2) ByteScale Efficient Scaling of LLM Training with a 2048K Context Length on More Than 12,000 GPUs

- Hybrid Data Parallelism (HDP): unify DP + CP into one dynamic device mesh.
- Length-aware sharding: use the minimum number of devices per sequence.
- Short sequences stay local (skip CP comm), long sequences shard across more GPUs.

(3) Ordering efficiency

- Reduce pipeline bubbles by optimizing scheduling.
- PipeDream, 1F1B, Seq1F1B.

What did you think?

Problem: In long-context 4D training, token count is a weak proxy for compute; attention cost is highly non-uniform.

- **Core idea #1 (PP)**: Reduce PP imbalance via attention-aware micro-batch packing.
- **Core idea #2 (CP)**: Reduce CP imbalance via fine-grained per-document sharding.

Bottom line: WLB-LLM improves training throughput ($\approx 1.23x$) without hurting convergence.

Packing Method		Imbalance Degree	Packing Overhead (ms)
Method	Config		
Original Packing	/	1.44	0
Fixed-Len Greedy	#global batch=1	1.41	4
	#global batch=2	1.22	5
	#global batch=4	1.11	5
	#global batch=8	1.08	5
Fixed-Len Solver	#global batch=1	1.40	467
	#global batch=2	1.18	1488
	#global batch=4	1.09	25313
WLB-LLM	#queue=1	1.24	8
	#queue=2	1.05	20
	#queue=3	1.05	23

Table 2: Packing imbalance degree and overhead analysis.