Seesaw: High-throughput LLM Inference via Model Re-sharding

Qidong Su[1,2,3], Wei Zhao[3,4], Xin Li[3], Muralidhar Andoorveedu[3],
Chenhao Jiang[1,2], Zhanda Zhu[1,2,3], Kevin Song[1,2],
Christina Giannoula[1,2,3], Gennady Pekhimenko[1,2,3]

1 University of Toronto
2 Vector Institute
3 CentML
4 Stanford University

Overview

Problem:

Throughput-*oriented* *offline* LLM text generation with multiple GPUs

Observation:

Prefill and *decode* favor different *parallelization* strategies.

Key Ideas:

Dynamically *switch* between different parallelization strategies

Evaluation Results:

1.36x throughput increase (up to **1.78x**) compared with vLLM.

LLM Offline Inference

Online:

Prompt → Completion

Chat bot

Metrics:
Latency,
TTFT/TBT

Offline:

Prompt
Prompt
Prompt

→

Completion
Completion
Completion

...

Batch API **50% cheaper**

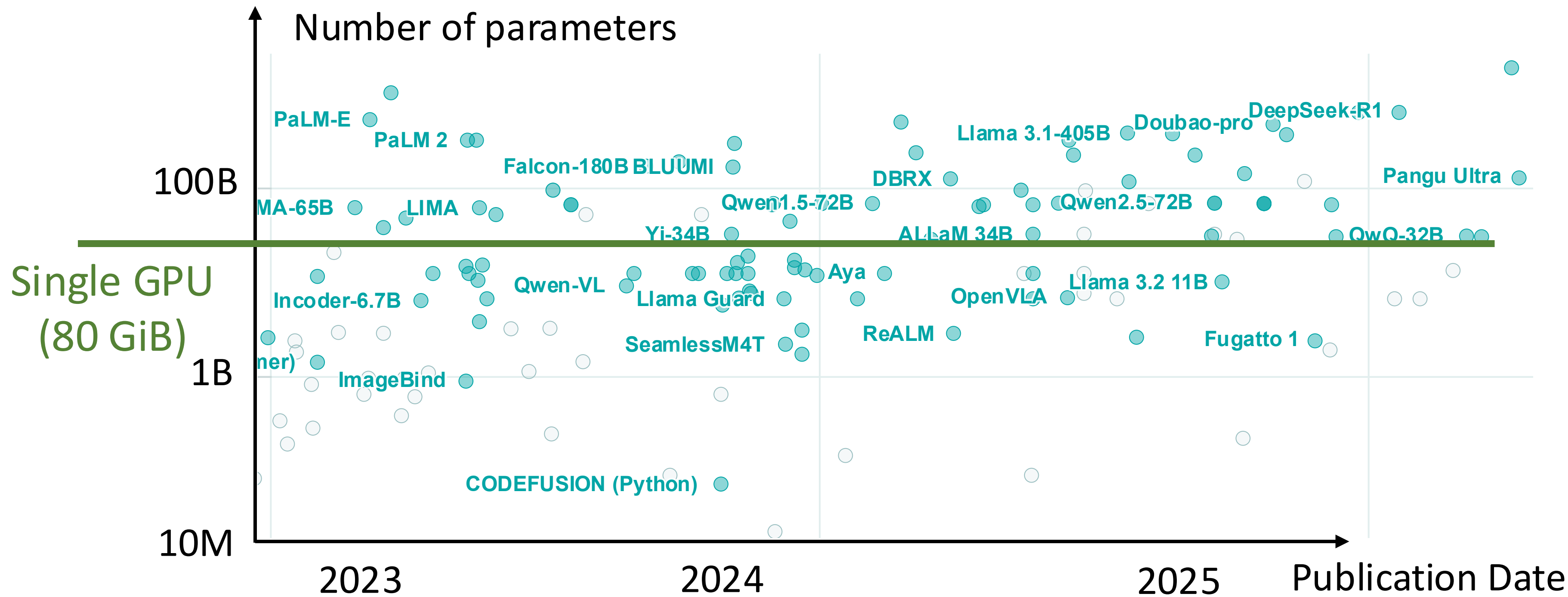
Benchmark

Codebase Analysis

Dataset Synthesis

Metrics:
Throughput
E2E time

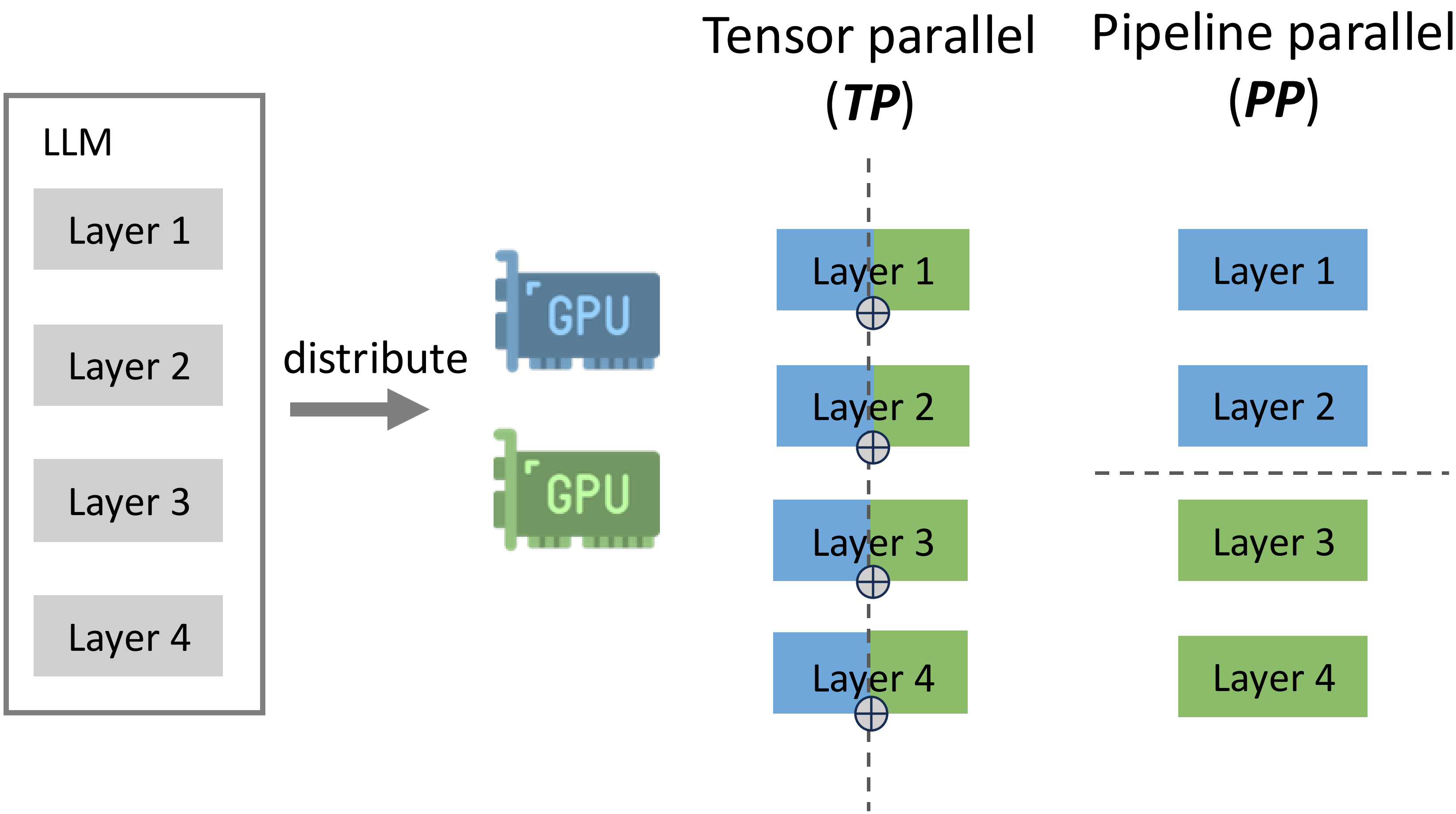
Modern LLMs are Large



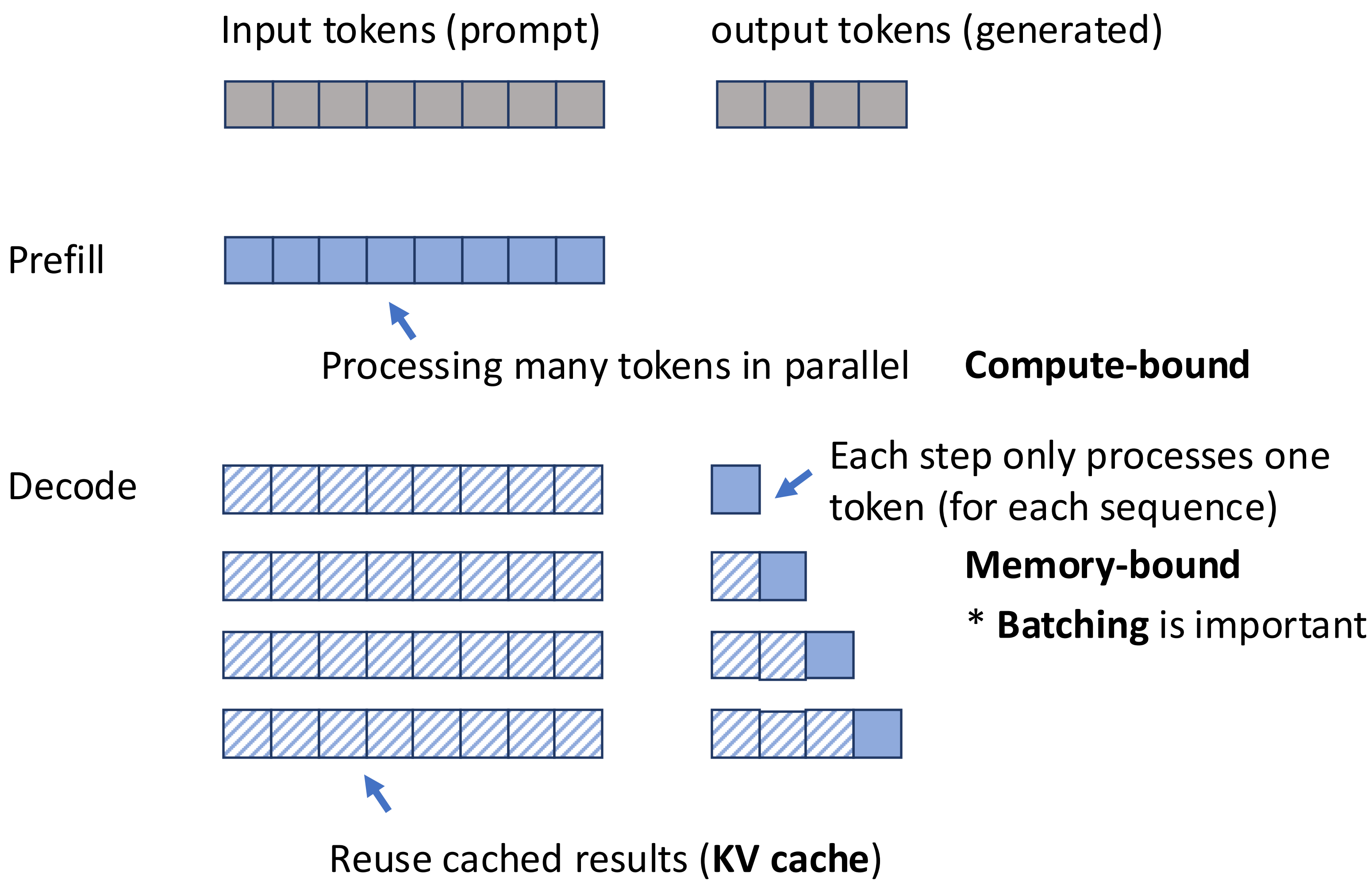
- We need multiple GPUs to deploy them.
- How to partition the model?

*source: https://epoch.ai/data/notable-ai-models

How to Partition a Model?



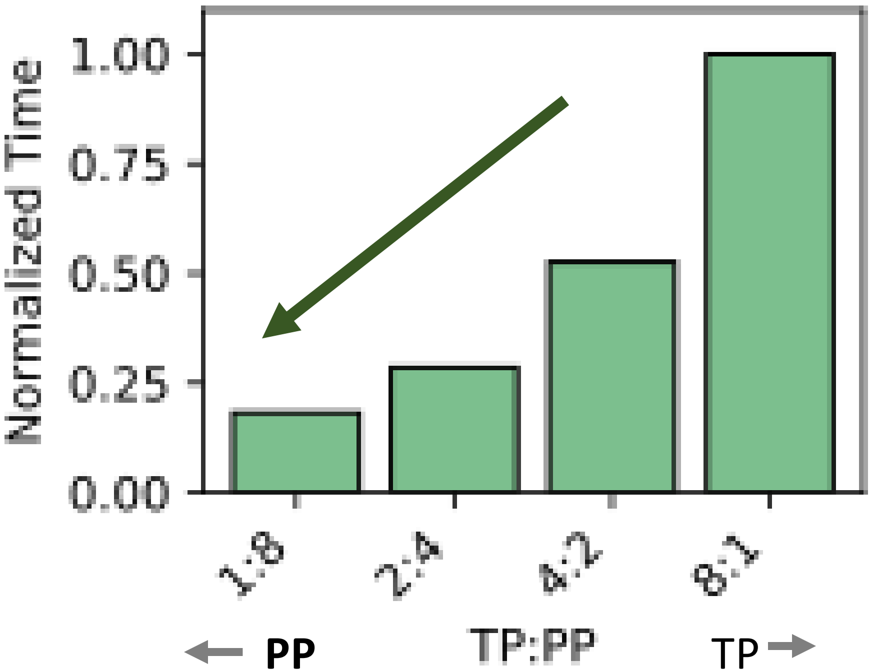
Prefill and Decode



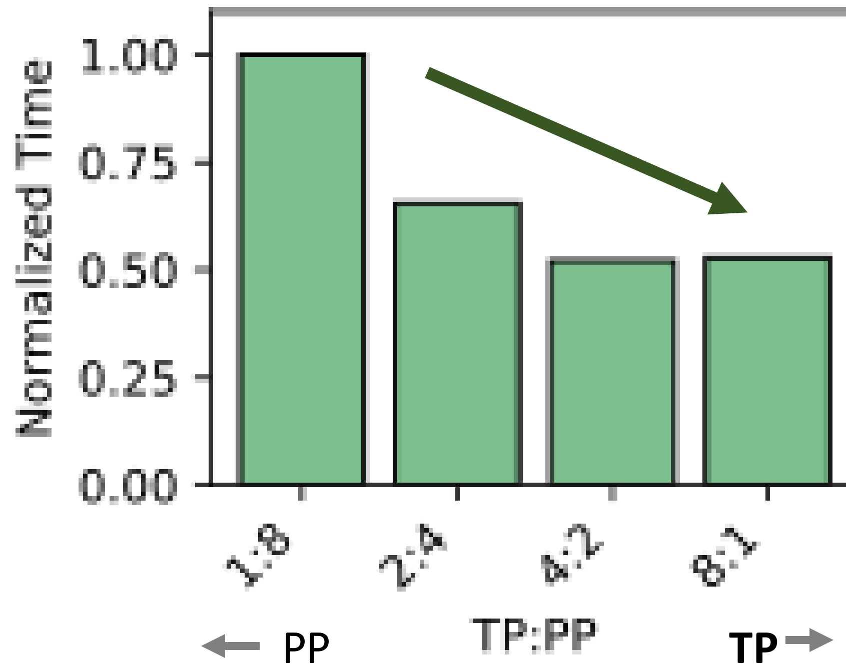
PP for prefilling, TP for decoding

(lower is better)

prefill

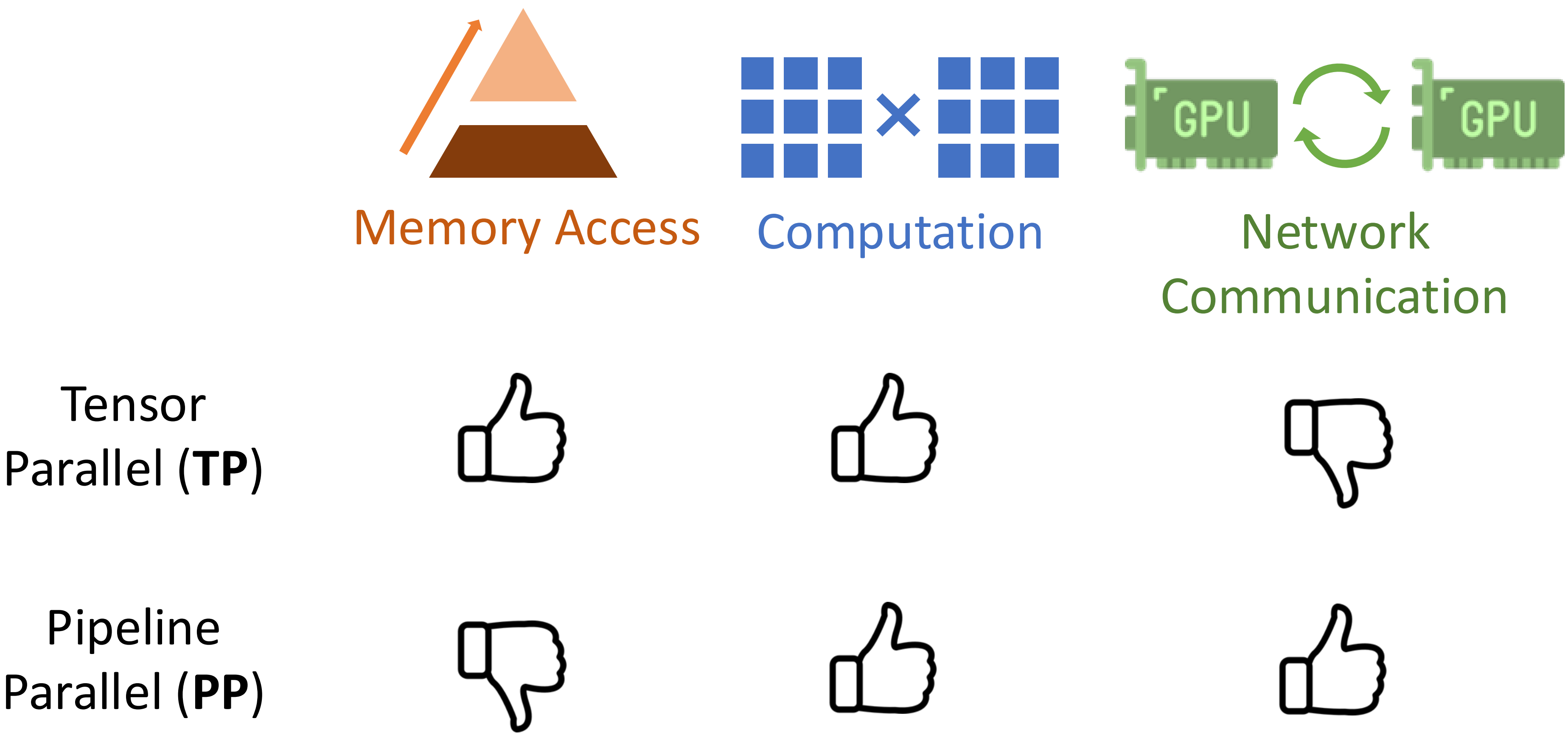


decode

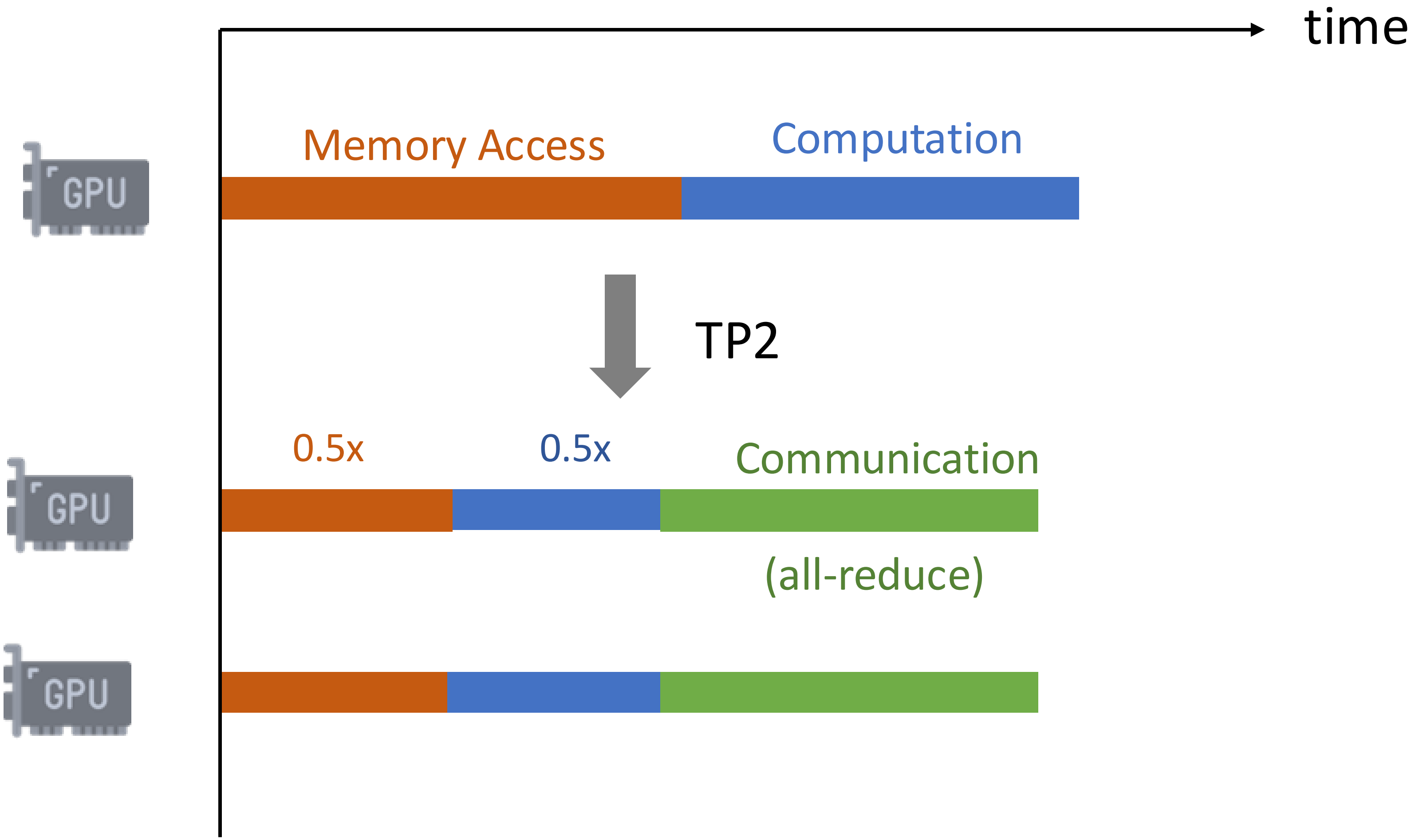


(Measured on 8xL4)

How are TP and PP different?

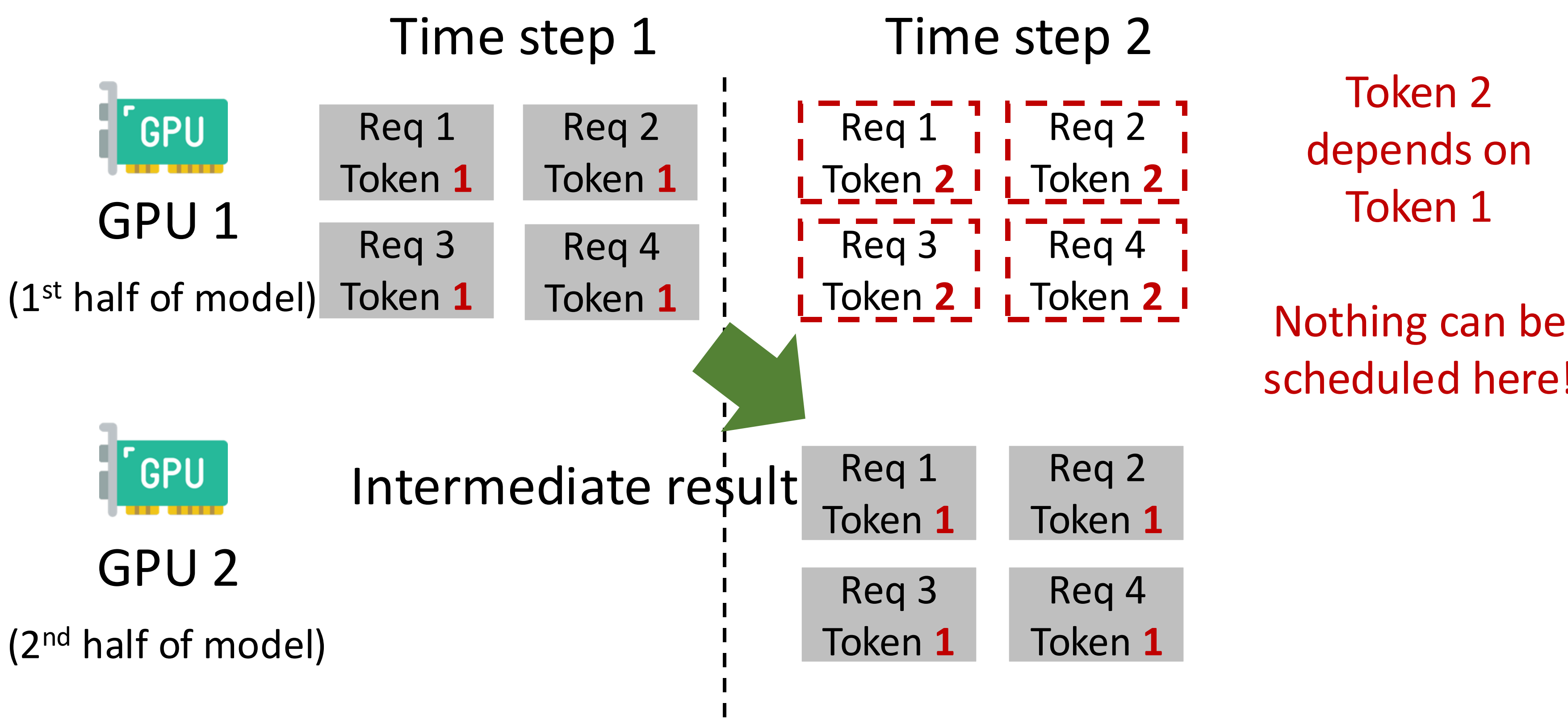


TP: communication overhead



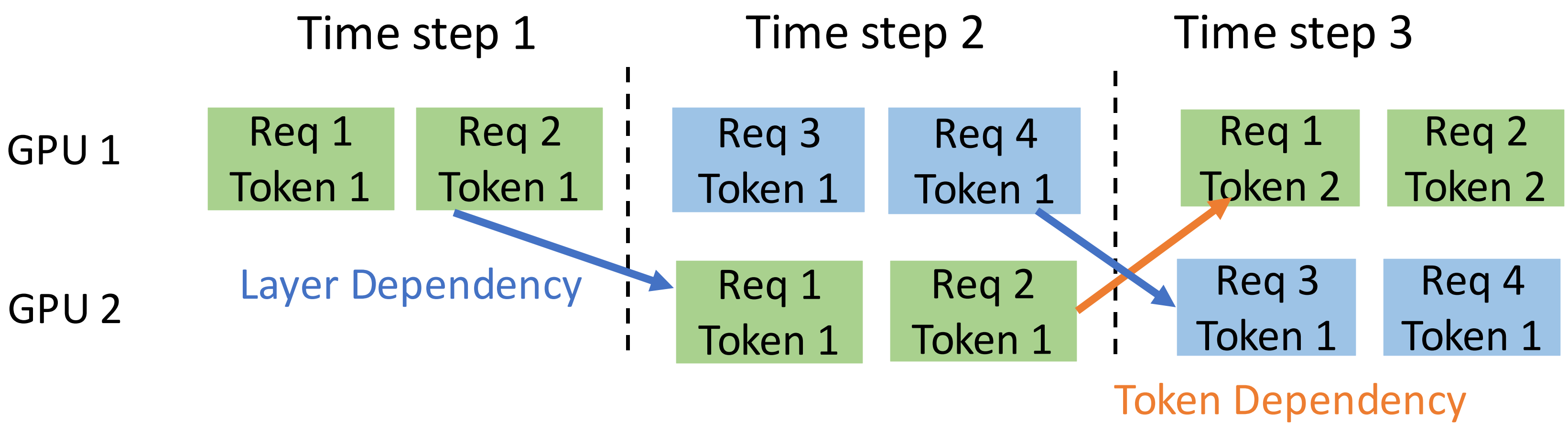
Pipeline Parallel: Micro-batching

Example: 2 GPUs, 4 requests



Pipeline Parallel: Micro-batching

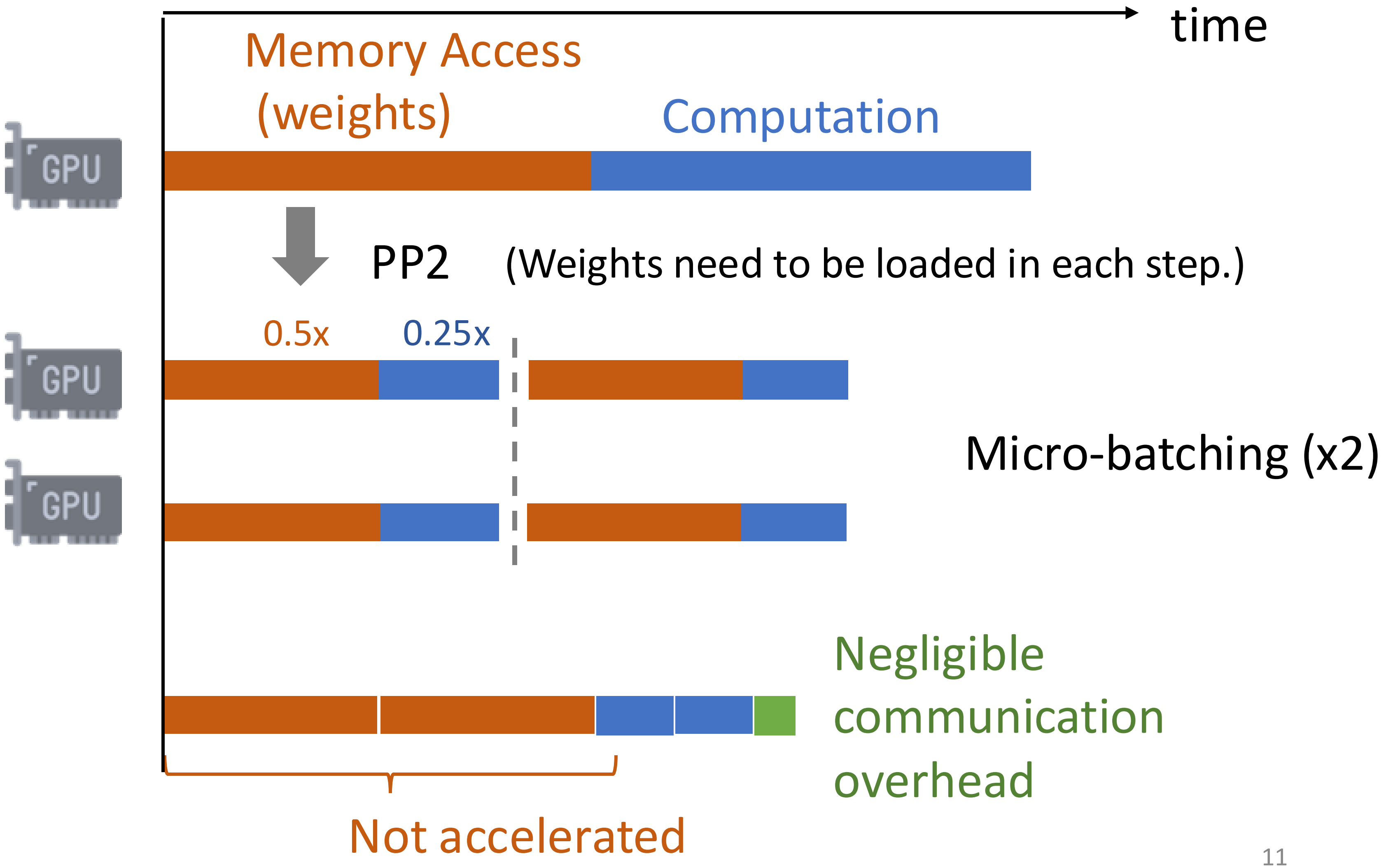
Example: 2 GPUs, 4 requests



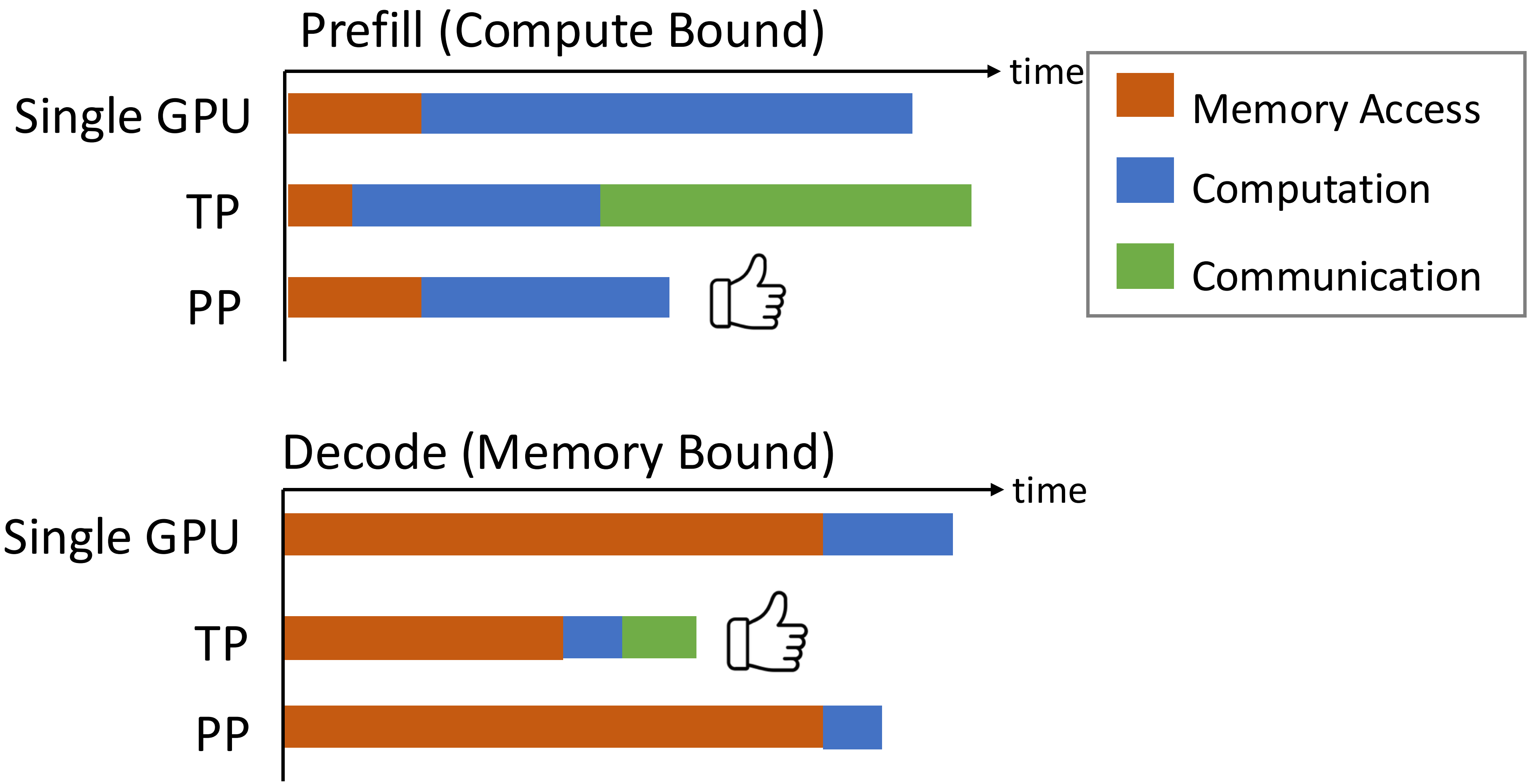
A step can only process $\frac{\mathbf{1}}{\boldsymbol{PP}}$ of all requests!

We need $\boldsymbol{PP}$x more steps!

PP doesn't accelerate loading weights



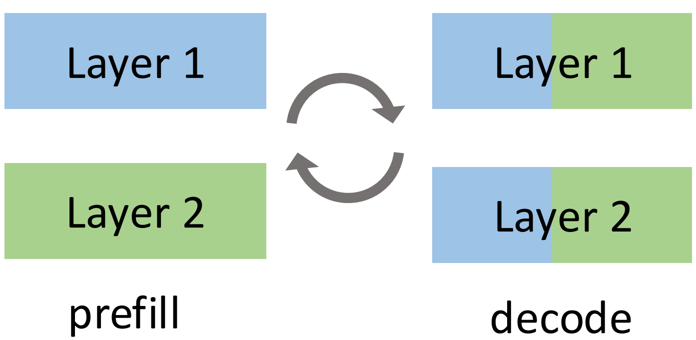
TP=efficient memory access
PP=reduce communication

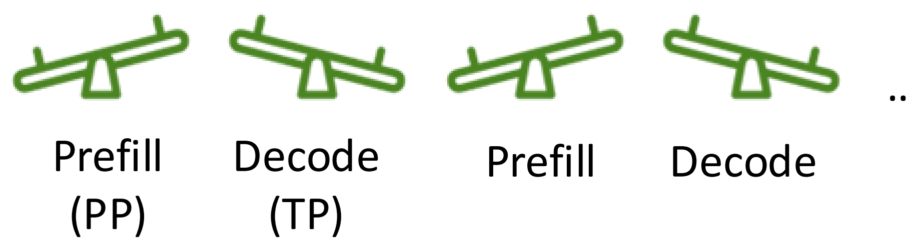


Seesaw: Key Ideas

- Pipeline Parallelism for Prefilling
- Tensor Parallelism for Decoding

- Re-Shard the model during the stage transition.





Challenge: Reshard Overhead

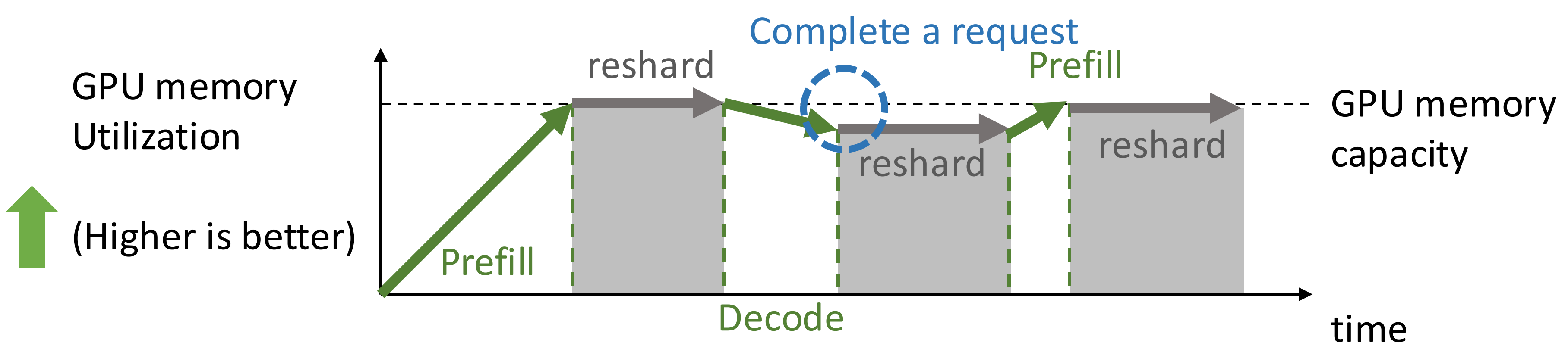
- Re-sharding moves tens of gigabytes of tensors
- We need to reduce the resharding frequency

However, **Continuous Batching** causes high reshard frequency.

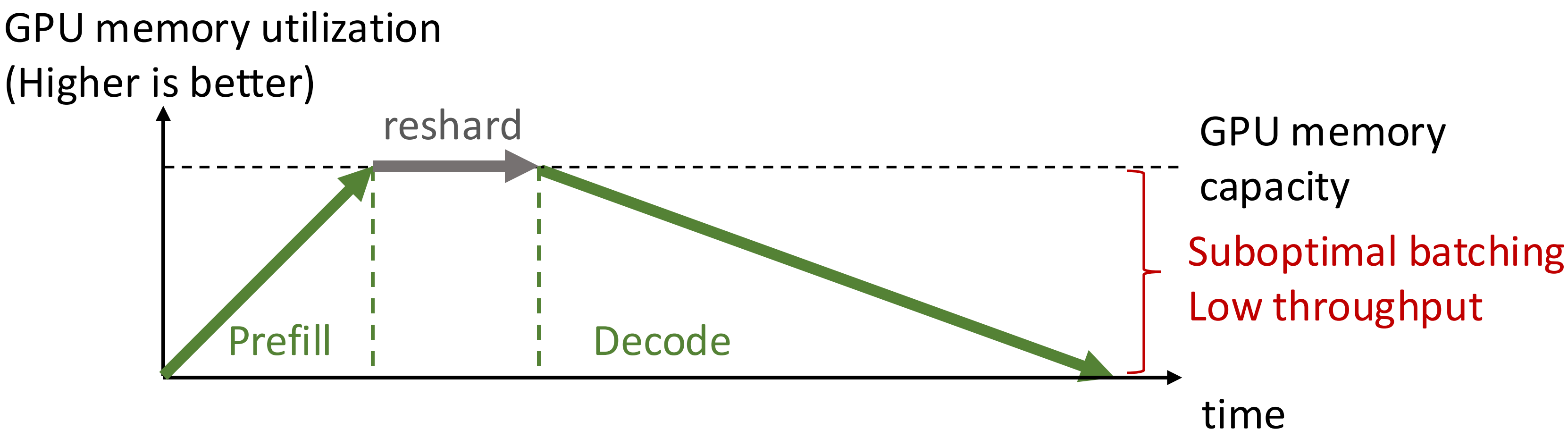
Challenge: Reshard Overhead

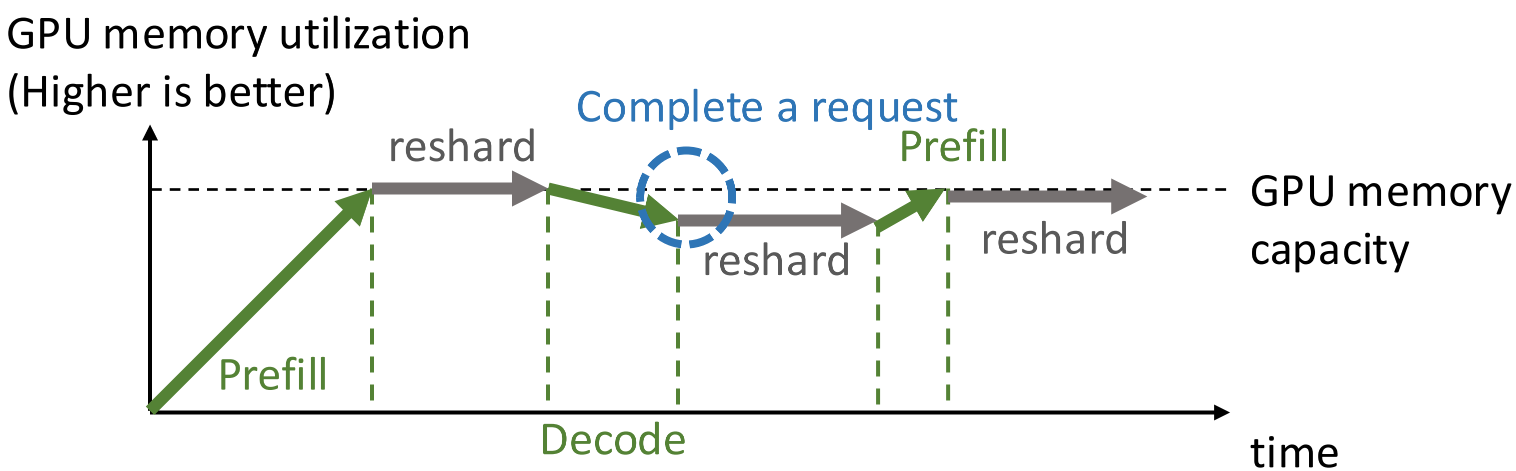
Recall:

- In decoding, large batch sizes = higher throughput
- Higher memory utilization gives larger batch sizes



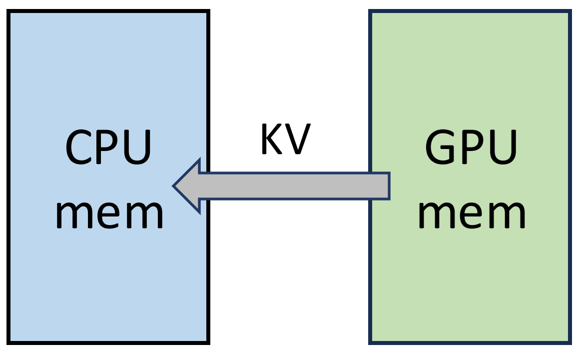
Alternative: Decode-Prioritizing





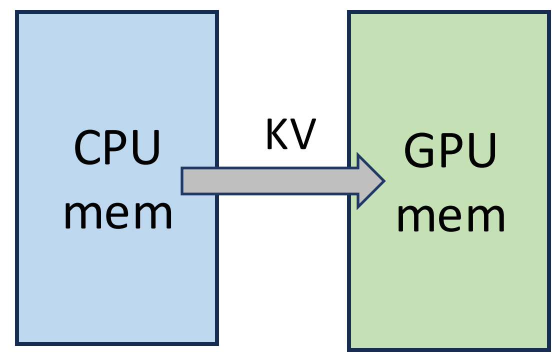
Seesaw: Tiered KV Cache Buffering

Prefill



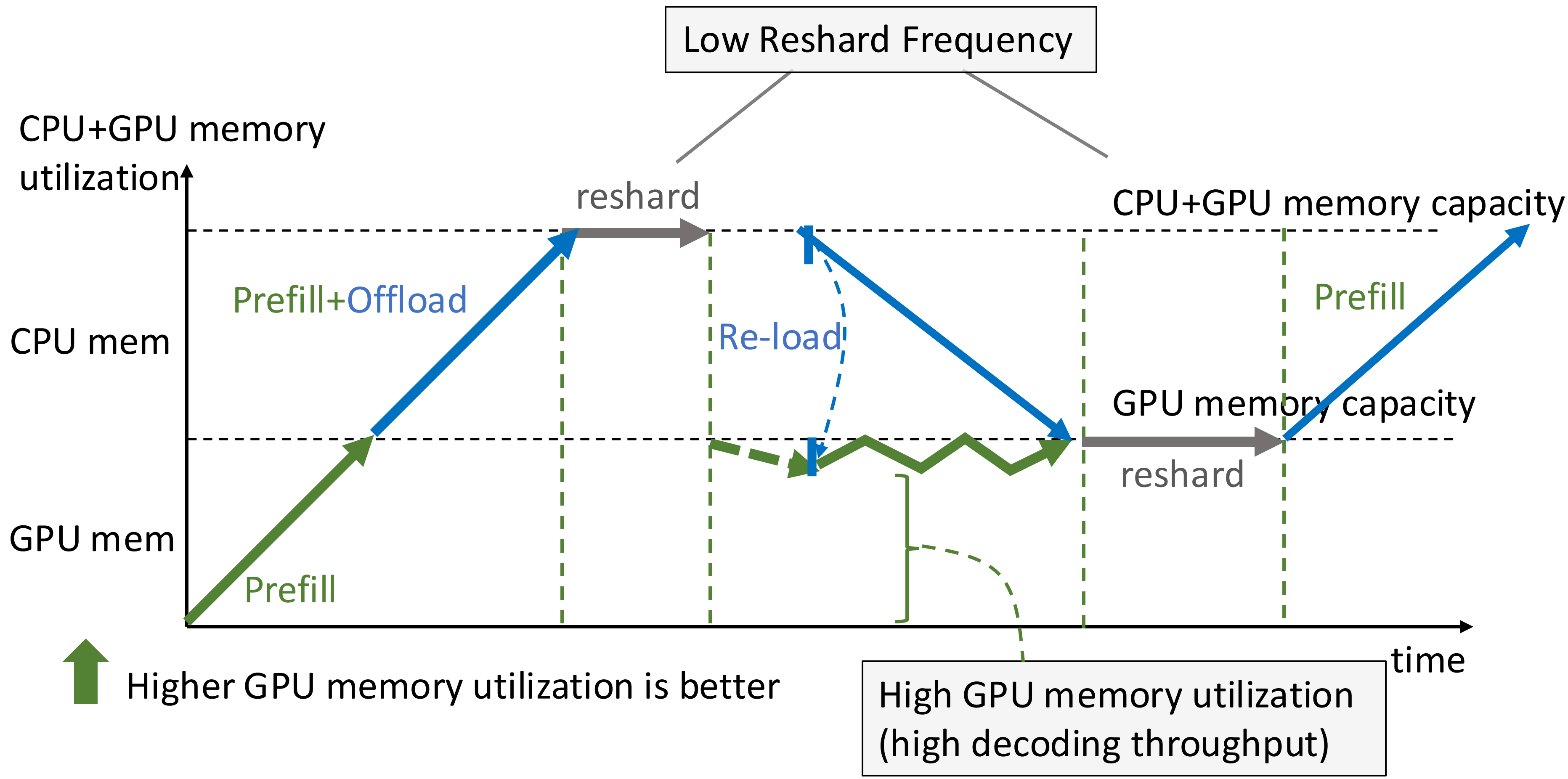
Offload KV cache to CPU
(Overlap with computation)

Decode



Reload KV cache from CPU
memory (asynchronously).

Seesaw: Tiered KV Cache Buffering



Higher GPU memory utilization is better

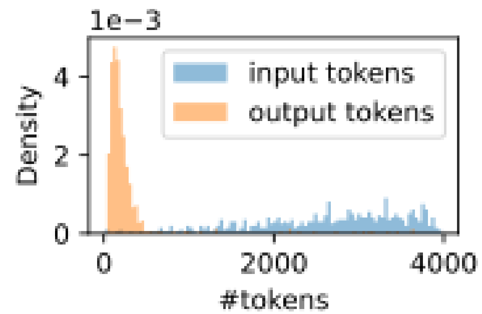
Evaluation Methodology

Hardware:
- A10 (24G, PCIe)
- L4 (24G, PCIe)
- A100 (40G, PCIe/NVLink)

Model:
- LLaMA3 15B
- CodeLLaMA 34B
- LLaMA2 70B

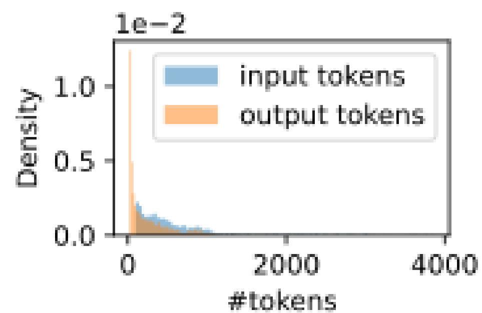
Baseline:
- vLLM
- + chunked prefill
- + tuned chunk size

Workload:



arXiv-Summarization

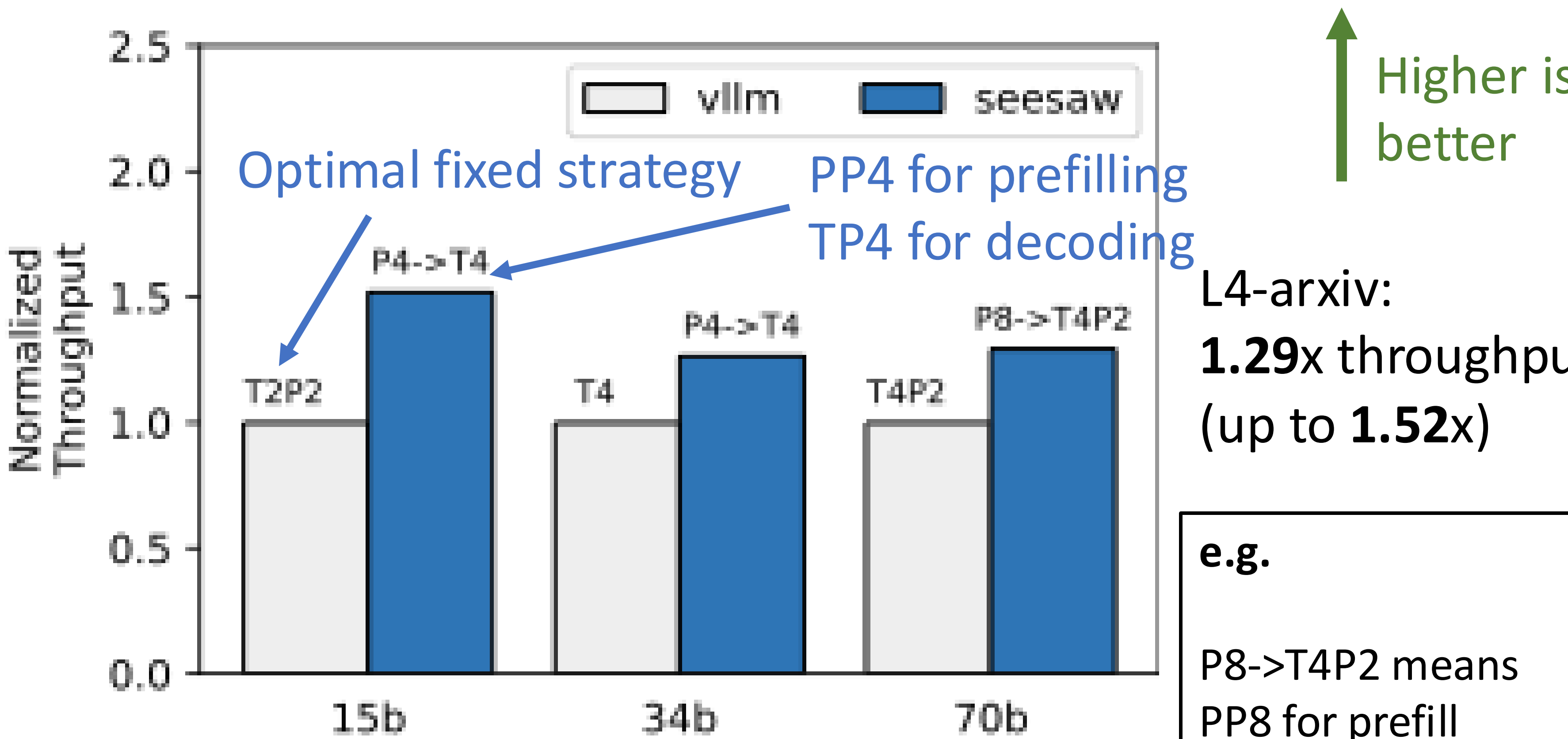
input > output



ShareGPT

Input ≈ output

End-to-end Throughput Compare

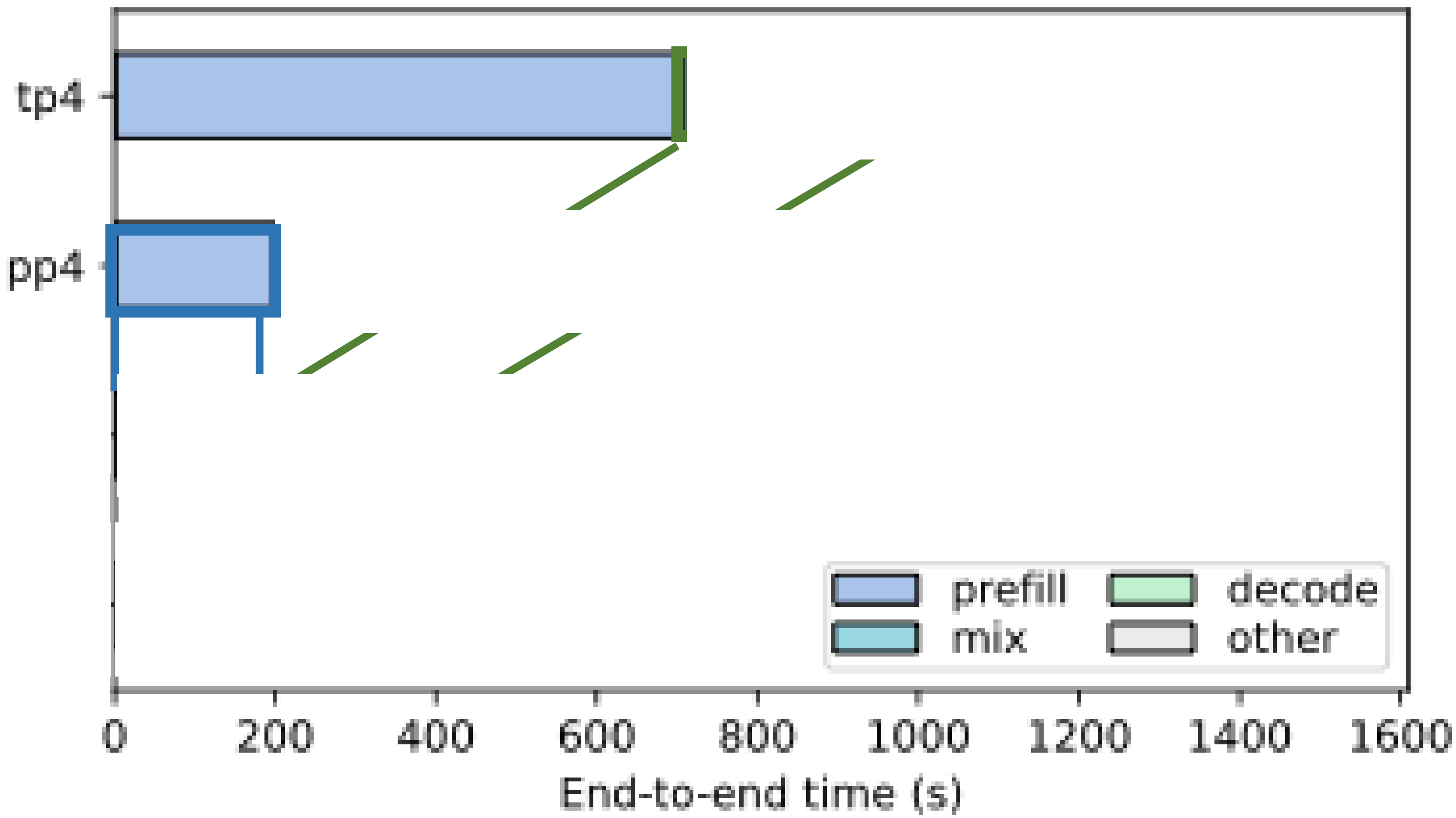


L4-arxiv:
1.29x throughput
(up to **1.52**x)

e.g.

P8->T4P2 means
PP8 for prefill
TP4 + PP2 for decode

Speedup breakdown



Lower is better

Summary

Problem:

Throughput-oriented offline LLM text generation

Observation:
Prefill and decode favor different parallelization strategies.

Key Ideas:

Dynamically switch between different parallelization strategies.

Tiered KV cache buffering to reduce transition overhead

Evaluation Results:

1.36x throughput increase (up to **1.78x**) compared with vLLM.