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

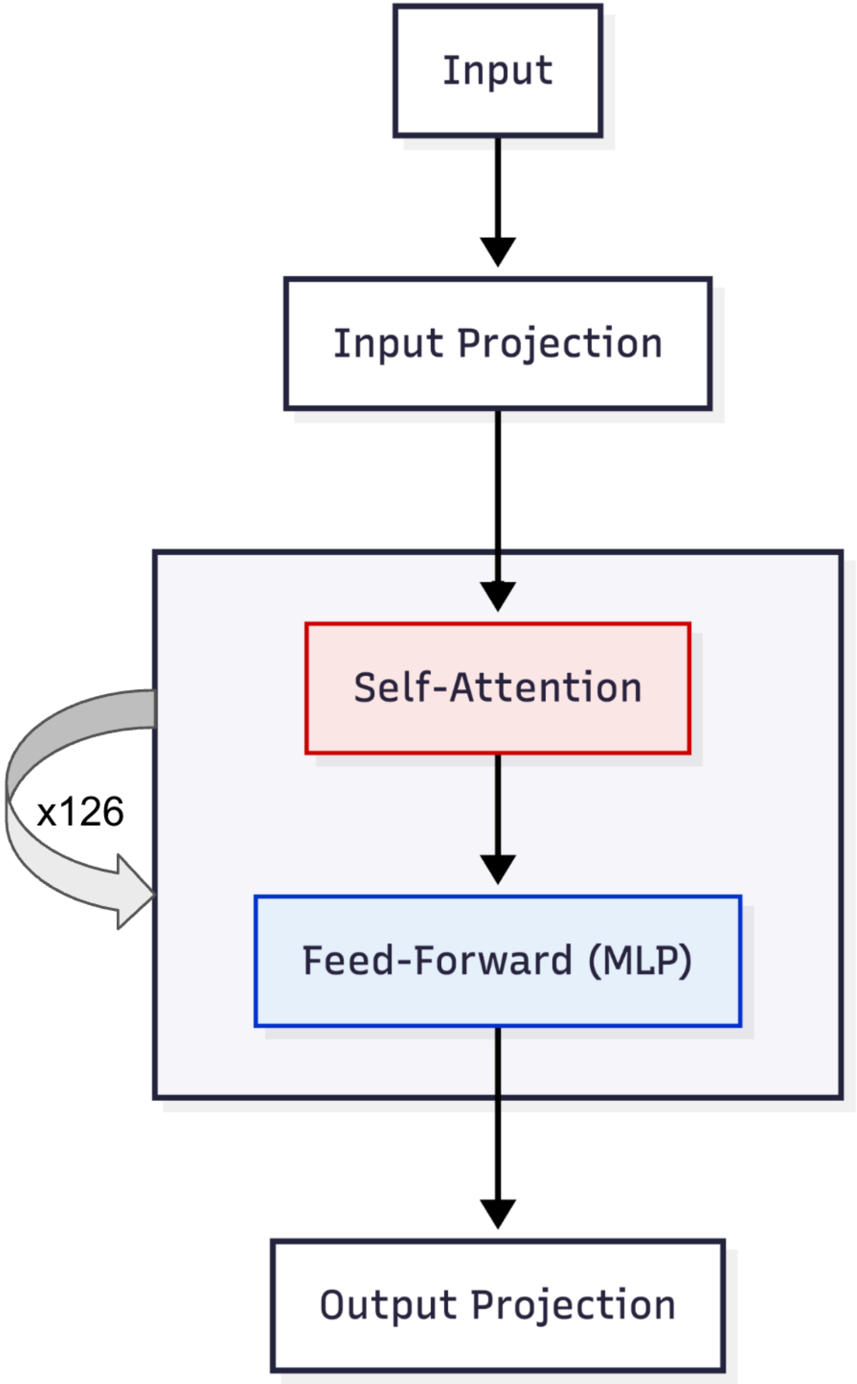
Introduction

- 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.
- WLLM is a work from Meta and UCSD, released ~6 months after their training Llama 3 that achieves this.

Background - Model Architecture

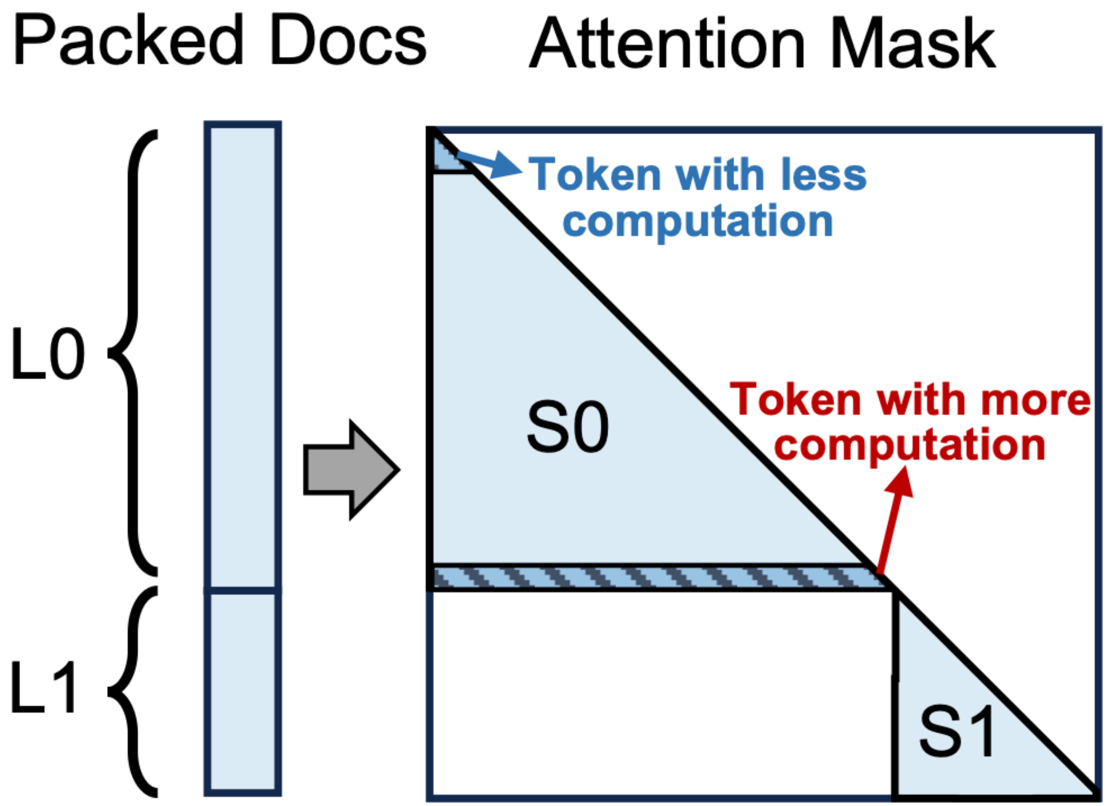
Things to keep in mind:

- Really big models, 405B = 910GB with 126 transformer blocks.
- Self-Attention, a layer that enriches tokens in a sequence with other token information.
- Feed-Forward, a linear matrix multiplication layer.



Self Attention

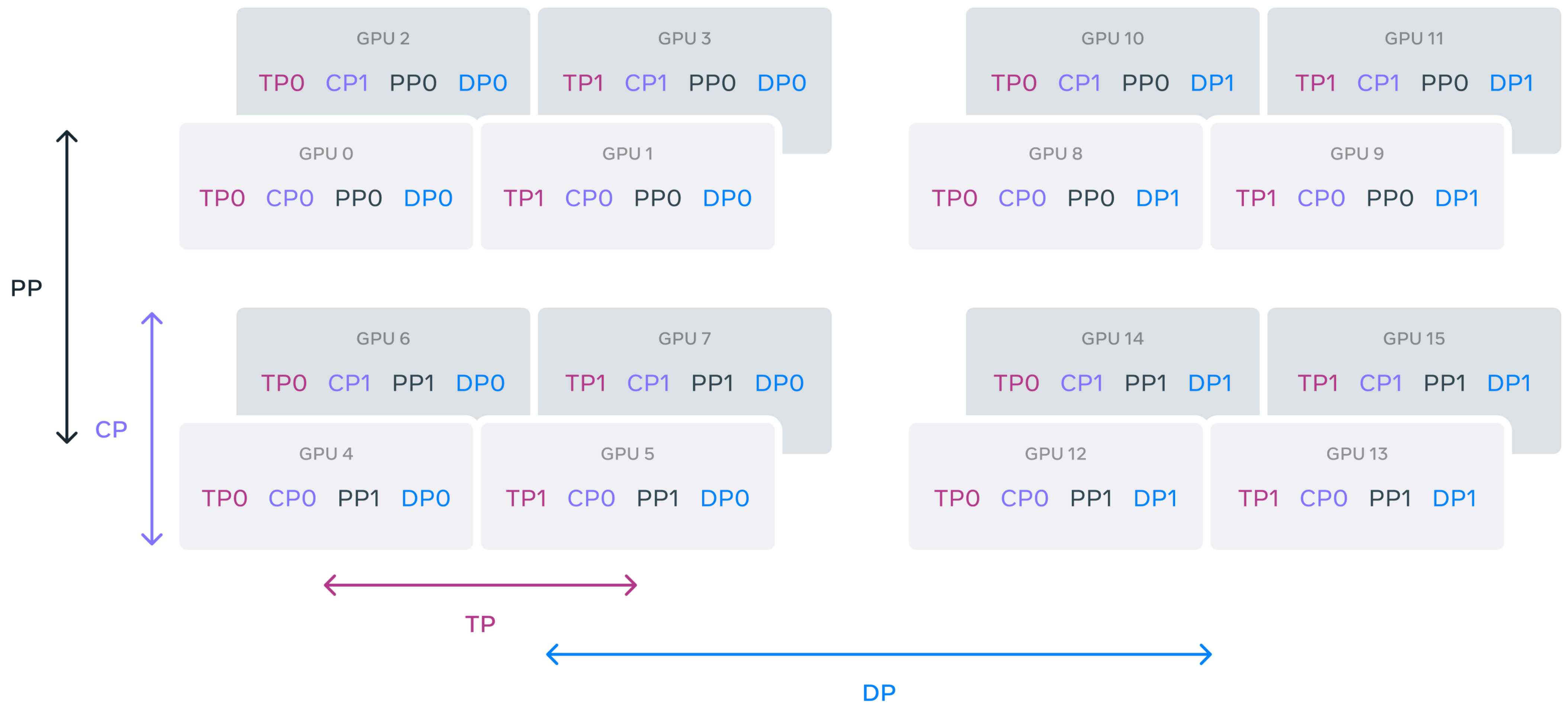
- Attention scales quadratically with the number of past tokens $O(T^2)$
- Not all tokens are equal: Processing later tokens in a document require more computation than earlier.
- When packing multiple documents for training, control attention span using masks.



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.

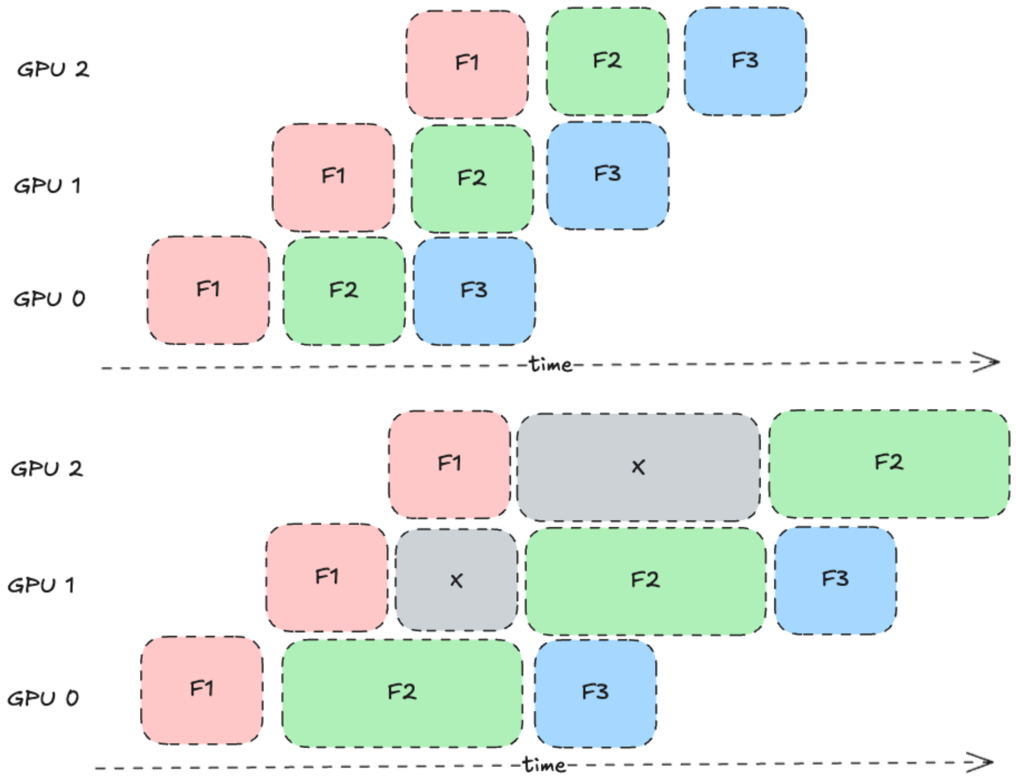
4D Parallelism



$$DP > PP > CP > TP$$

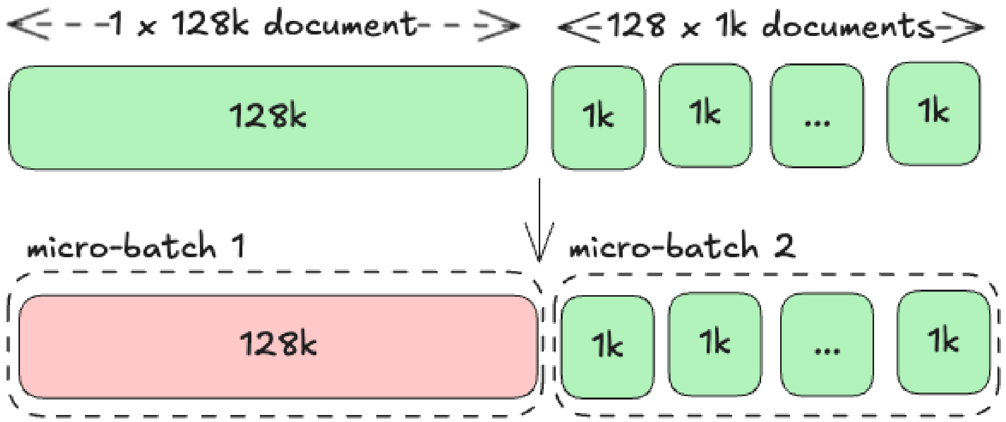
Motivation #1 - Pipeline Imbalance

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



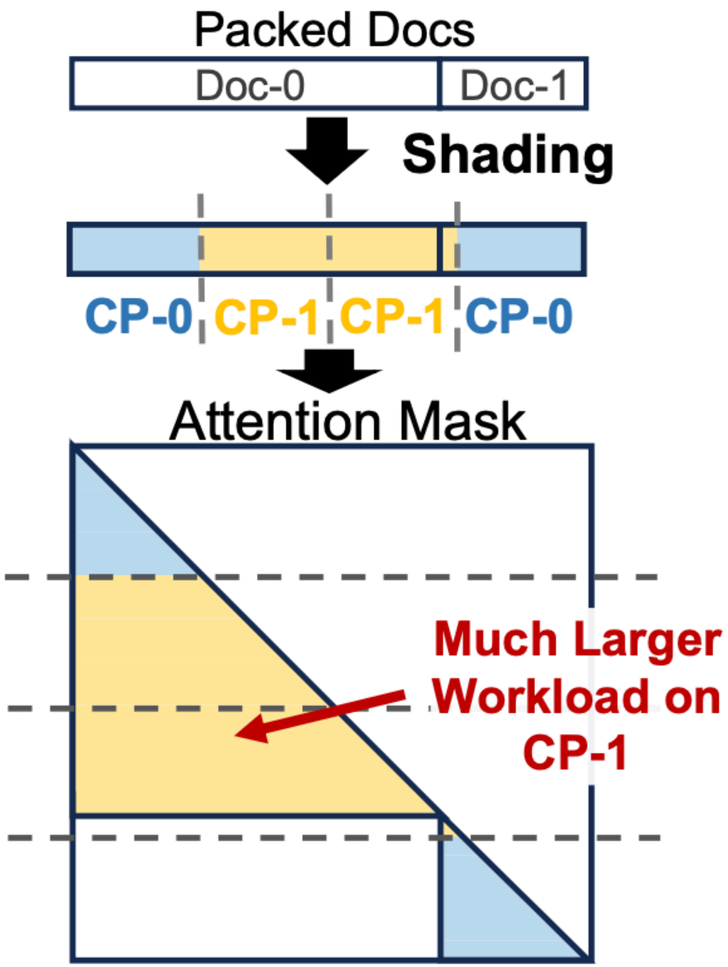
Motivation #1 - Pipeline Imbalance

- Naive approach $\rightarrow$ split by tokens.
- Each μB has uniform sequence length = `CONTEXT_WINDOW_SIZE`
- Does this lead to equal attention?
- $k.128^2 >> k.128 * 1^2$



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.
- Not a huge problem, but every millisecond adds up!



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 improvemnts $\rightarrow$ 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.

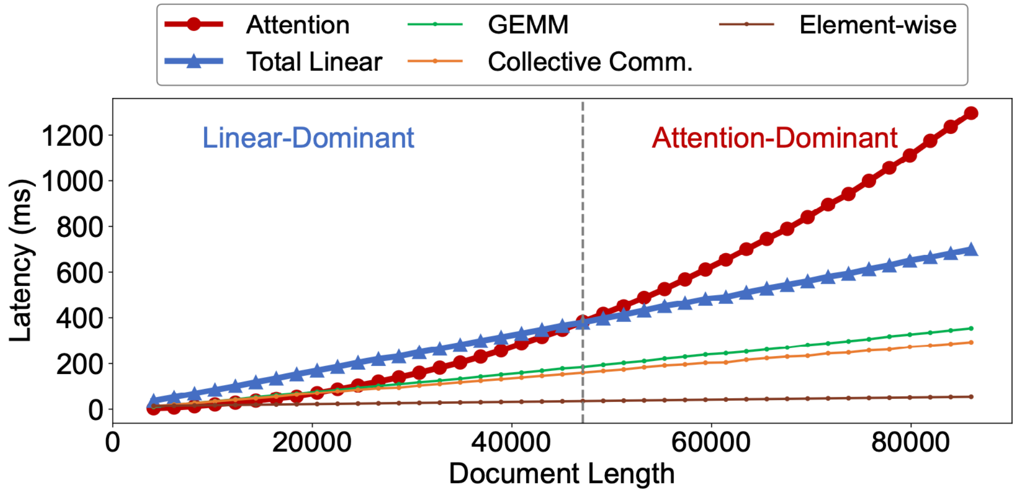
$$
\begin{aligned}
\text{minimize} \quad & \max\left(\sum_{i=1}^{N} x_{ij} \cdot d_i^2\right), \; j = 1, \cdots, M \\
\text{subject to} \quad & \sum_{j=1}^{M} x_{ij} = 1, \qquad\qquad i = 1, \cdots, N \\
& \sum_{i=1}^{N} x_{ij} \cdot d_i \leq L, \qquad j = 1, \cdots, M \\
& x_{ij} \in \{0, 1\}
\end{aligned}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

Variable-Length Packing

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

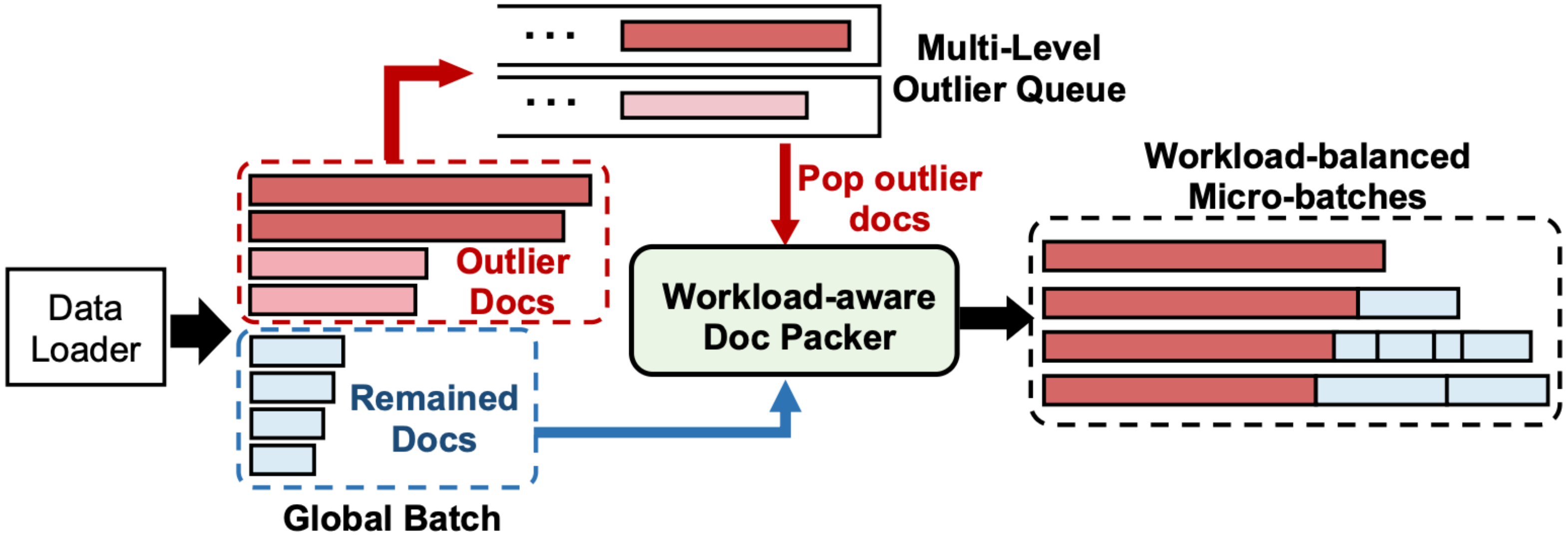
- Attention is Quadratic, but other operations are linear (feed-forward, comms etc.)
- Balance the total workload, not just attention.
- Balance long documents against many shorter documents.

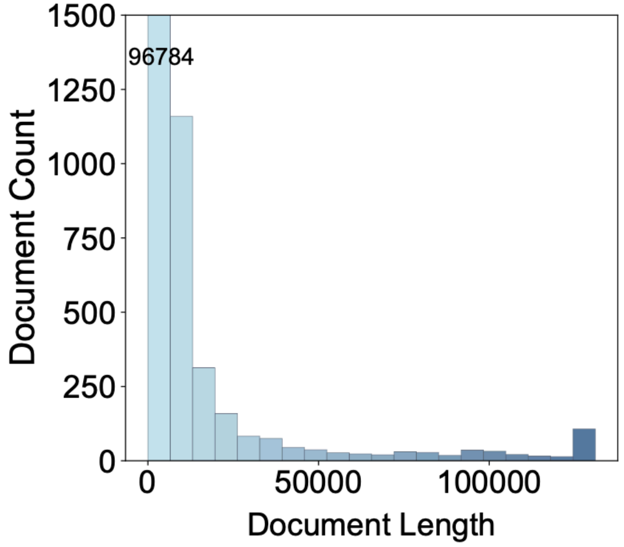
$$\min\left(\max\left(\textstyle\sum_{i=1}^{N}\left(W_a(x_{ij}\cdot d_i)+W_l(x_{ij}\cdot d_i)\right)\right)\right)$$



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.





Main Algorithm

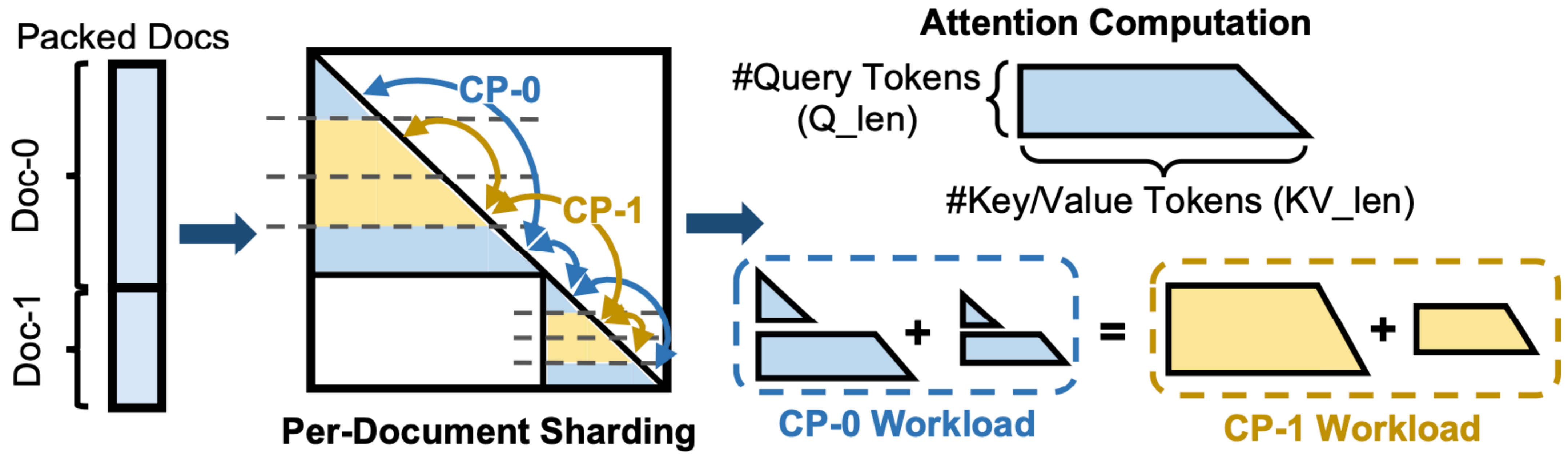
Given a DP bach

- construct a document set by removing the outlier documents, filling from the queue, and rolling over any documents from the previous iteration.
- now you effectively have a set of documents that you solved for using an ILP in S3.3, however they claim that it would take too long.
- so they instead pack greedily into an array of N micro-batches. sort the documents in reverse by length, and for each document:
    - find the least loaded microbatch (first by load, next by length)
    - if it can fit this current document, great.
    - else reserve this document for a later stage.

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.

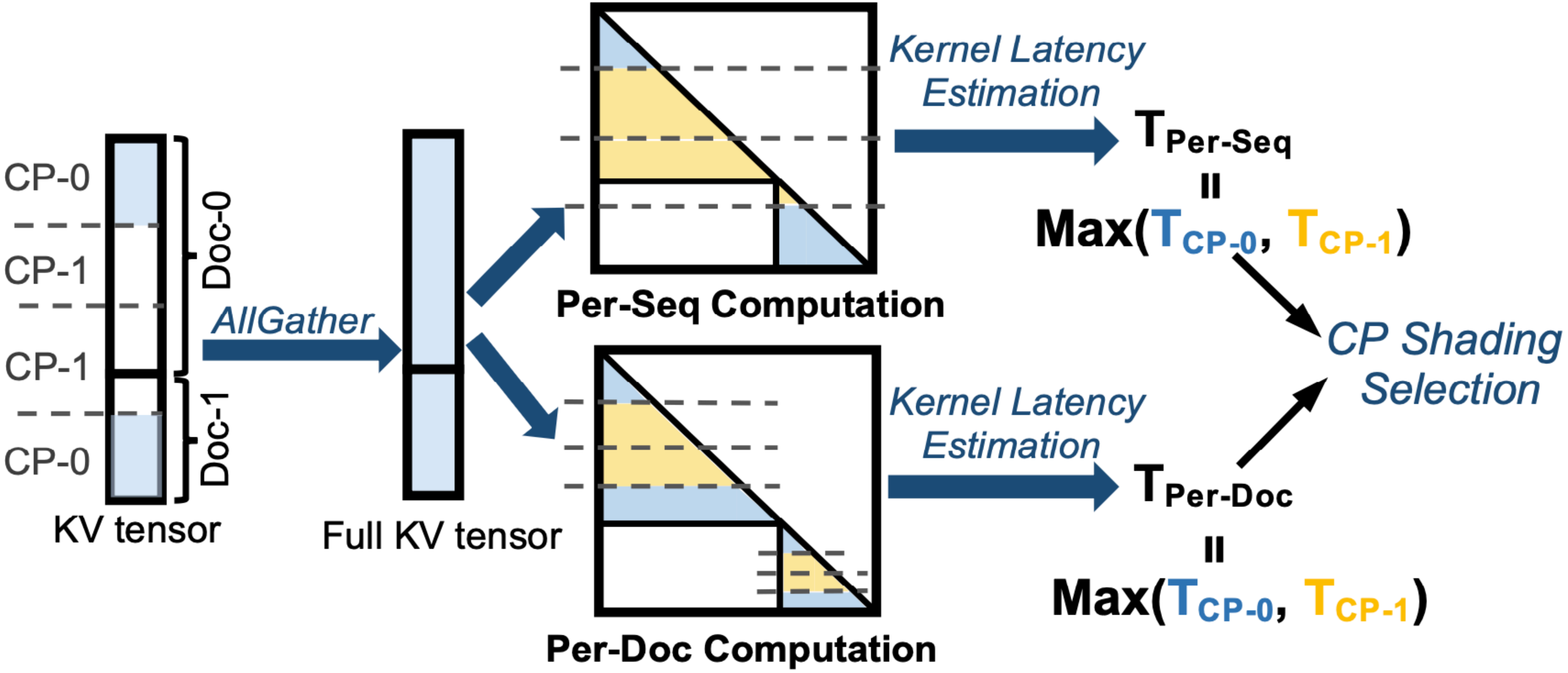


Kernel inefficiencies

- Per-Document sharding achieves better balance, but it does not guarantee better performance.
- The attention kernel might become less efficient for smaller documents.
- Sharding lowers the Q dimension, which might cause:
  - Longer kernel time due to redundant computations (padding < 128 tokens)
  - Decreased throughput (unable to leverage TMA)

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

Kernel Inefficiences



Evaluation

- their contributions are in PP and CP balancing. Let's think, when would PP and CP imbalance hurt most?
  - CP: longer context, more chance of long document being assigned to imbalanced worker.
  - PP: larger model, micro-batch imbalance adds more to the latency.

Speedup Breakdown

Which optimization helps us the most?

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

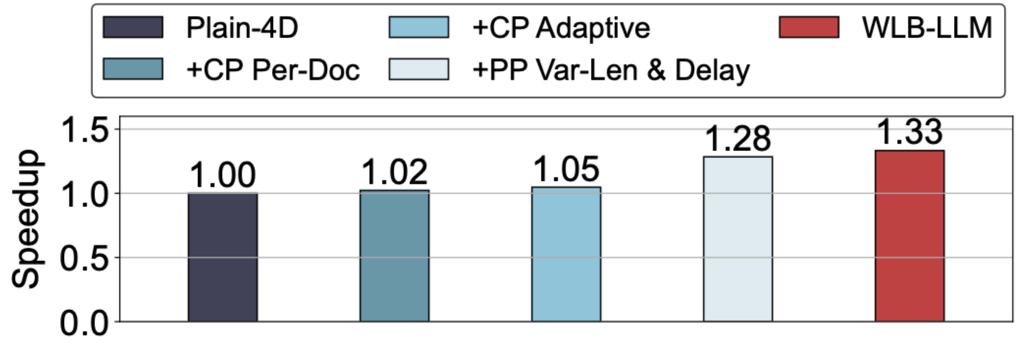


Figure 13: Performance breakdown of *WLB-LLM* on the 7B model with a 128K context window.

Speedup across Model + Context

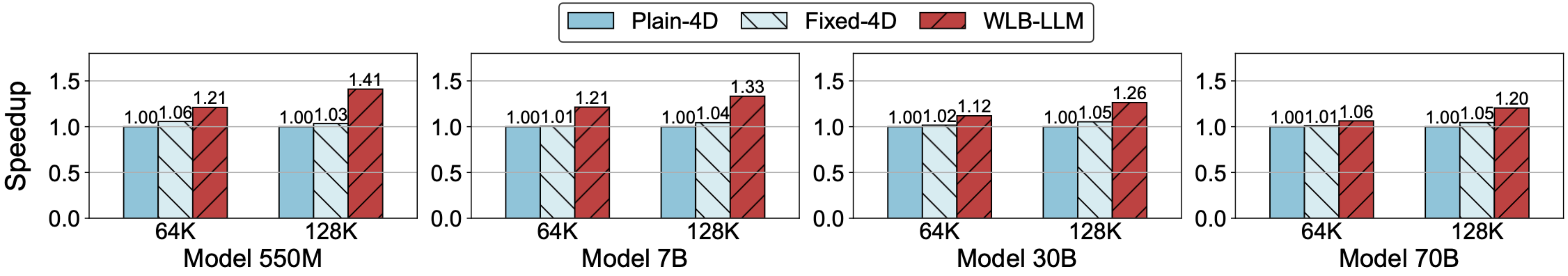


Figure 12: Training performance speedups of *WLB-LLM* and *Fixed-4D* over *Plain-4D* across various configurations.

Speedup vs Context Window

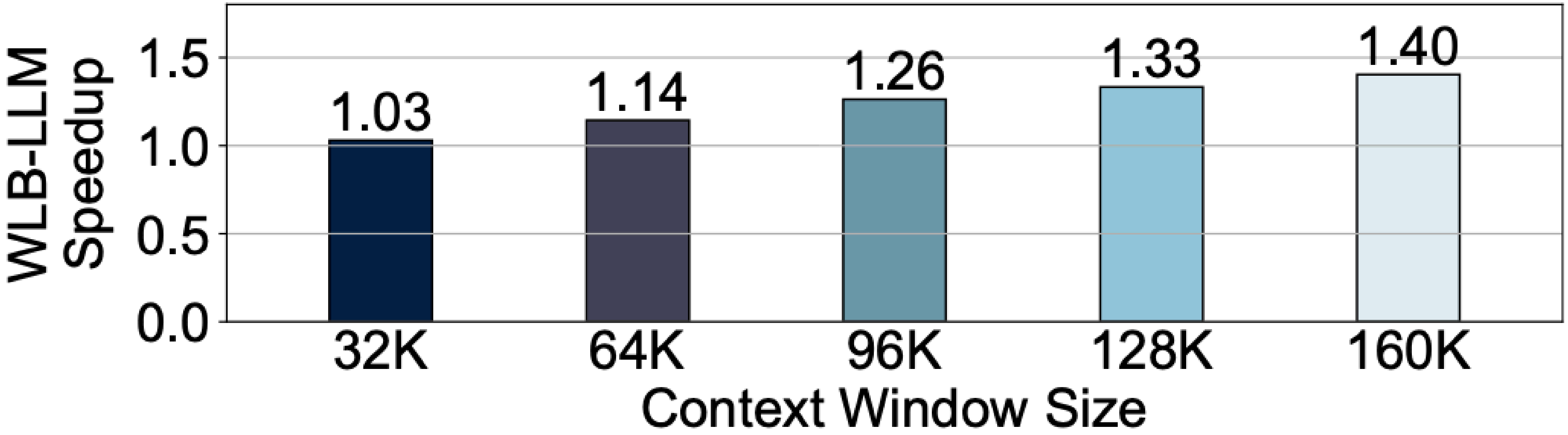


Figure 14: Speedups of *WLB-LLM* on the 7B model across context window sizes.

Ablation - Packing Overhead

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.

Ablation - CP Sharding

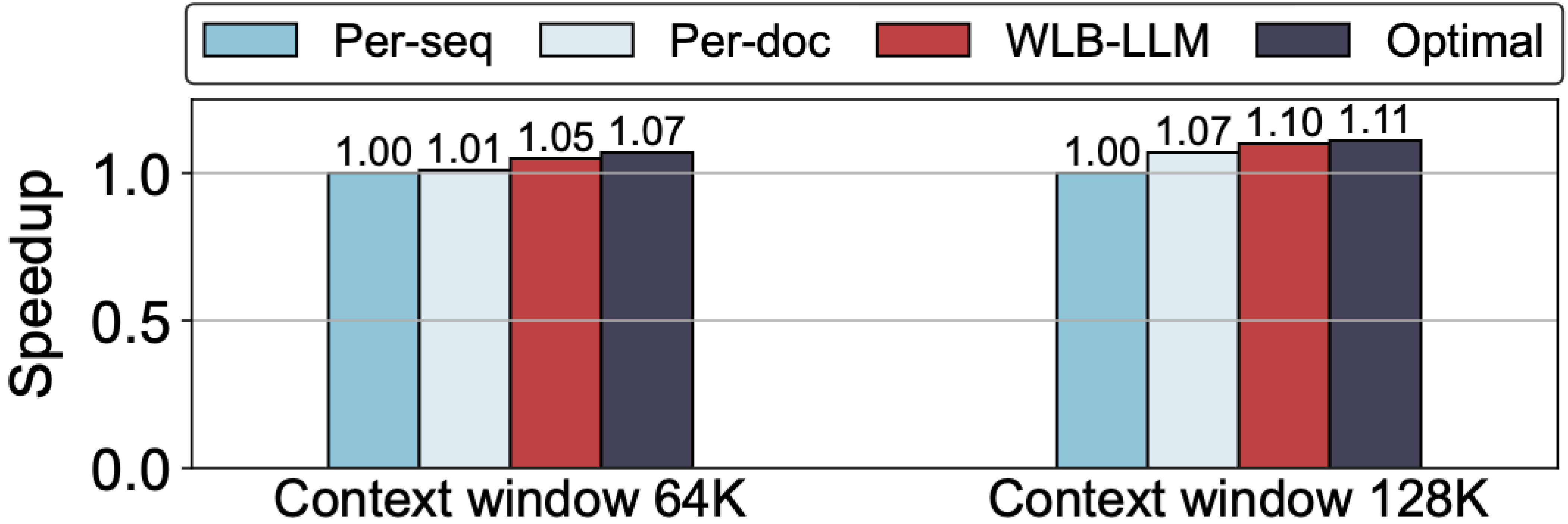


Figure 15: CP sharding performance comparison.

Discussion & Critique

Strengths:

- Clear bottleneck identification with evidence and analysis.
- Practical, deployable solution that demonstrates measurable gains.
- Careful consideration of overheads and convergence.

Questions:

- Larger models show smaller relative gains due to communication overhead, do gains vanish at extreme scale?
- How does this approach scale to MoE models?

Key Takeaway

Batch compute, not tokens.