TokenWeave:

Accelerating Tensor Parallelism LLM Inference Through Efficient Compute-Communication Overlap

Raja Gond, Nipun Kwatra, and **Ram Ramjee**

Microsoft Research, India

Microsoft

Compute-Communication Overlap

- Idea: Split compute; while one task computes, the other communicates

- Micro-split: Matmul is split into tiles; fuse communication with tile compute
  - CoCoNET [ASPLOS'22], TPU [ASPLOS'23], TileLink [MLSys'25], Comet [MLSys'25]

- Macro-split: Split input batch; compute one with communication of another
  - PipeMOE [Infocom'23], Lancet [MLSys'24], Nanoflow [OSDI'25] , DeepSeek TWO-batch

- Overlap NOT enabled by default for tensor-parallel inference in vLLM/SGLang

➢ Communication takes significant resources, lengthening compute time
➢ Splitting has significant overheads & requires large batches for gains

TokenWeave: Summary of Contributions

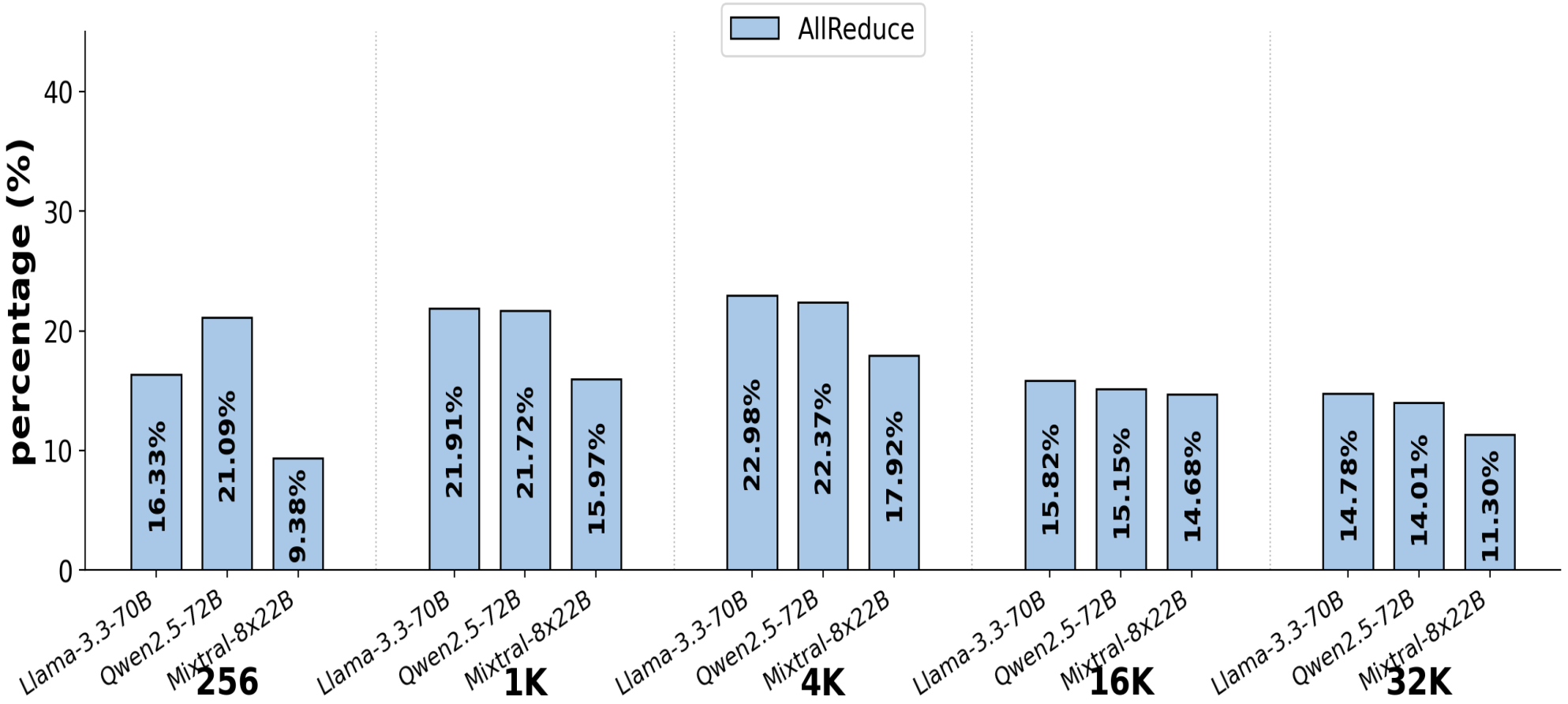
- TokenWeave overlaps compute with communication & normalization
  1. New AllReduce-RMSNorm Kernel
  2. Symmetric memory + Multimem/NVLS to reduce resources for comm.
  3. Wave-aware macro-split to reduce splitting overhead

- Up to **1.28×** speedup in latency, **1.19×** higher end-to-end throughput

- Results on H100 DGX/B200 DGX over a strong **vllm_multimem** baseline (vLLM enhanced with Symmetric memory & NVLS support)

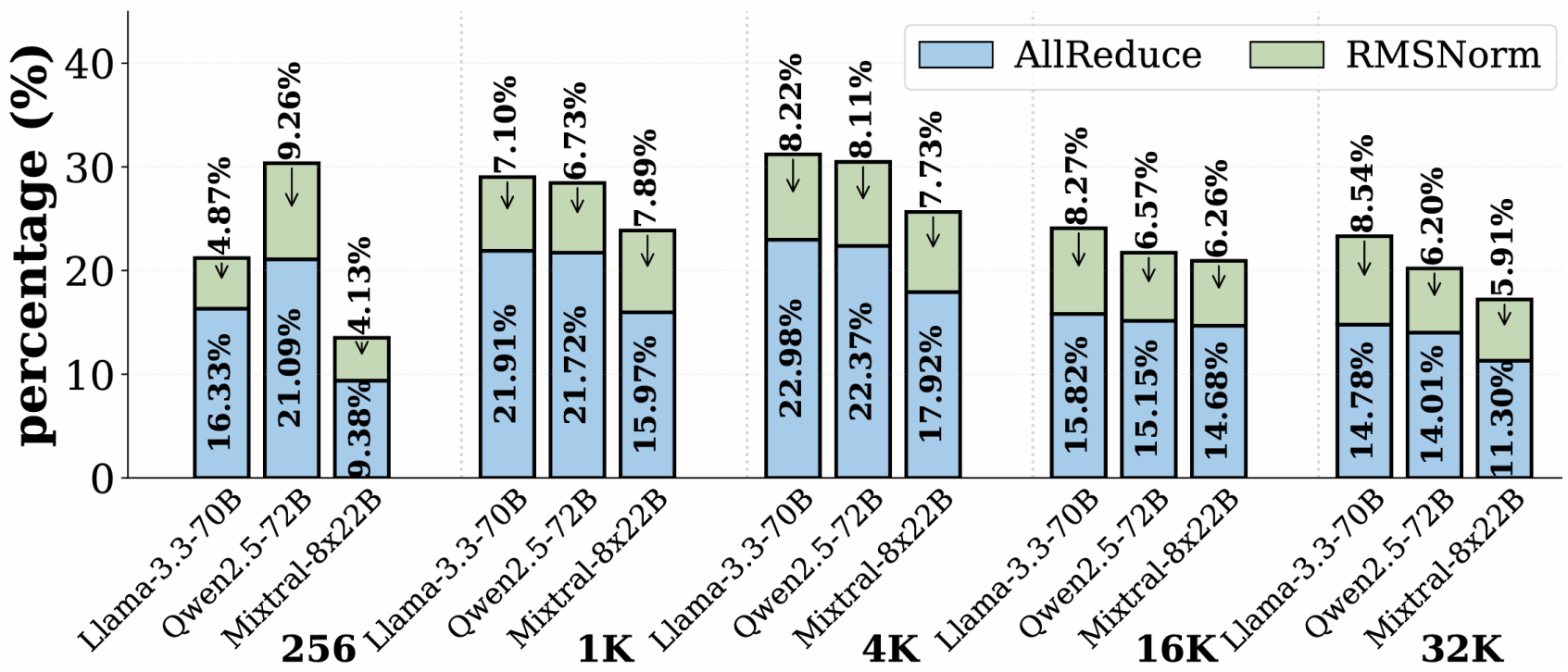
- Code open-sourced at github.com/microsoft/tokenweave

Communication Overhead



➢ 9-23% overhead on 8xH100 DGX with high bandwidth NVLINK

RMSNorm: Surprising source of significant overhead!



➢ 4-9% overhead for RMSNorm on 8xH100 DGX

Why is RMSNorm overhead so high?

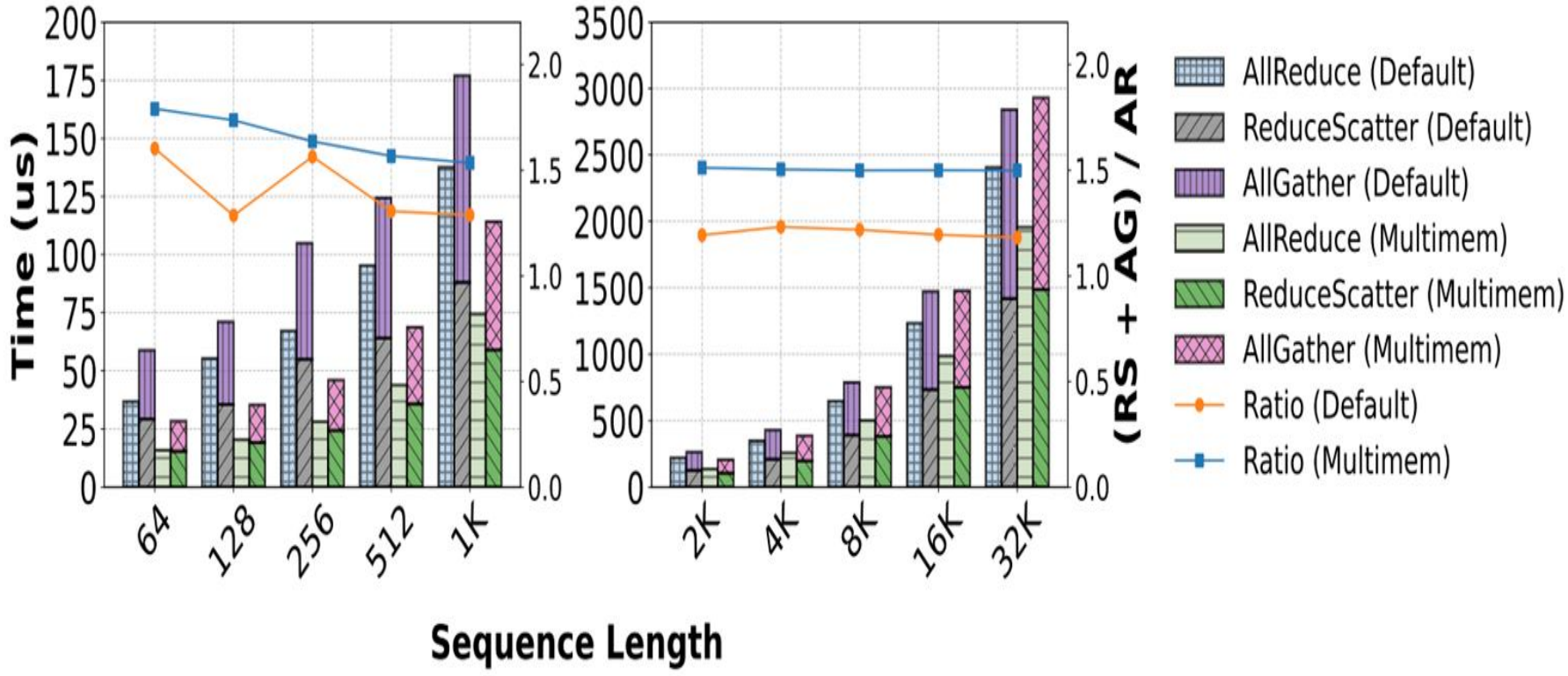
- Tensor-parallel inference today: AllReduce → RMSNorm
  - RMSNorm does redundant work operating on Full Tensor on each GPU

- Idea: ReduceScatter → RMSNorm →AllGather
  - RMSNorm does 1/G work (G GPUs) and mathematically equivalent

- Slower!

➢ AllReduce → RMSNorm is more performant despite redundancy!

ReduceScatter + AllGather vs AllReduce



➢ ReduceScatter+AllGather is 1.2–1.7× latency of AllReduce on 8 × H100

1. New AllReduce-RMSNorm Kernel

AllReduce → RMSNorm into ReduceScatter → RMSNorm →AllGather

- Idea: Avoid HBM access by leveraging Symmetric Memory + MultiMem/NVLS support in modern GPUs (Hopper/Blackwell/…)

i. ReduceScatter using multimem_ld_reduce_add PTX instruction

- In-network reduction & store result directly in *register*

ii. RMSNorm using *only register/shared memory* (avoid HBM read/write)

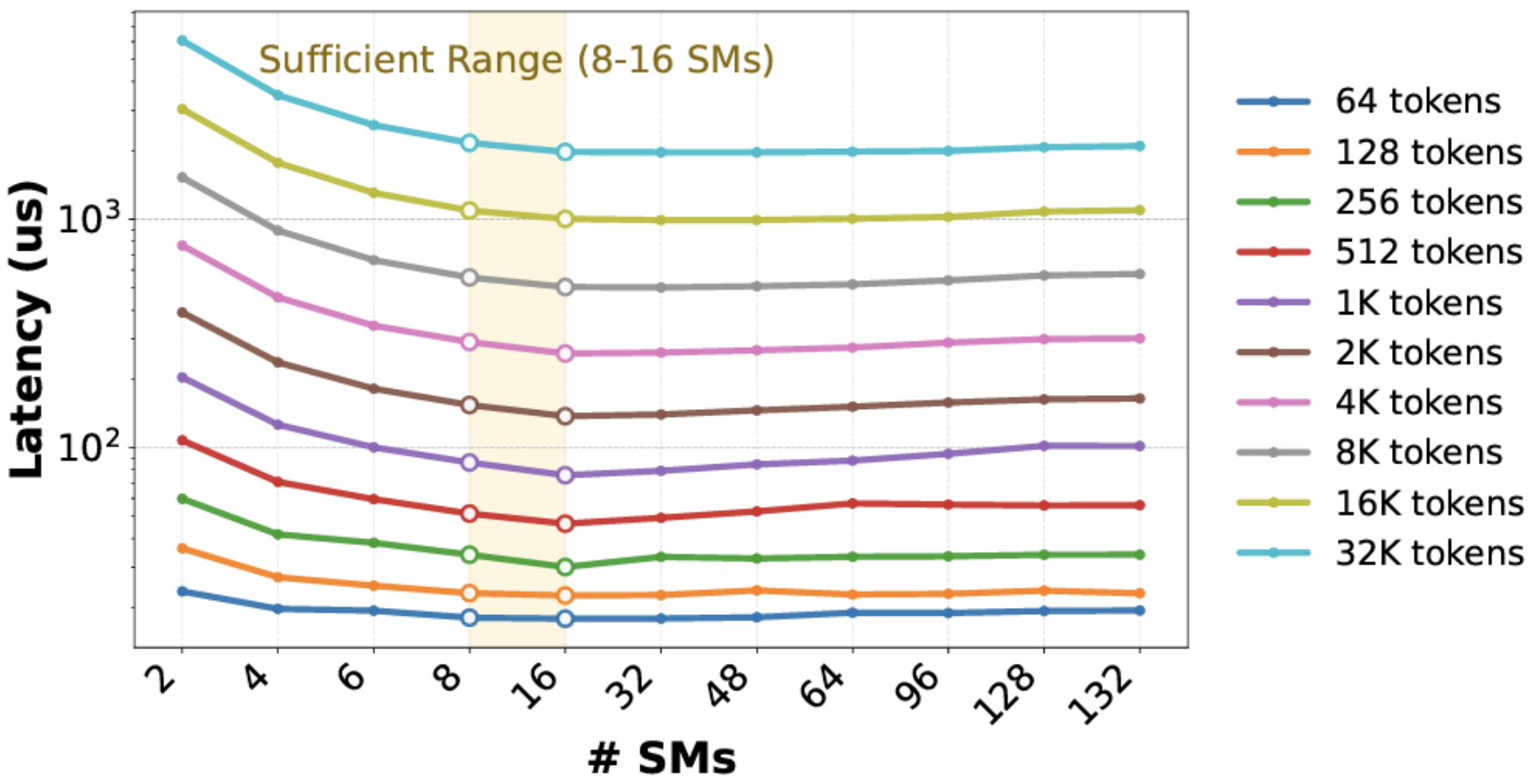
iii. AllGather using multimem_st PTX

- Read from *register* (avoid HBM read) and write into remote HBM

Saves 2N/G reads and N/G writes per GPU (N: #tokens, G: #GPUs)

➢ AllReduce-RMSNorm 1.3–1.4× AR+RMSNorm & 4+% end-to-end gains

2. Symmetric Memory + Multimem/NVLS benefit: Reduced compute resources for communication



- 16 Streaming Multiprocessors (SMs) for optimal performance
- For overlap, 2-8 SMs sufficient
  - Only need to be faster than the overlapped compute

- Only 2-8 SMs sufficient for Communication + Normalization overlap!

3. Reducing Splitting Overhead

Example: 100 SMs, Matmul with 250 Cooperative Thread Arrays (CTAs)

Full

- Matmul with 250 CTAs
- 2 Full waves, last wave 50% occ.
- Wasted fraction = 0.5 / 3 = 16.6%

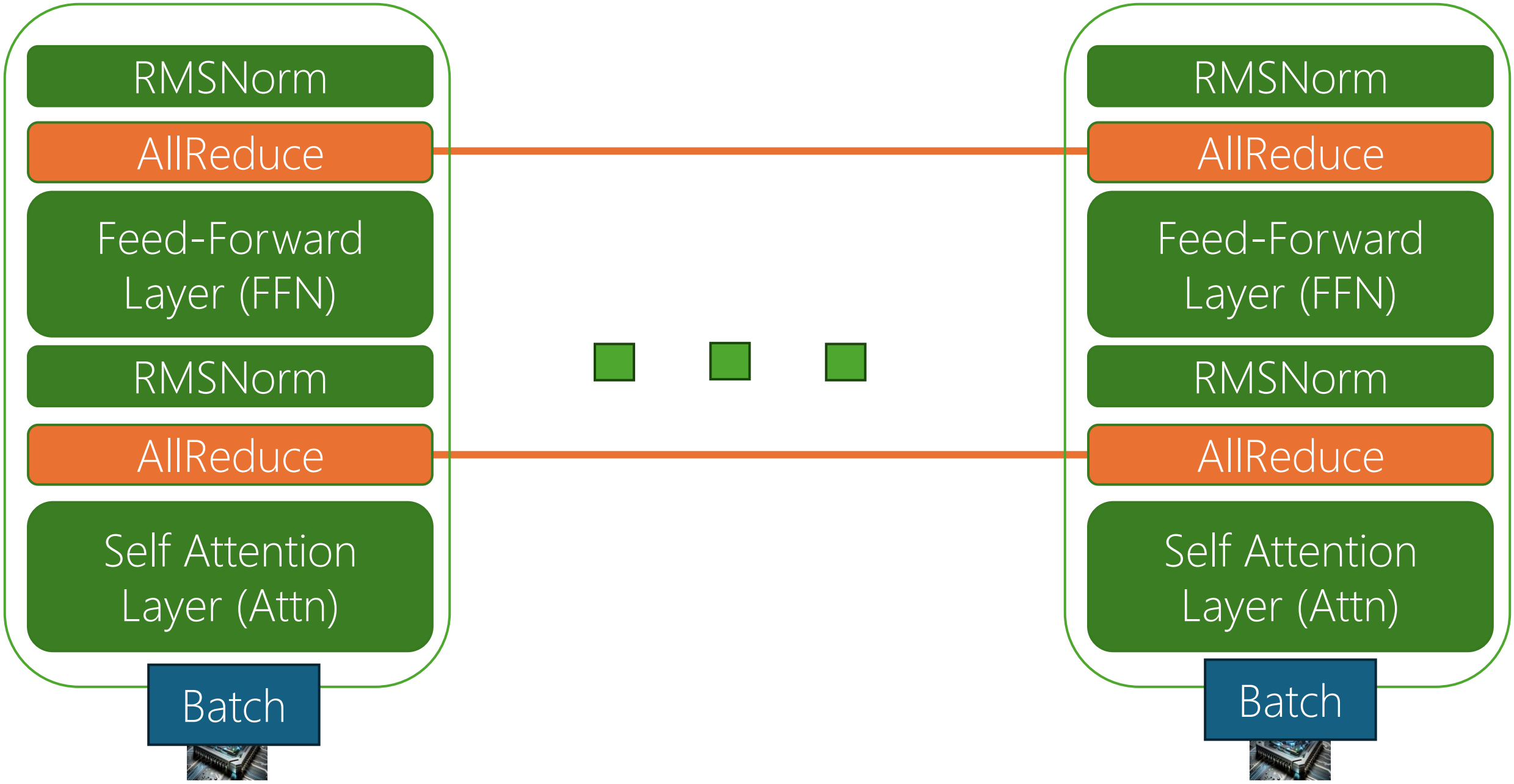
Two-way Split

- 2 Matmuls with 125 CTAs each
- 1 Full wave, last wave 25% occ.
- Wasted fraction = 0.75/2 = 37.5%

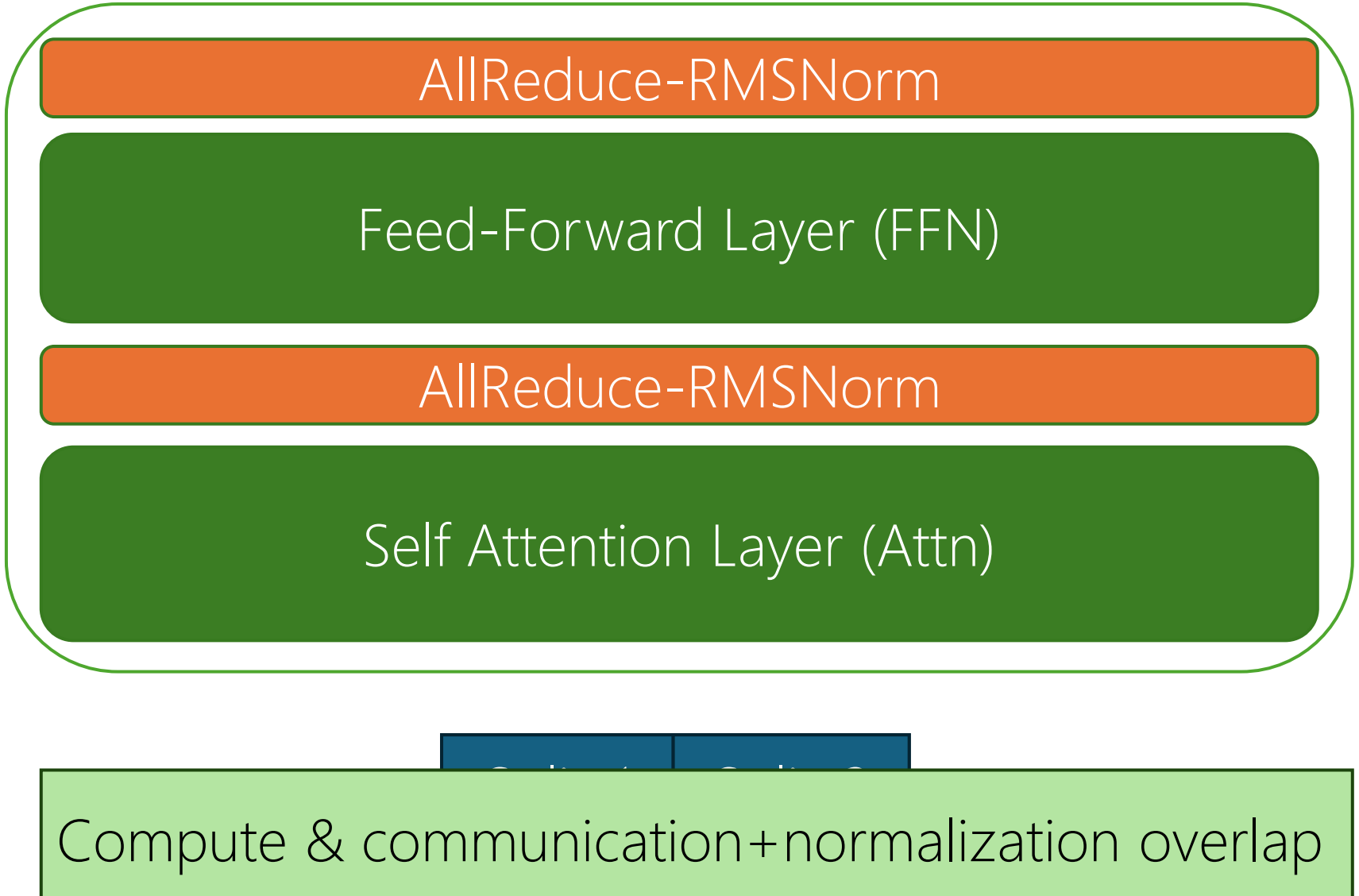
Instead: 2-split of 100 CTAs + 150 CTAs; Wasted fraction = 0.5/3=16.6%

➢ Wave-aware split: Ensures no additional quantization overhead over Full

Tensor-Parallelism today



TokenWeave

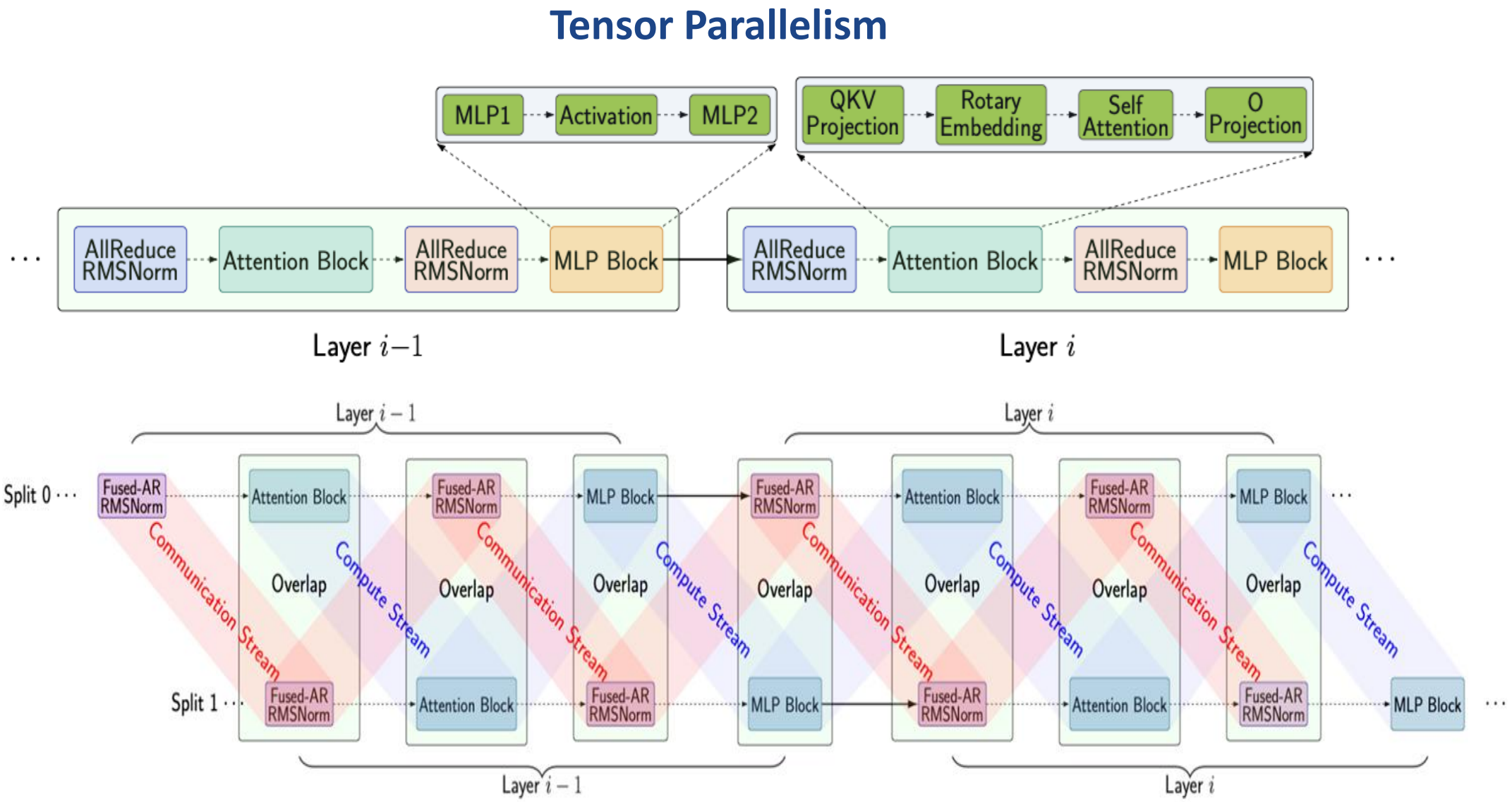


Splitting Approach:

- Don’t split if less than 1-wave
- If more than 1-wave, use wave-aware two-split

Staggered Execution

TokenWeave: Implementation



Experimental Setup

Hardware:

- **Primary:** 8×H100 NVIDIA DGX (NVLink4 +NVSHARP)
- **Also:** 4×H100 & 8×B200

Models:

- Llama-3.3-70B (dense)
- Qwen2.5-72B (dense)
- Mixtral-8x22B (MoE)

Workloads:

- **Real-world:** ShareGPT, arXiv traces
- **Synthetic:** fixed (input, output) lengths

Baselines:

- **vLLM-Multimem:** vLLM + optimized AllReduce
- **vLLM-nocomm:** communication removed (theoretical performance lower bound)

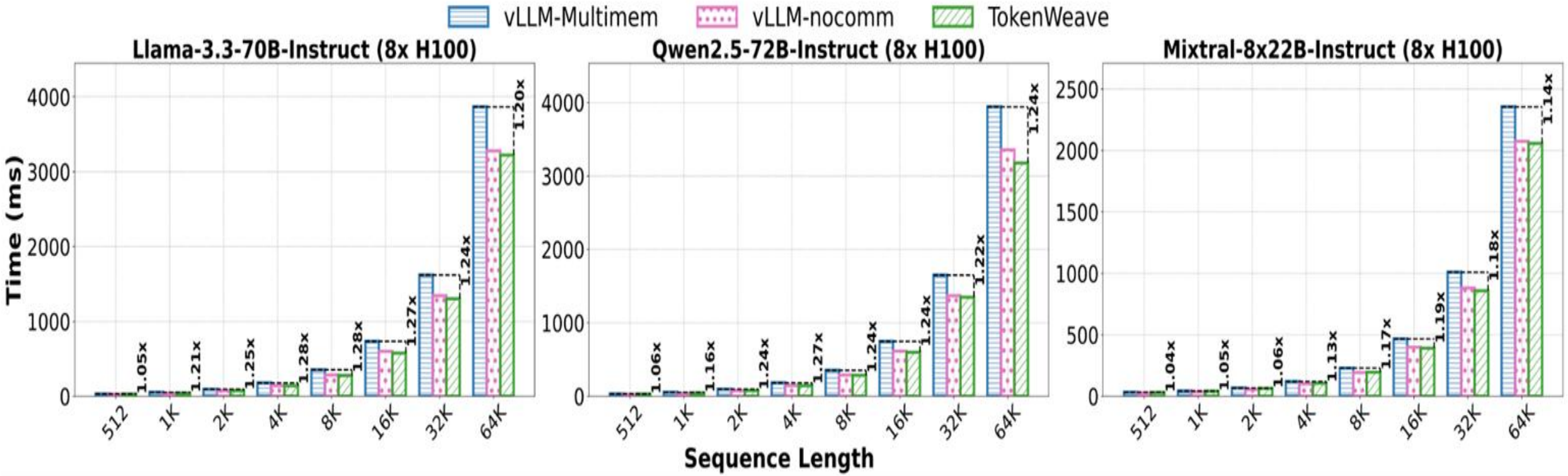
Also (see paper):

- vLLM-Default: vLLM v0.8.5 (no overlap)
- TileLink: state-of-the-art tile-level overlap
- NanoFlow: prior coarse-grained overlap system

Implementation:

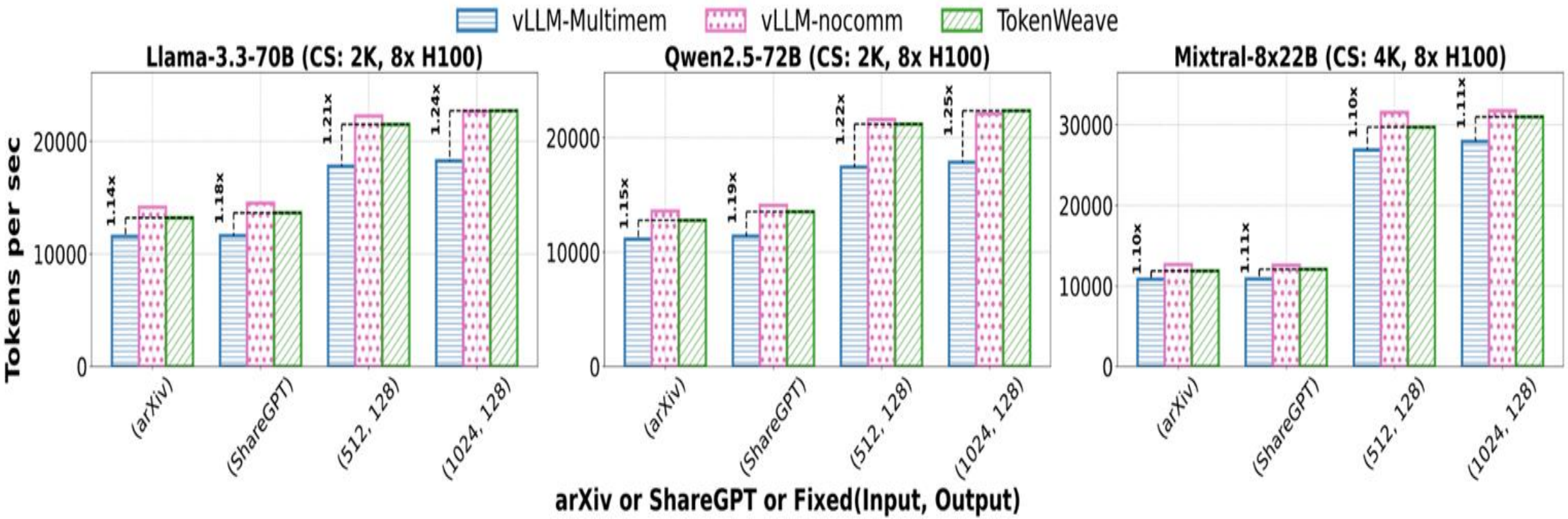
- vLLM V1 (v0.8.5), PyTorch 2.6.0, CUDA 12.4, Triton 3.2.0
- github.com/microsoft/tokenweave

Evaluation: Latency



- Overall: TokenWeave achieves up to 1.28× speedup
- 20+% gains at 1K+ tokens in Dense and 13-19% for 4K+tokens in MoE
- At sequence lengths ≥ 4K+, TokenWeave matches/beats vLLM-nocomm

Evaluation: End-to-end Throughput



➢ TokenWeave achieves up to 1.19× speedup on sharegpt & arxiv traces

TokenWeave: Summary

- Challenges

- ✗ RMSNorm adds 4–9% overhead
- ✗ Communication uses many SMs, lengthening compute
- ✗ Splitting overhead overwhelms gains at small batch sizes

- Solution

- ☑ New AllReduce–RMSNorm kernel
- ☑ 2–8 SMs for both comm + norm using symmetric memory + NVLS
- ☑ Wave-aware at most two-way token split

➢ Up to 1.28× speedup in latency, 1.19× higher end-to-end throughput

➢ Code: github.com/microsoft/tokenweave