NVIDIA

SHIP: SRAM-based Huge Inference Pipelines for Fast LLM Serving

Presenter: Andrew Bitar, LPU Team

MLSys'26 | May 21, 2026

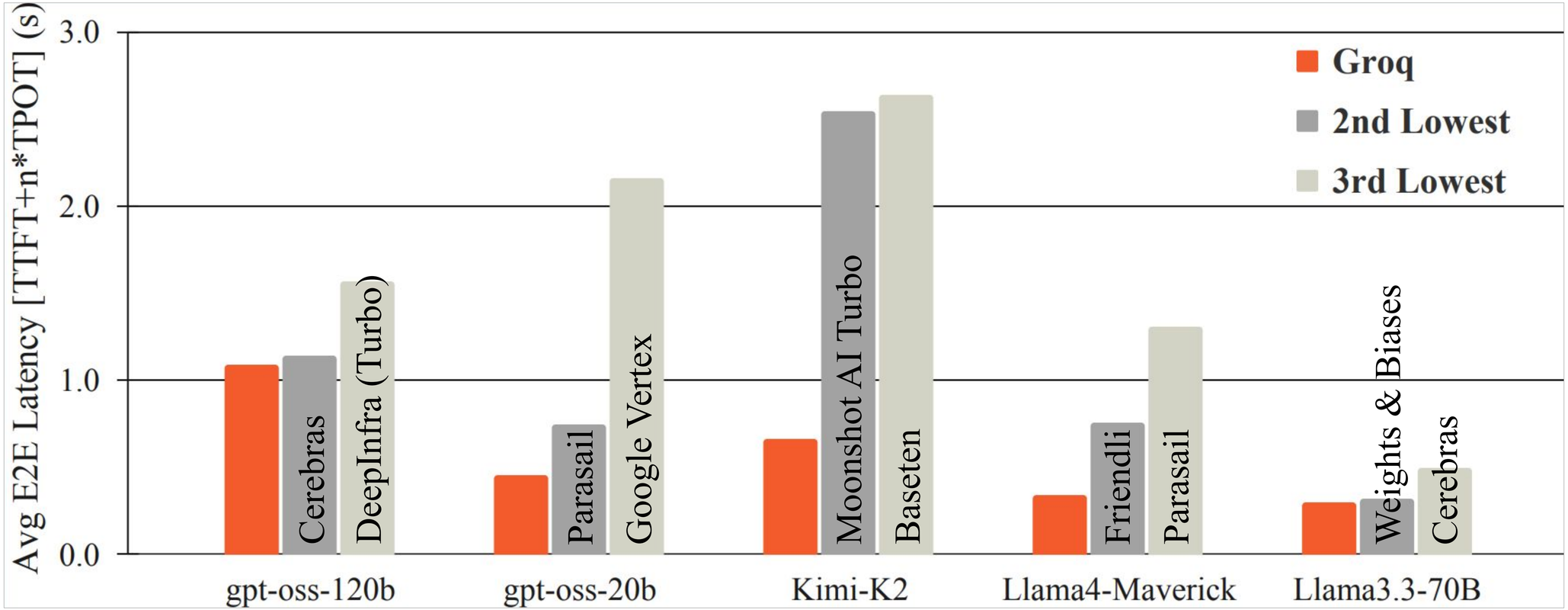
Authors:

Andrew Bitar · Aravind Vellora Vayalapra · Baorui Zhou · Matthew Boyd · Charlie Wang · Sahil Parmar · Eugene Sha · Gautam Rayaprolu · Peter Hicks · Alex Bowe · Roberto DiCecco · Santosh Raghavan · Evan Patrick · Josip Smolcic · David Han · Kris Kang · Andy Rock · Josh Hay · Mohamed Eldafrawy · Mikhail Kandel · Daulet Zhanguzin · Omar Kilani · Liming Gong · Andrew Paprotskyi · Arash Taheri-Dezfouli · Josh Fender · Andrew Ling

A retrospective on
Groq LPUv1
used to build
the first SRAM-based
LLM inference cloud

Fast SRAM + Massive Scale

= Fastest LLM Serving Platform



Average request latency over one-month period (Sep 26 – Oct 26, 2025).

On Groq using LPUv1 versus next two fastest providers for each model.

Data from OpenRouter (https://openrouter.ai/).

Agenda

Background

Motivation: LLMs in SRAM

Scale-Up Topology

Memory Capacity Management

Dynamic and Balanced Pipeline

LLM Serving

Background

Not all tokens have same compute and memory needs

- Prefill vs. decode, context position
- Systems must dynamically adjust to maintain throughput & latency

Negative interaction from concurrent prefill/decode and contexts

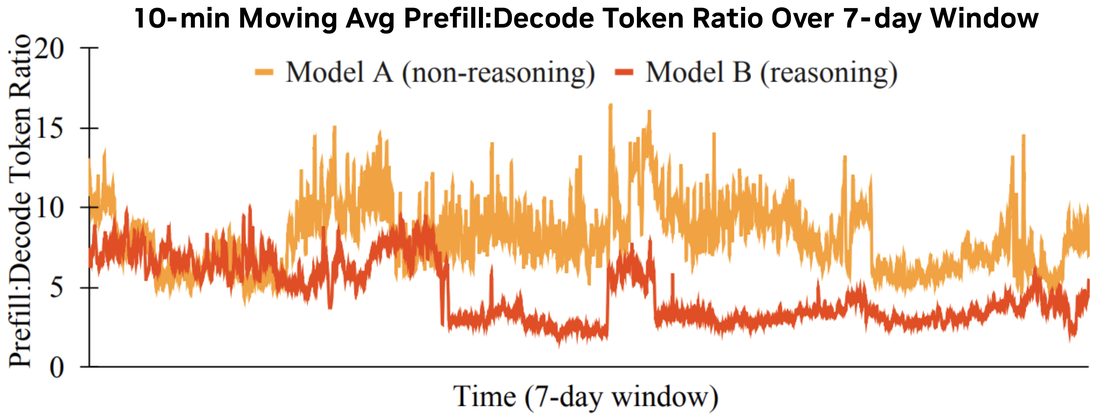
- Decode tokens add KV-cache load, increasing latency, hurting TTFT
- Long prompt requests increase iteration latency and drive up TPOT

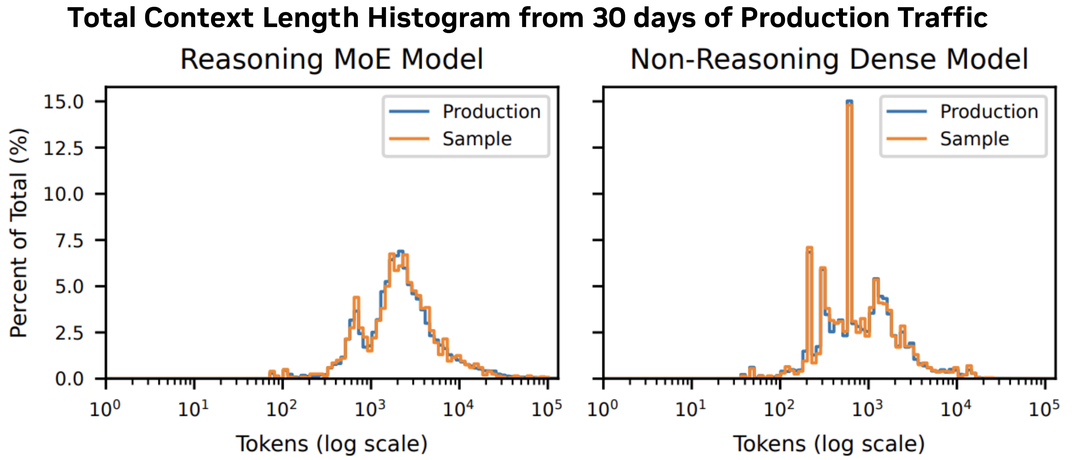
Continuous batching[1] and prefill chunking[2] help

- But still suffer from sub-optimal utilization with fluctuating P:D ratio and context lengths

Disaggregated prefill-decode

- Move KV cache between prefill and decode workers
- Continuous re-allocation of compute to prefill vs decode workers
  - Dynamo!

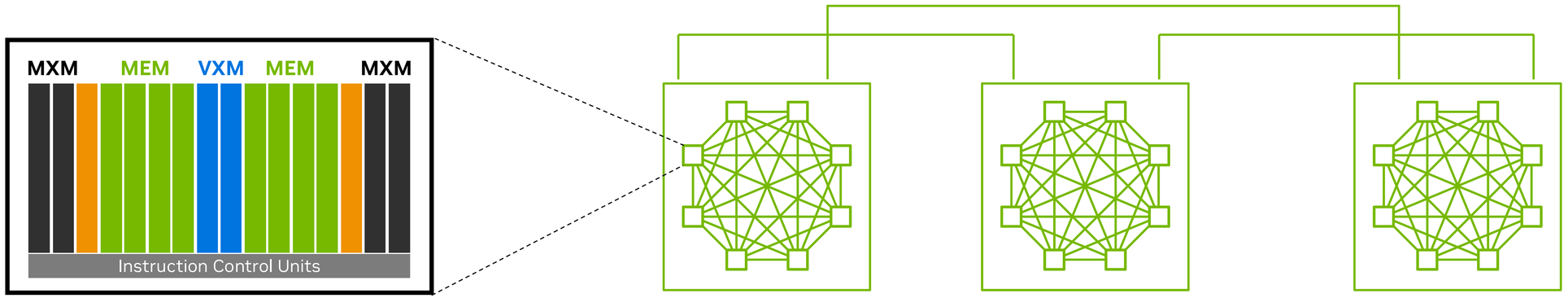




[1] Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for {Transformer-Based} generative models. In 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), pp. 521–538, 2022.
[2] Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., and Ramjee, R. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. arXiv preprint arXiv:2308.16369, 2023.

LPU Architecture

Background



LPUv1

GlobalFoundries 14nm (2016)

230 MB SRAM, 235 GB/s C2C

Deterministic execution, no reactive hardware

Statically-scheduled by LPU compiler

LPU System

Deterministic execution extended to entire system of LPUs

No dedicated router chips – LPUs act as both processor and router

LPU compiler schedules all network traffic

LLMs in SRAM

Motivation

Latency

Ultra-low TPOT

- Shrink the “thinking” lag in reasoning models
- Accelerate long-running agentic tasks

Ultra-low TTFT

- Fast decode tokens = fewer delays for prefill tokens
- Huge systems = user prompts strongly scaled across more compute resources

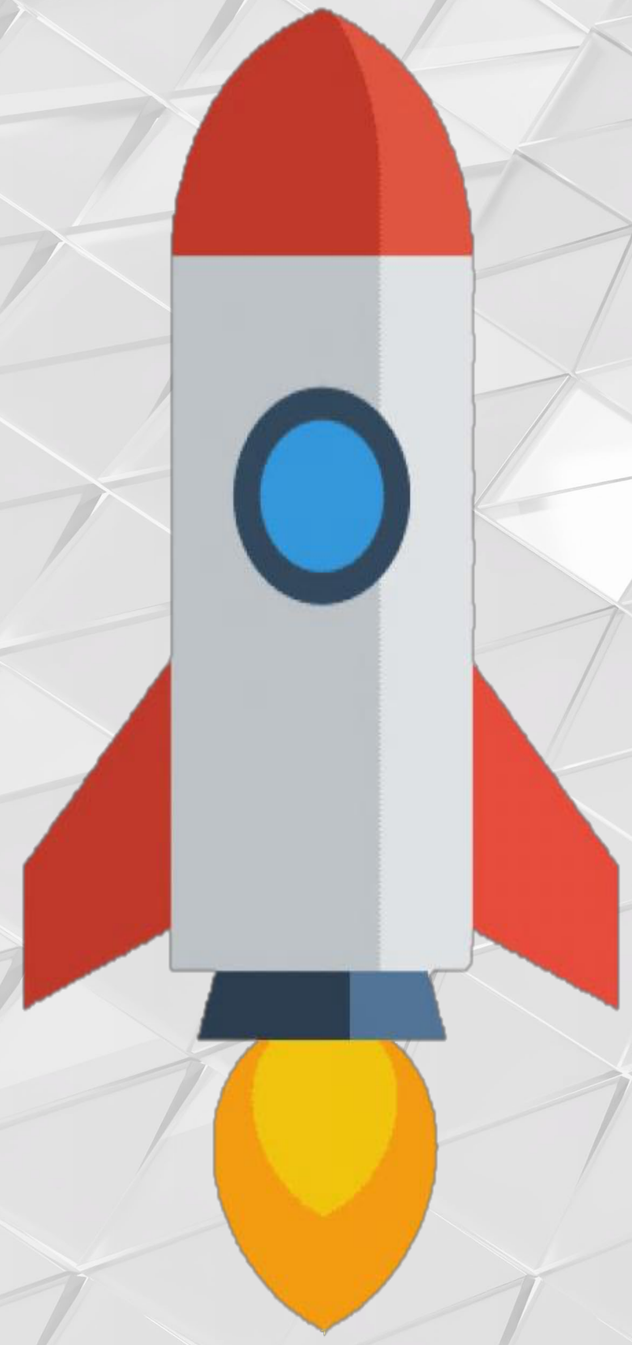
Efficiency

SRAM reduces compute-to-memory bandwidth gap

- Increases efficiency potential when operational intensity (OI) is low
  - Attention, MoEs
- But needs huge scale to access enough capacity

Scaling efficiently is the challenge

- Communication costs
  - Collective communication from tensor parallel (TP) scaling
  - Must overlap communication with compute
- Low-batch
  - SRAM = low-batch = latency-sensitive
  - Must minimize exposed sync latencies
- Pipeline parallel (PP)
  - Offers a low communication-cost method to access more capacity
  - Must minimize pipeline imbalance



SRAM-based

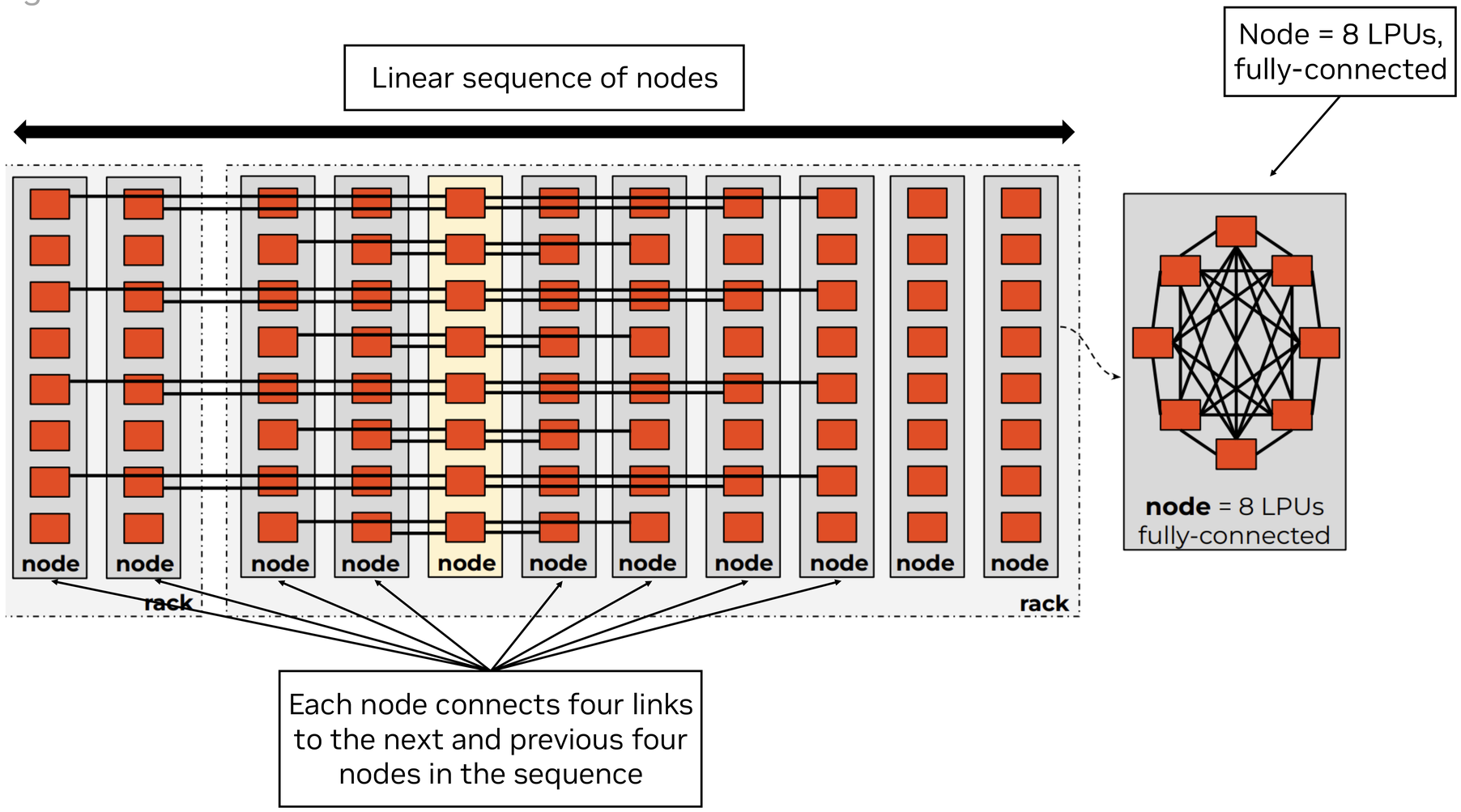
Huge

Inference

Pipelines

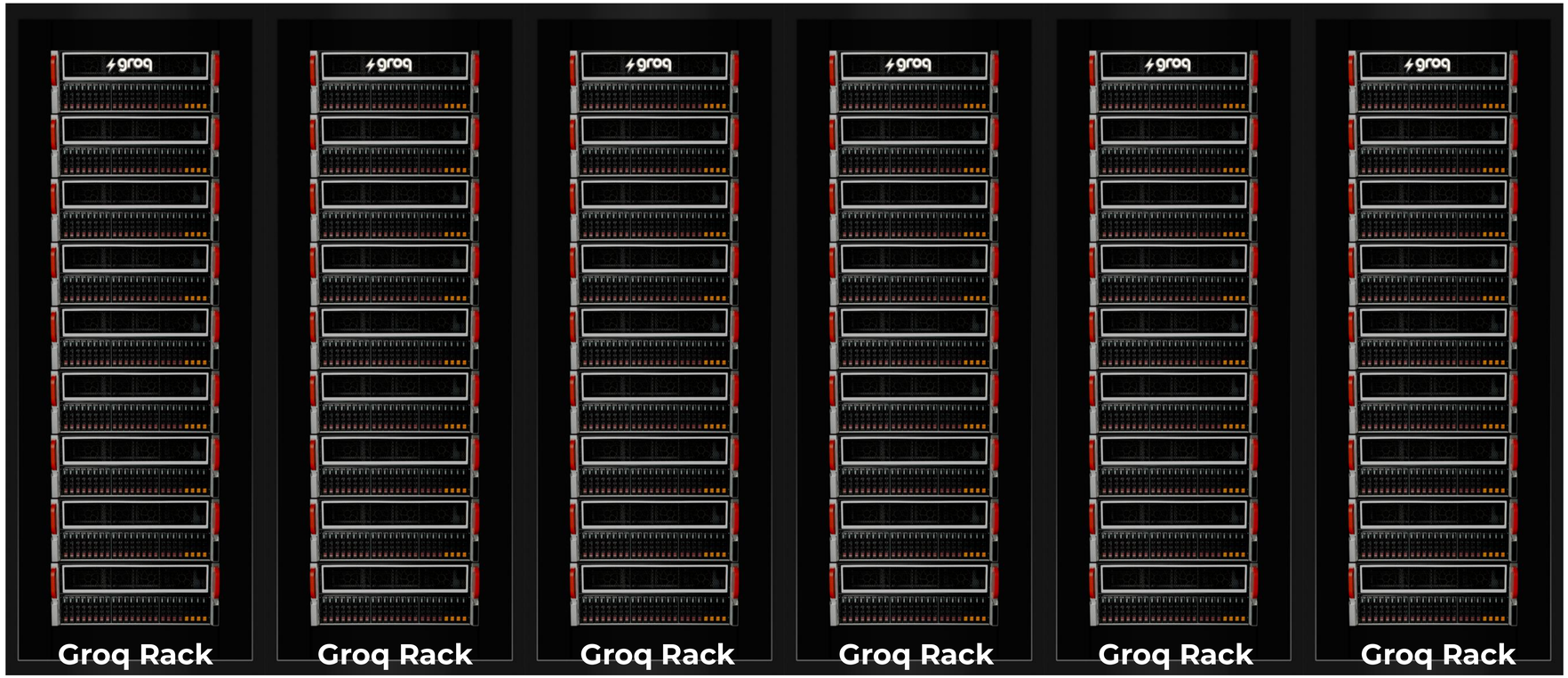
Scale-Up Topology: QuadFour

Very Large Scaling



Scale-Up Topology: QuadFour

Very Large Scaling



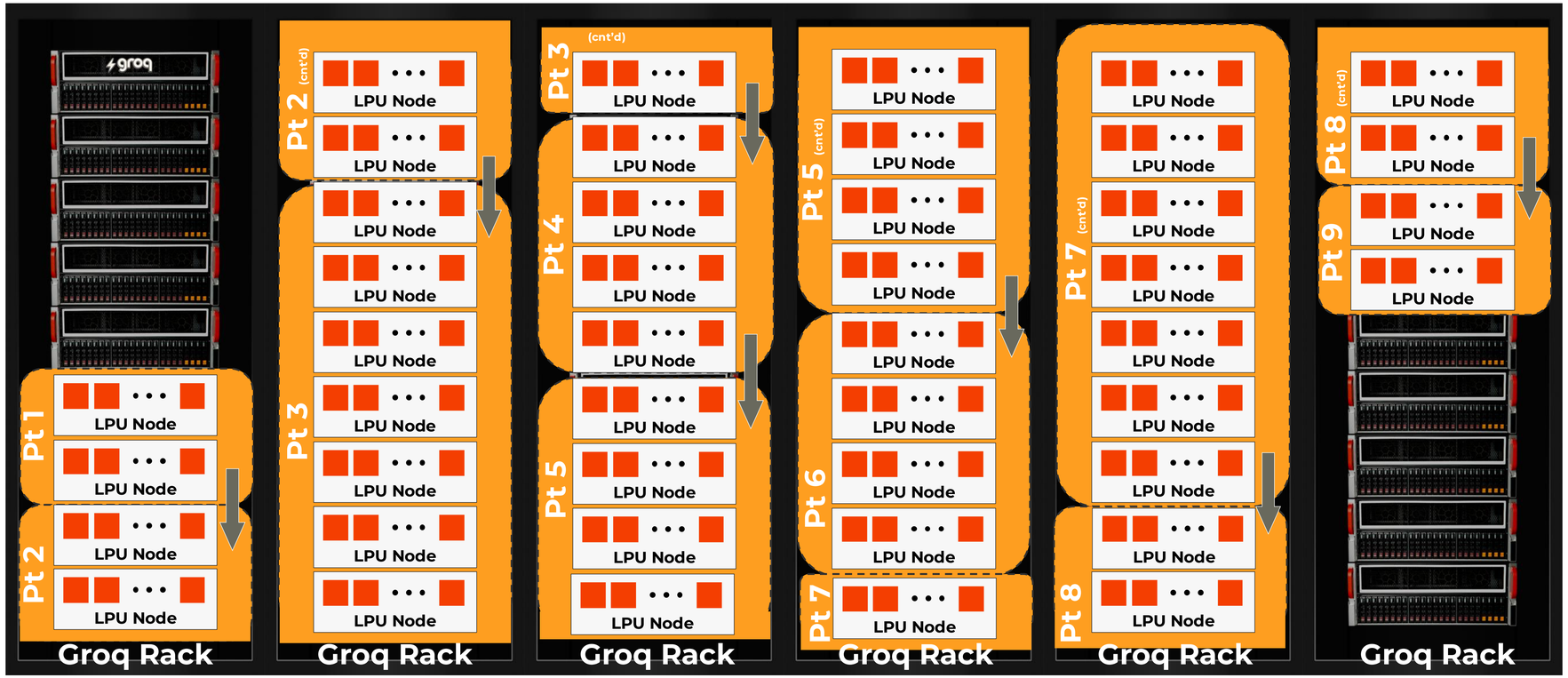
Scale-Up Topology: QuadFour

Very Large Scaling

Multiple partitions form a pipeline

Partitions can span rack boundary without loss in performance

Partitions are flexibly sized to their compute and memory needs



NVIDIA

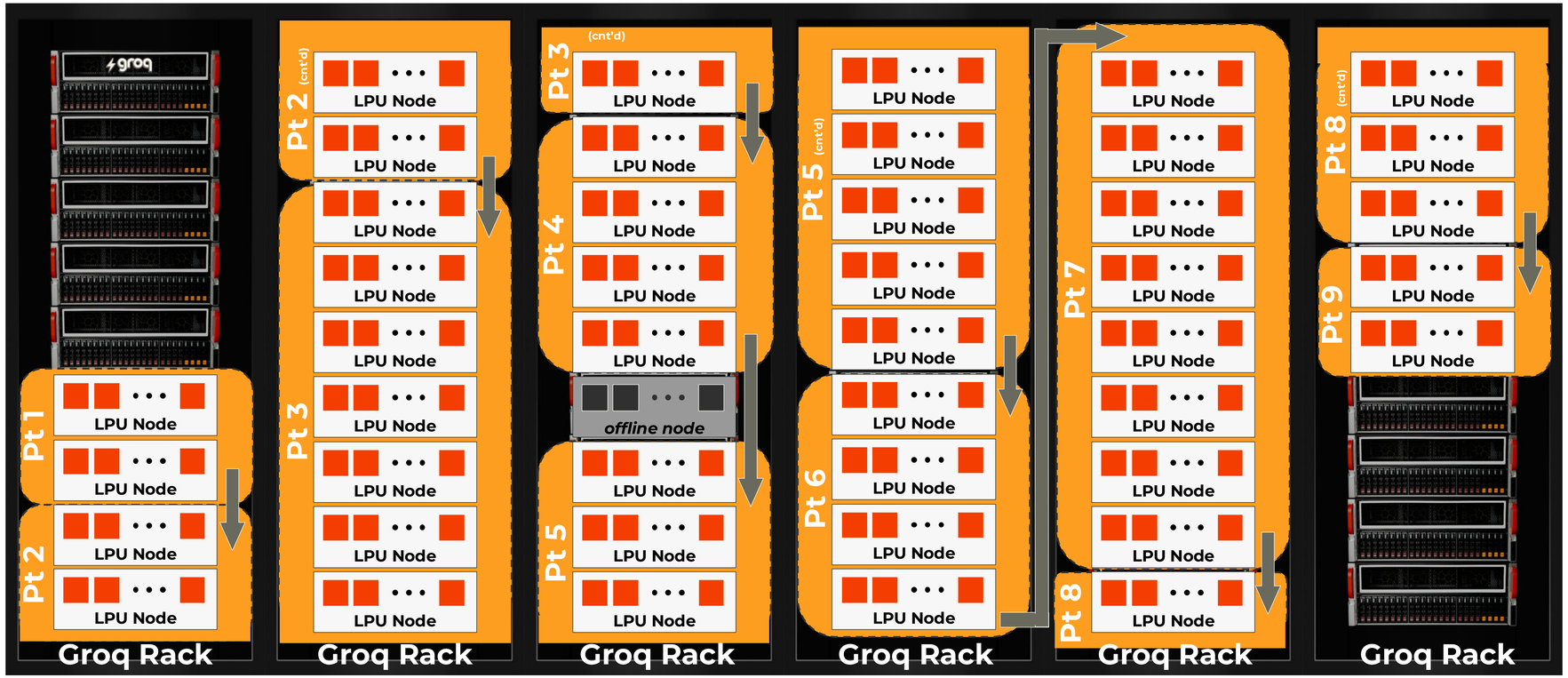
Scale-Up Topology: QuadFour

Very Large Scaling

Nodes experiencing failures can be skipped!

Partitions are rotated around failing node while being serviced

Up to three nodes can be skipped between partitions



Communication Collectives

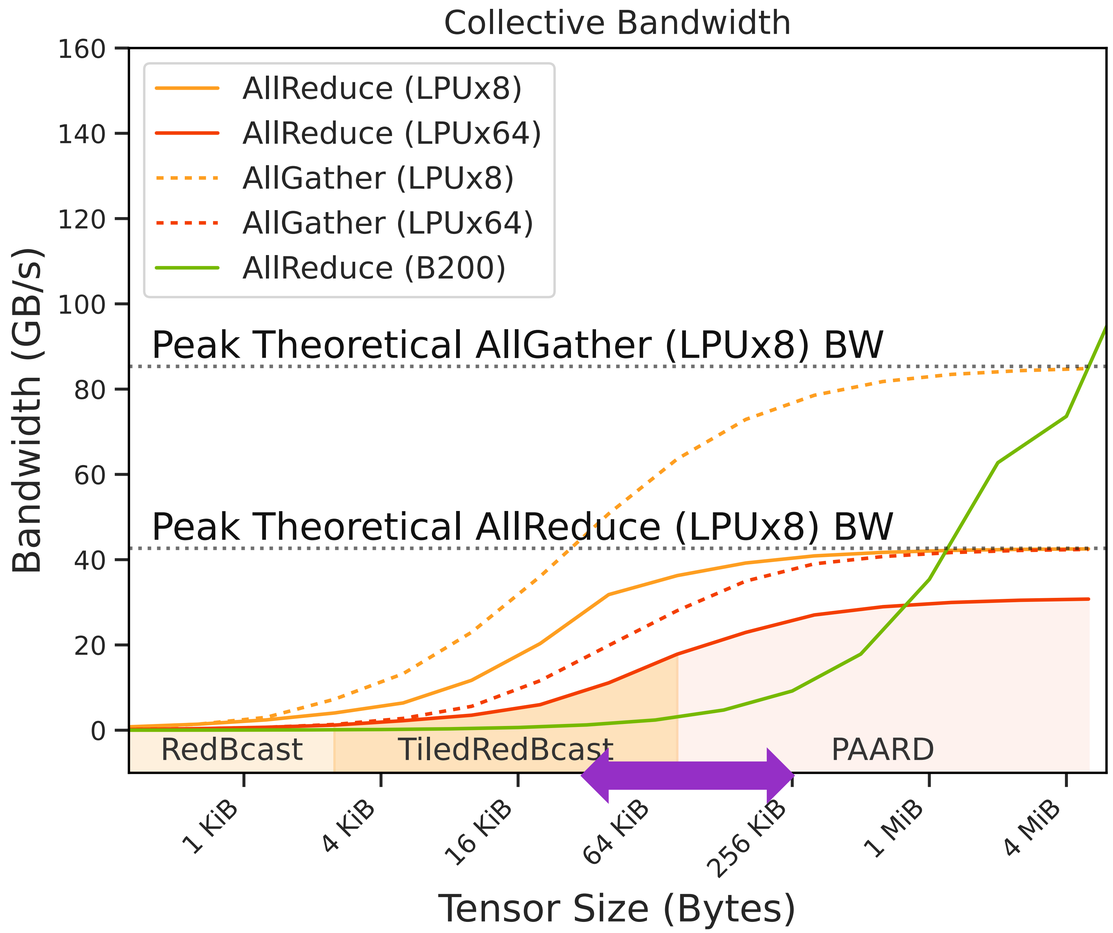
Very Large Scaling

Superior collective performance for low-batch tensors

- Typical SHIP collective tensor size: 32-256 KB
  - More latency-sensitive than bandwidth-sensitive!

Communication collective latency minimized by:

1. Low network diameter
   - 8 LPUs: 1 hop
   - 72 LPUs: 3 hops
   - 136 LPUs: 5 hops
2. Low hop latency (300 ns) via synchronous LPU C2C network
   - No intermediate routers
   - No synchronization latency per collective
   - No memory acquire-release latencies (lock-free)



Memory Management

Efficiently utilize precious SRAM capacity

Paged Attention

SHIP runtime manages allocating KV cache memory in LPU SRAM

Runtime sends “mask encoding” at start of pipeline that determines memory allocation across all partitions

Prompt: “Four score and seven years ago our” ①

Outputs: “fathers” ② → “brought” ③ → . . .

KV pages

Page 0	① Four	① score	① and	① seven
Page 1				
Page 2				
Page 3				
Page 4	③ brought			
Page 5				
Page 6	① years	① ago	① our	② fathers
Page 7				

① mask encoding $= [4, 0, 0, 0, 0, 0, 3, 0]$

② mask encoding $= [4, 0, 0, 0, 0, 0, 4, 0]$

③ mask encoding $= [4, 0, 0, 0, 1, 0, 4, 0]$

Prefix Caching

Expand SHIP memory for prefix caching via LPU node DDR4

Spreading single model across many nodes accesses large amounts of distributed DRAM

gpt-oss-120B prefix cache memory in SHIP

Memory Type	Memory Capacity (GB)	Memory BW (TB/s)
SRAM	69.7*	10,616**
DDR4	72,000	18.4

Speculative Decoding

Reduces KV cache capacity requirement

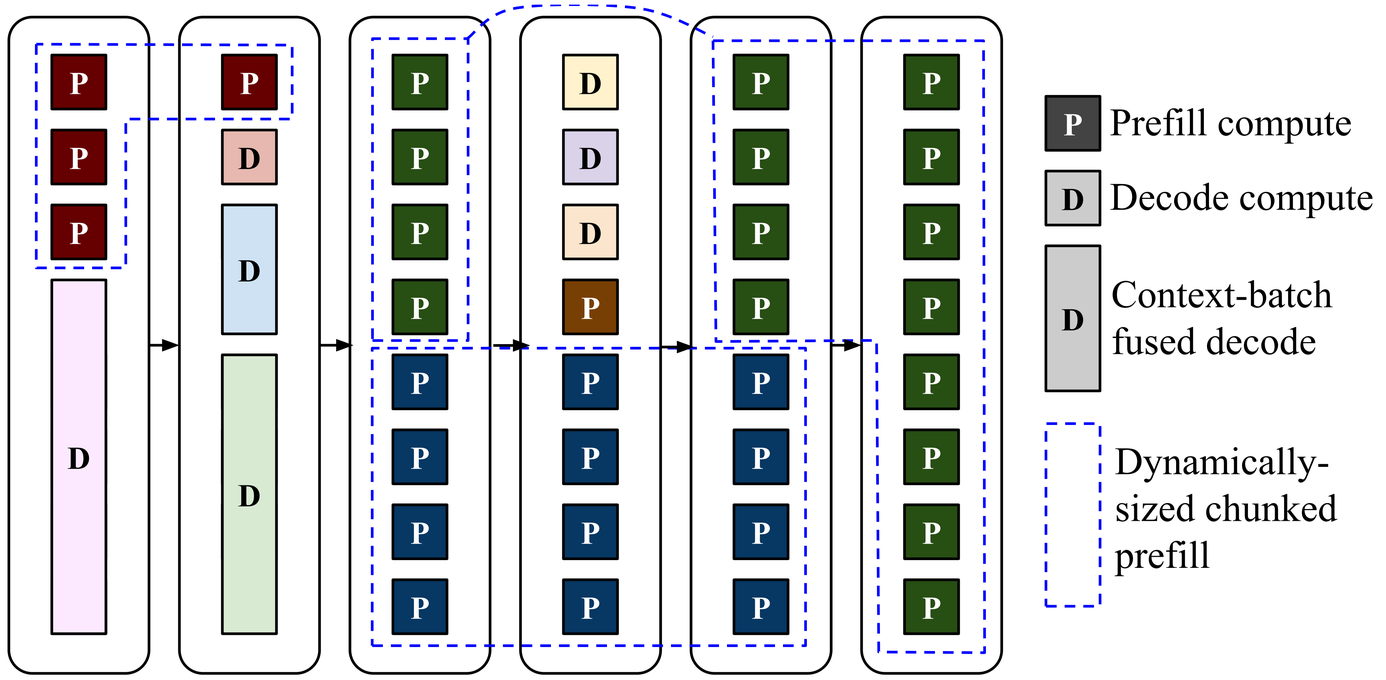
External draft: add additional pipe stage in SHIP to hold drafter

see paper for more details

Dynamic and Balanced Pipelines

Avoiding pipeline bubbles and maintaining stable throughput and latency

Dynamic Chunked Prefill | **Fused Context-Batch** | **Capacity-Filling Prefill**



Example six-partition SHIP. Boxes sharing color = tokens from same request.

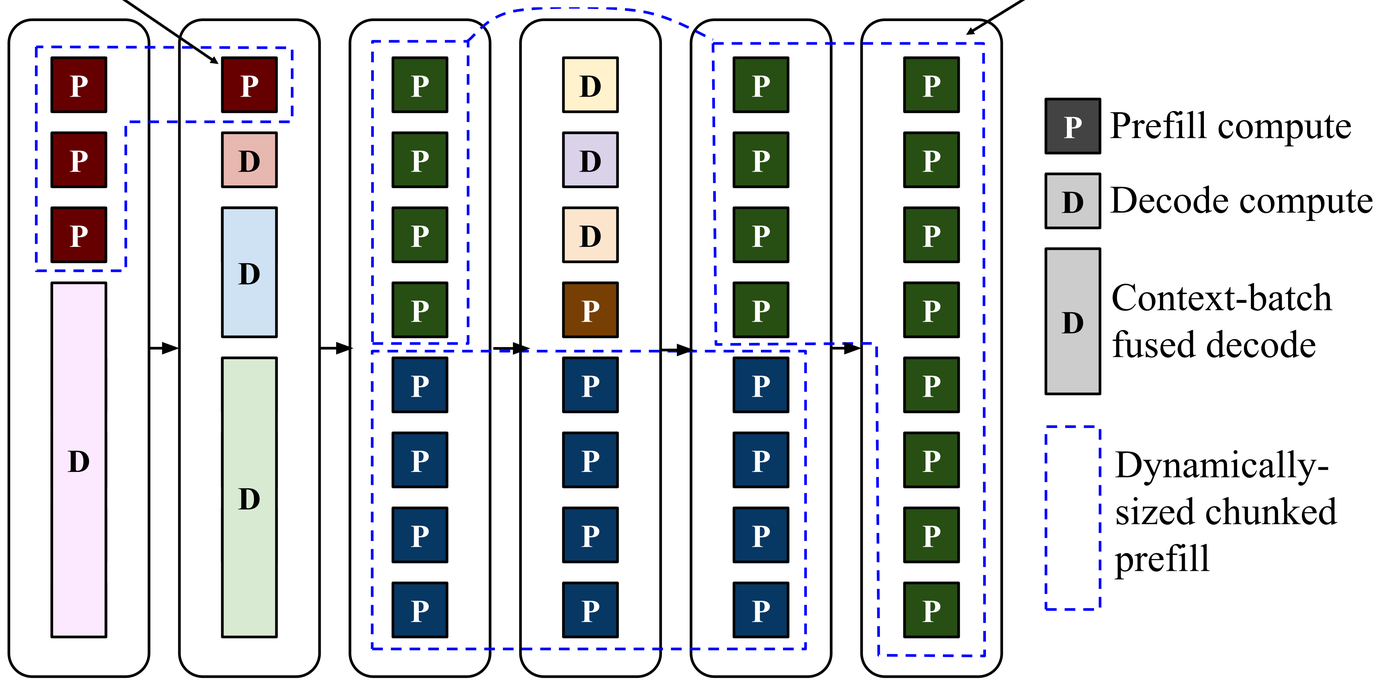
Dynamic Chunked Prefill

Avoiding pipeline bubbles and maintaining stable throughput and latency

Prefill chunk minimum size of 1-2 tokens thanks to high memory-to-compute BW

Each pipe stage has a fixed compute budget dynamically allocated to prefill and decode tokens.

Runtime scheduler can form larger chunks when needed, such as to reduce TTFT for long context prompts



Example six-partition SHIP. Boxes sharing color = tokens from same request.

Fused Context-Batch

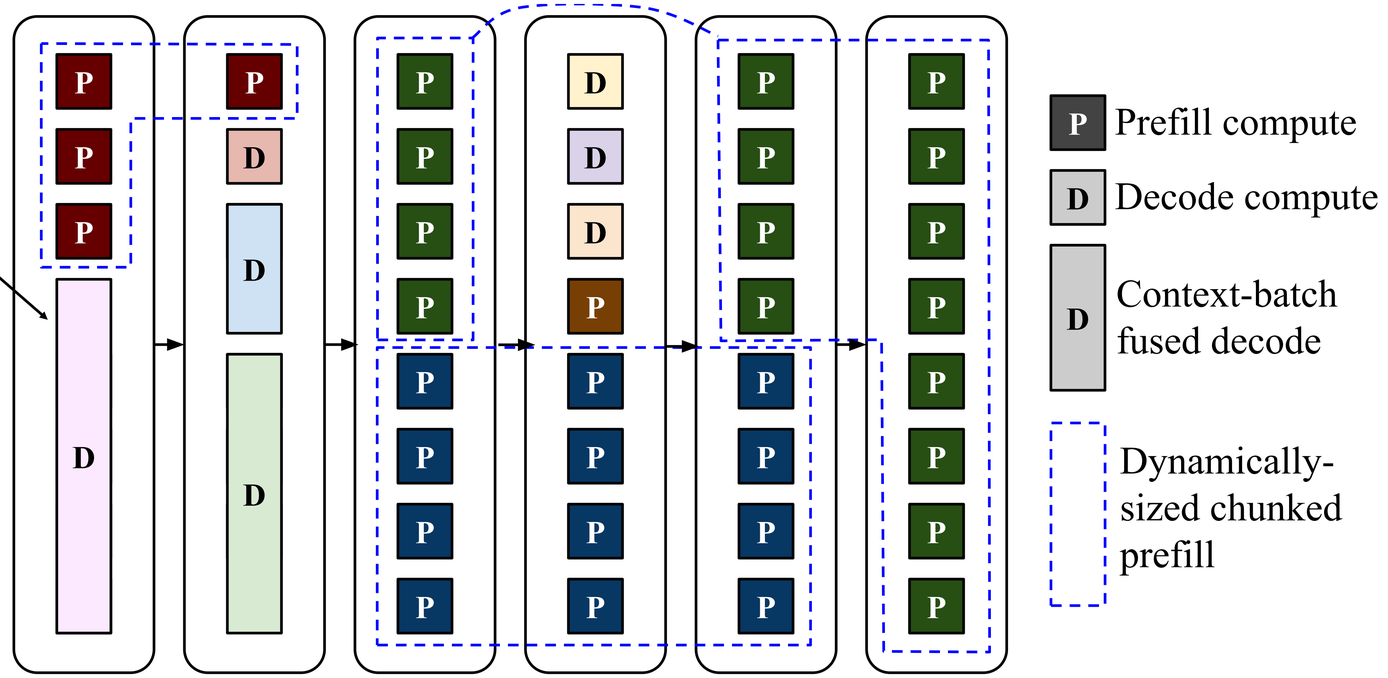
Avoiding pipeline bubbles and maintaining stable throughput and latency

Attention context length and batch are fused into a single dimension.

Pipe stages maintain fixed latency regardless of context lengths and batch sizes.

As context increases, batching is proportionally reduced (and vice versa).

No stage is stalled waiting for longer context tokens ahead in pipeline!



Example six-partition SHIP. Boxes sharing color = tokens from same request.

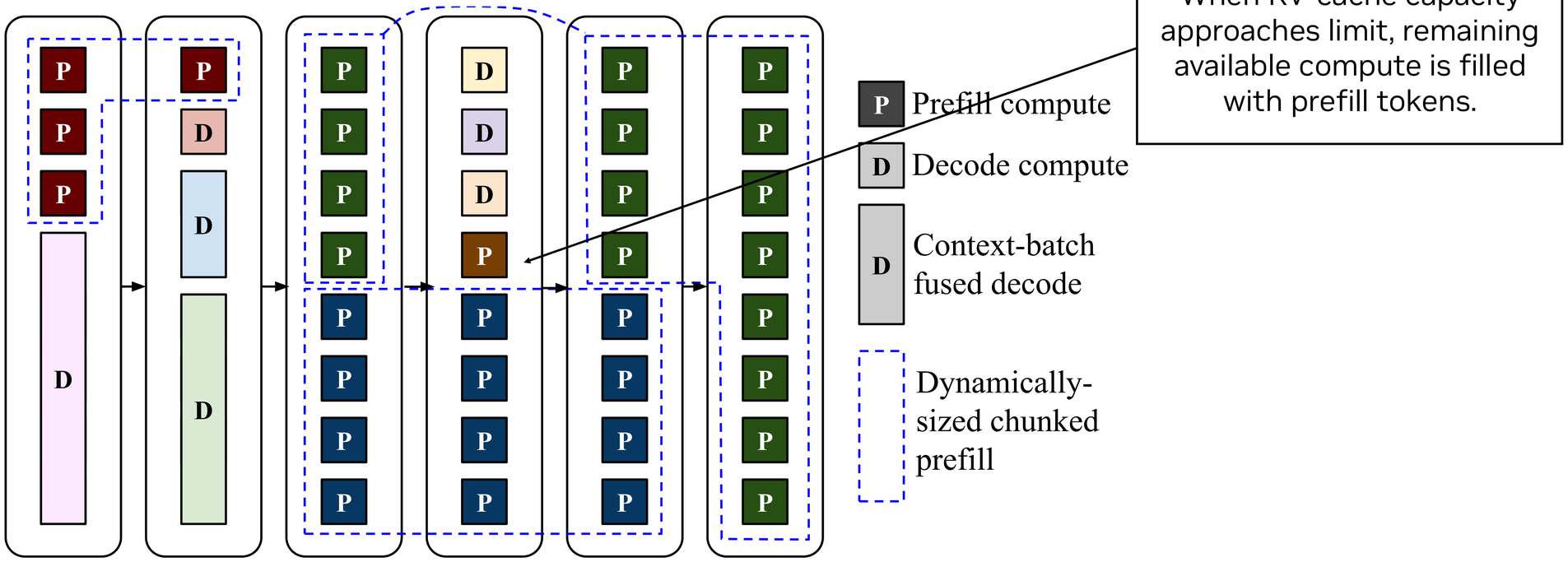
Capacity-Filling Prefill

Avoiding pipeline bubbles and maintaining stable throughput and latency

Runtime scheduler prioritizes decode tokens over prefill.

Decode, unlike prefill, is KV-cache capacity constrained.

When KV-cache capacity approaches limit, remaining available compute is filled with prefill tokens.

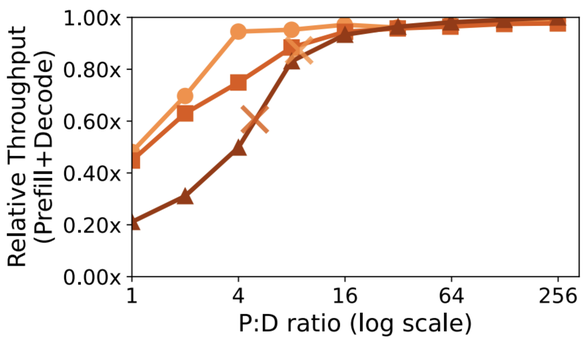


Example six-partition SHIP. Boxes sharing color = tokens from same request.

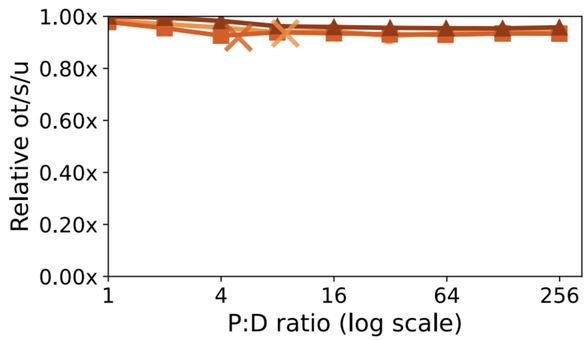
Results

Avoiding pipeline bubbles and maintaining stable throughput and latency

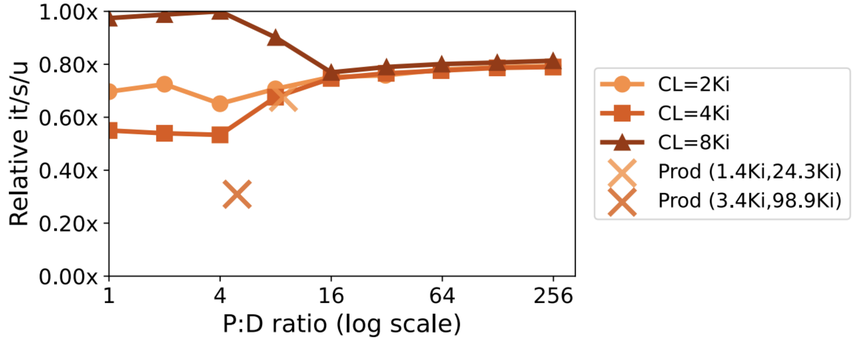
QWEN3-235B-A22B across Prefill:Decode Token Ratios (P:D) and Context Lengths (CL)



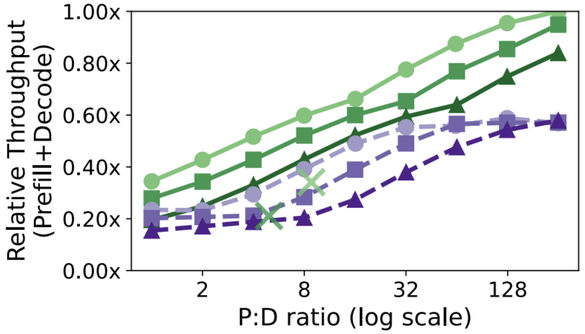
(a) LPU SHIP system throughput



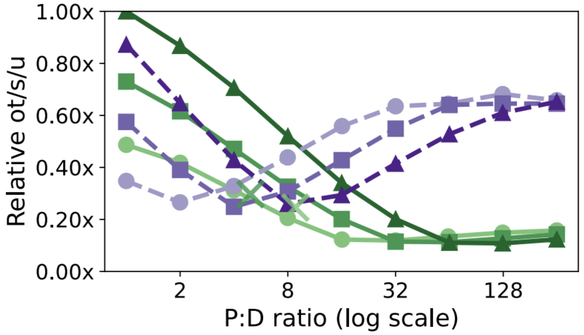
(b) LPU SHIP decode ot/s/u



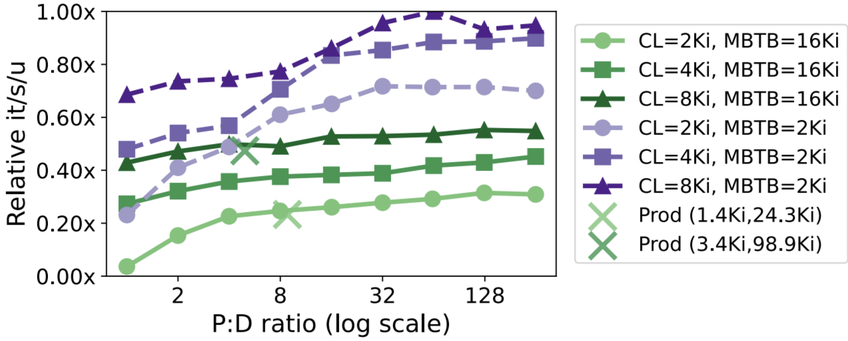
(c) LPU SHIP prefill it/s/u



(d) B200 DGX system throughput



(e) B200 DGX decode ot/s/u



(f) B200 DGX prefill it/s/u

see paper for detailed results explanations

More Topics Covered in Paper

Power & Cost

- LPU-only systems don’t reach peak efficiency achieved by NVIDIA GPUs
- Mitigate some of the gap with lower power & cost profiles

Benefits and challenges from deterministic/synchronous execution

- Power-aware scheduling
- Floating-point determinism

Previews of next-gen LPUs

- More memory, more C2C
- Rubin + LPUv3

THANK YOU!

THIS WORK WAS A TEAM EFFORT WITH CONTRIBUTIONS FROM
THE ENTIRE GROQ TEAM!

OUR JOURNEY CONTINUES AT **NVIDIA** 🚀

NVIDIA