Serving Intelligence: The Unseen Limits of LLM Inference and the Hardware Race to Overcome Them

Karu Sankaralingam, NVIDIA

With contributions from Michael Davies and Neal Crago

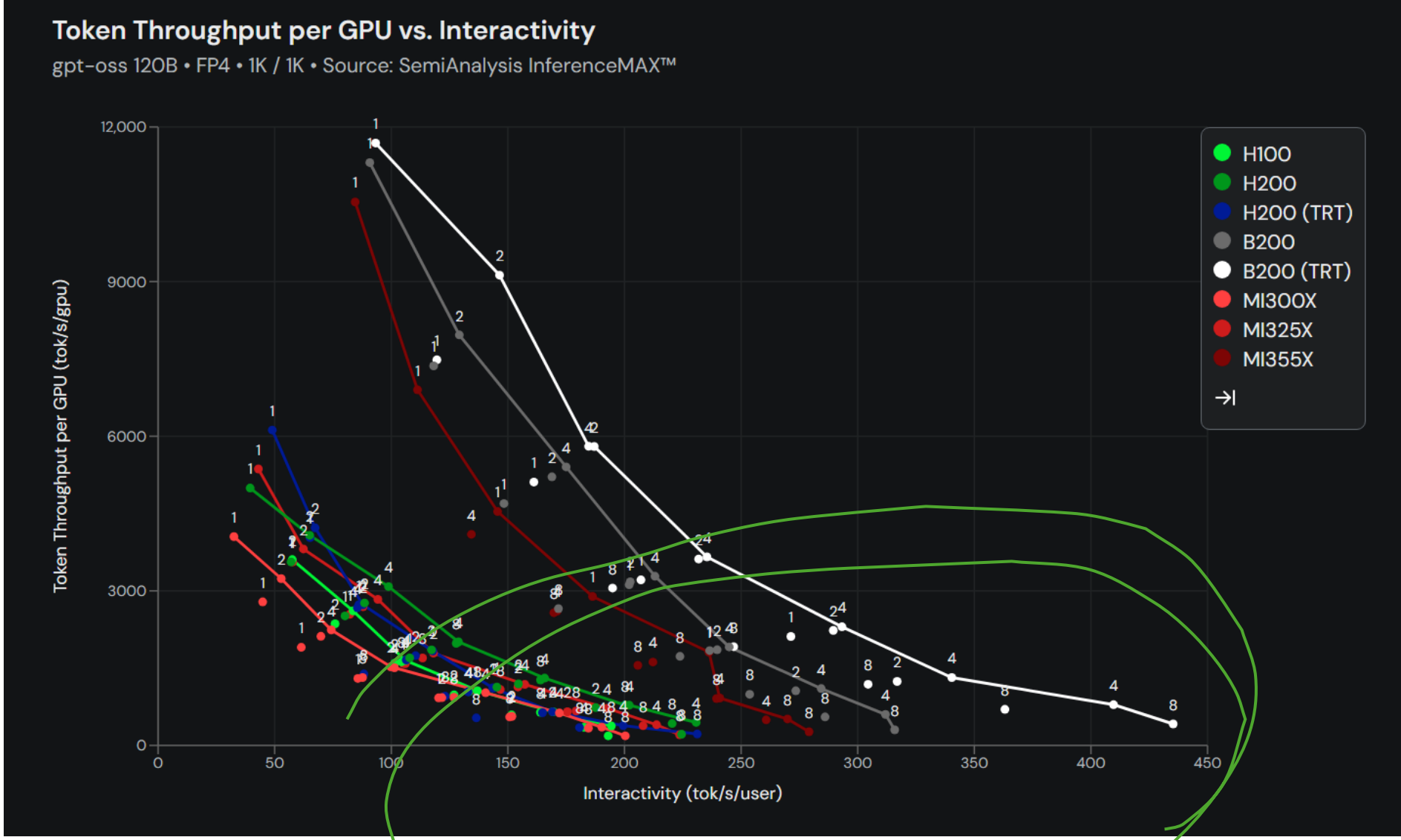
What’s this talk about?

- Large Language Models are rapidly becoming a new, universal application layer.
- But their immense cost and latency are the single greatest barriers to their ultimate potential.
- The entire field is in a race to optimize… but are we at risk of solving the **wrong problem.**
- Today, we'll answer two questions:
  1. **What is the absolute performance ceiling for current and near-future hardware?**
  2. **What must we do to achieve the next 10X, and where must algorithms take over?**

One Minute Summary of Talk

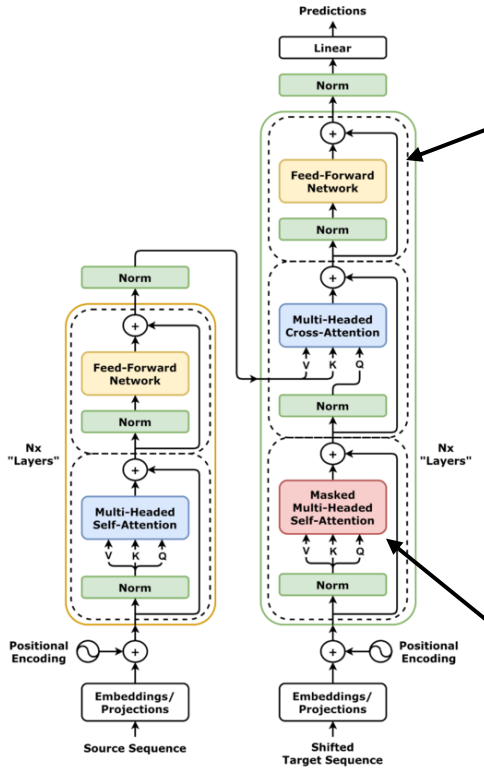
- LLMs stress hardware's **capacity, bandwidth, compute and collective communication** capabilities.
  - *Architect's Dream or Nightmare workload*.
  - *HW must provide caching, prefetching, high bandwidth, fast synchronization*
- Established technologies can easily get to 5000 user tokens/second
  - Large power/energy savings by trading off very little user latency
  - 5% to 10% "slower" gets you 5X more efficiency
- Path to 50,000 and 100,000 needs:
  - New algorithms and **flexible** hardware – caching, prefetch, synchronization (within and between chips)
  - For the research community: focus less on what AI chips can do today

How far can be push this? What are the “true” limits?



Source: SemiAnalysis

Computer Architect's view of an LLM



BERT: 2017

MLP or MoE

GQA or MLA

Today

LLM	Architecture	Parameter Count	Year
Llama 3	GQA+MLP	8B, 70B, 405B	2024
Llama 4	GQA+MoE*	109B, 400B	2025
DeepSeek V3	MLA+MoE*	671B	2025
Qwen 3	GQA+MoE*	4B, 30B, 235B	2025
Kimi K2	MLA+MoE*	1T	2025
GPT-OSS	GQA+MoE	20B, 120B	2025

- Software pipelining, caching, prefetch, synchronization, capacity all come into play…because
- The sizes of these things are huge and **a single model is often spread across 8 to 128 GPUs** to generate a single token

```
for each layer:
    read all weights, read KV Cache  ← GBs of data
    perform work                     ← GFLOPs
        matmul, softmax, etc..
```

How can we reason about this as architects?

```
for each layer:
    read all weights, read KV Cache  ← GBs of data
    perform work                     ← GFLOPs
        matmul, softmax, etc..
```

Liminal: A Performance Model

Hardware Parameters	
N	Number of chips
FLOPS	Scalar and Tensor FLOPS (Peak)
BW	Total memory bandwidth
Workload Parameters	
L	Number of Layers
F_{op}	FLOPS in operator
M_{op}	Bytes loaded for operator

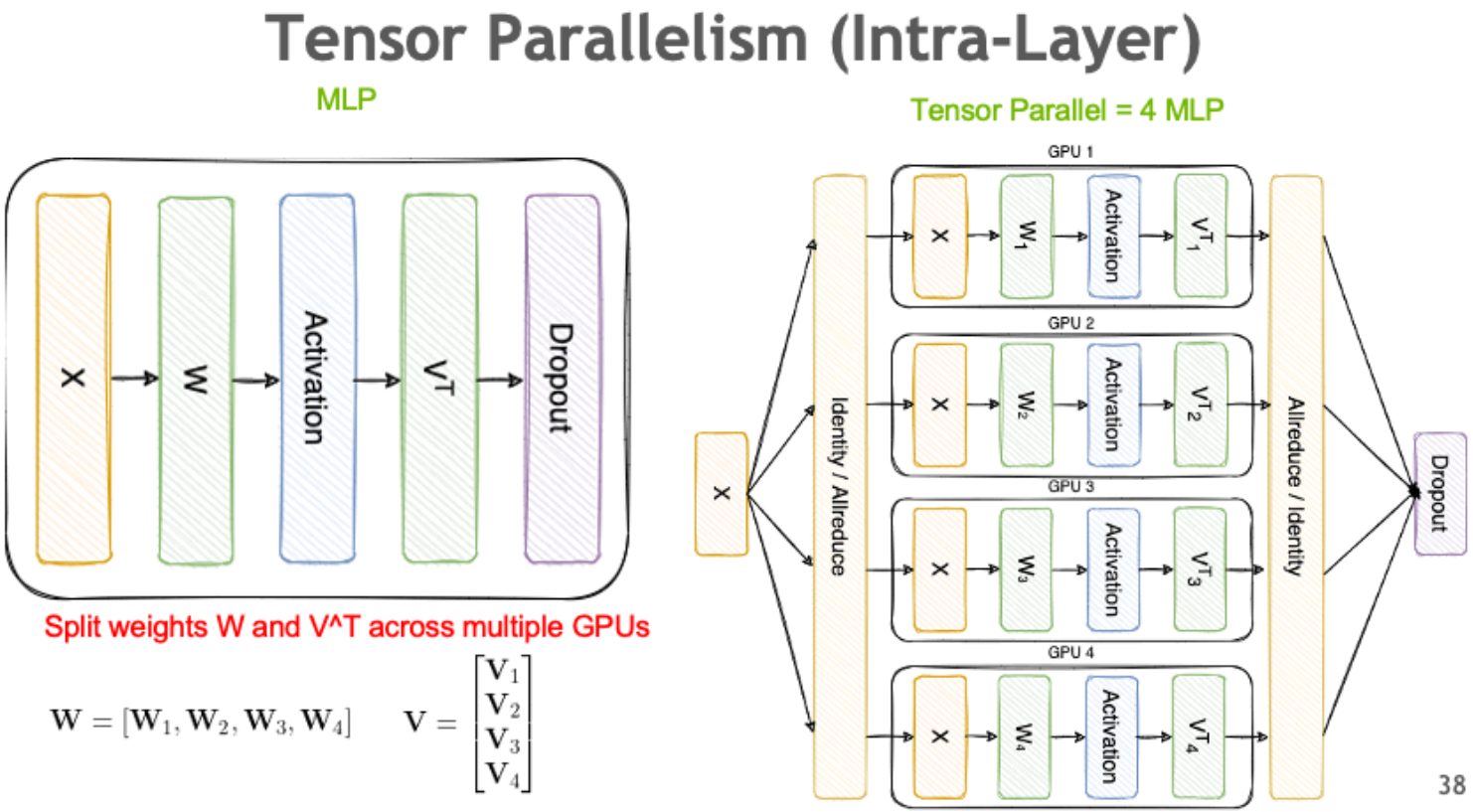
$$T_{Batch} = max\{T_{Compute}, T_{Mem}\}$$

$$\sum_{op} \frac{F_{op,tensor}}{N * FLOPS_{tensor}} + \frac{F_{op,scalar}}{N * FLOPS_{scalar}}$$

$$\sum_{op} \frac{M_{op}}{N * BW}$$

$$UTPS = \frac{1}{T_{Batch}} \qquad STPS = \frac{P * B}{T_{Batch}}$$

What about multiple chips?



There is also Expert Parallel, Context Parallel, and Pipeline Parallel. A mapper decides on this strategy, assigns data and code to different chips

What does this mean for chip architecture?

a. Essentially, we need to know **number** of **inter-chip collectives**
b. They become serialization points

Liminal: A Performance Model

Hardware Parameters	
N	Number of chips
FLOPS	Scalar and Tensor FLOPS (Peak)
BW	Total memory bandwidth
$T_{TP,N}$ $T_{PP,N}$	Latency of Collectives
Workload Parameters	
L	Number of Layers
F_{op}	FLOPS in operator
M_{op}	Bytes loaded for operator
S_N	# of collectives

$$T_{Batch} = max\{T_{Compute}, T_{Mem}\} + T_{Exposed}$$

$$\sum_{op} \frac{F_{op,tensor}}{N * FLOPS_{tensor}} + \frac{F_{op,scalar}}{N * FLOPS_{scalar}}$$

$$\sum_{op} \frac{M_{op}}{N * BW}$$

$$T_{TP,N} * S_N * L + T_{PP} * P$$

$$UTPS = \frac{1}{T_{Batch}} \qquad STPS = \frac{P * B}{T_{Batch}}$$

We now have enough to understand performance

Capacity	Memory Bandwidth	Synchronization Delays	Compute

Performance Metrics: User Tokens per Sec, System Tokens per Sec

Application “inputs”: Batch Size, Context Length, The DL Model

Memory Capacity

			GB			
Model	**Active**	**# Param**	**T=4K, B=1**	**T=4K,B=32**	**T=128K, B=1**	**T=128K, B=32**
Llama3-8B	7B	7B	7	14	14	262
Qwen3-4B	4B	4B	4	12	12	291
Qwen3-30B	3B	30B	28	34	34	220
Llama3-70B	68B	68B	64	84	84	704
Llama3-405B	402B	402B	375	406	406	1382
Llama4-Scout	15B	106B	99	110	110	482
Llama4-Maverick	15B	399B	372	383	383	755
DeepSeekV3	61B	694B	647	651	651	784
Qwen3-235B	21B	234B	218	229	229	594
Kimi-K2	43B	1T	965	970	970	1103
GPT-OSS-120B	5B	116B	108	112	112	252

LLM inference systems must have at least 100 GB of memory. To serve 32 users simultaneously, at least 482 GB is needed, with up to 1.1 TB and beyond required for very large models like Kimi K2.

8 to 128 GPUs
HBM3 memory
Fast switch

What can System Scale get you?

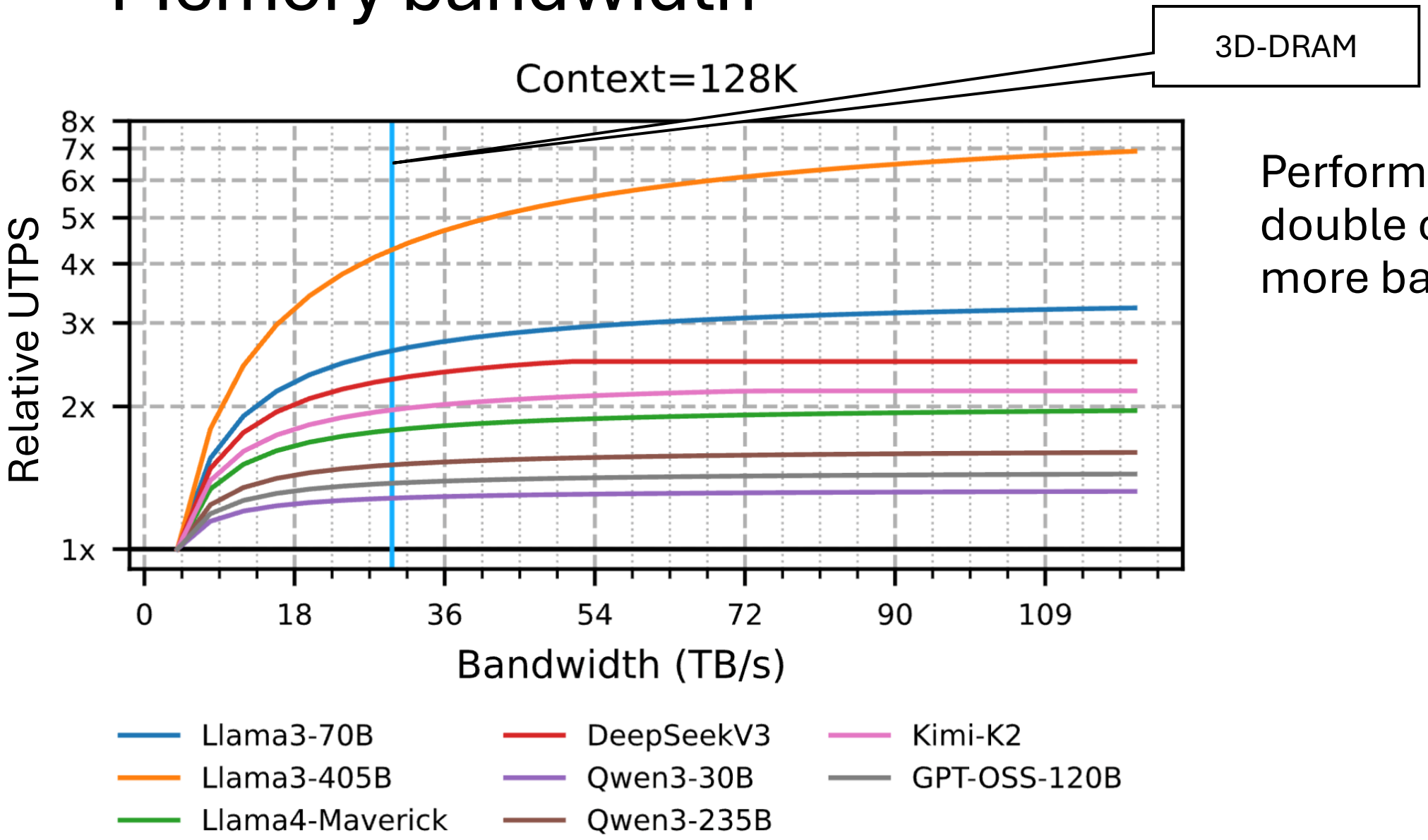
Model	UTPS, TP8	UTPS,TP128	STPS (UTPS), TP8	STPS (UTPS), TP128
Llama3-70B	381	2.1K	1.5K (43)	26K (42)
Llama3-405B	80	780	520 (43)	16K (42)
Llama4-Maverick	1.3K	3.8K	2.2K (67)	42K (42)
DeepSeekV3	522	2.4K	1.8K (63)	114K (42)
Qwen3-30B	2.7K	3.9K	5.2K (43)	86K (42)
Qwen3-235B	863	1.9K	2.0K (43)	43K (42)
Kimi-K2	-	**2.6K**	- (-)	**111K (42)**
GPT-OSS-120B	3.2K	**5.0K**	6.3K (43)	**115K (42)**

Latency/User Experience

Throughput

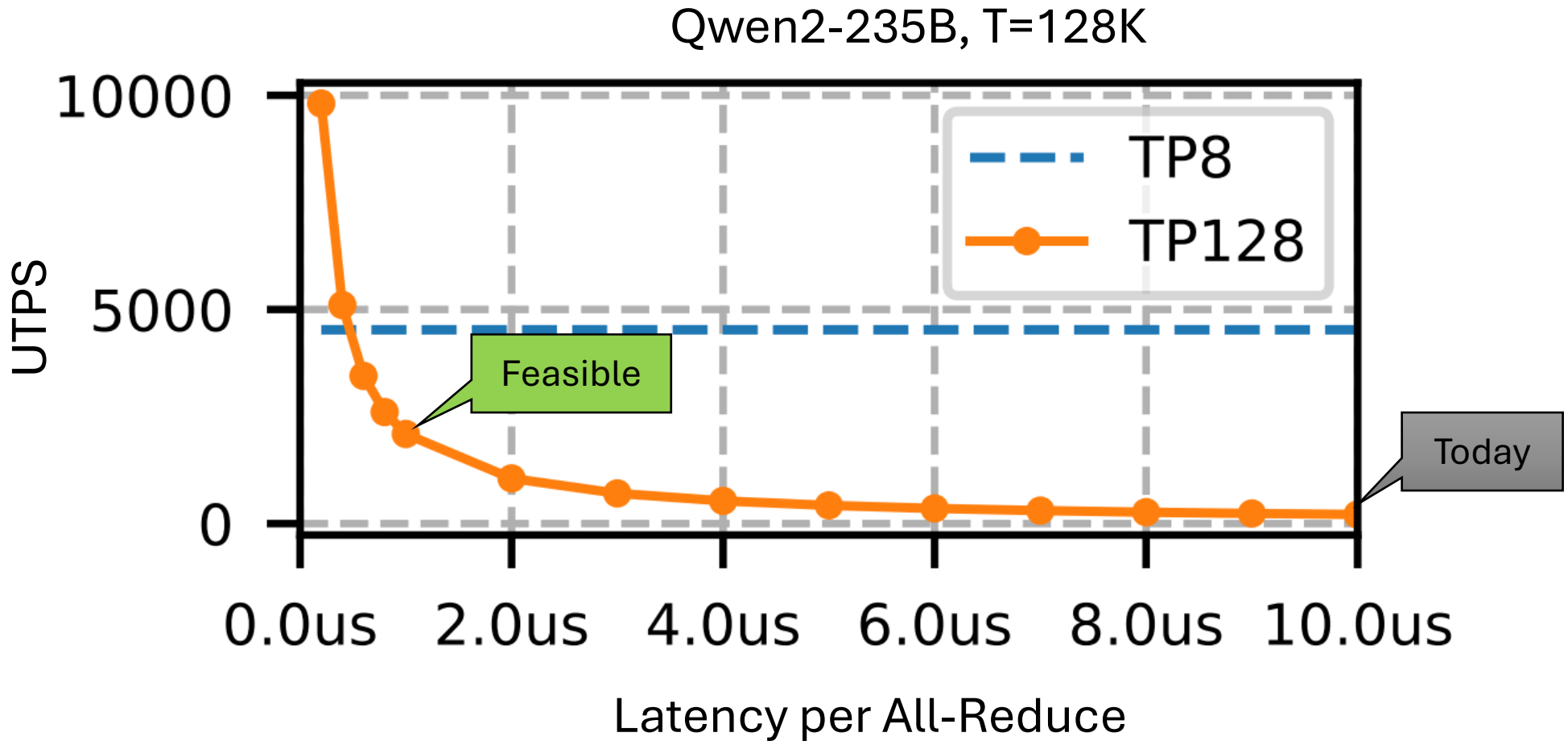
Well tuned systems with 128 GPUs can reach 2000 to 5000 UTPS
Large memory capacity, can provide >100K tokens/sec of **STPS**

Memory bandwidth



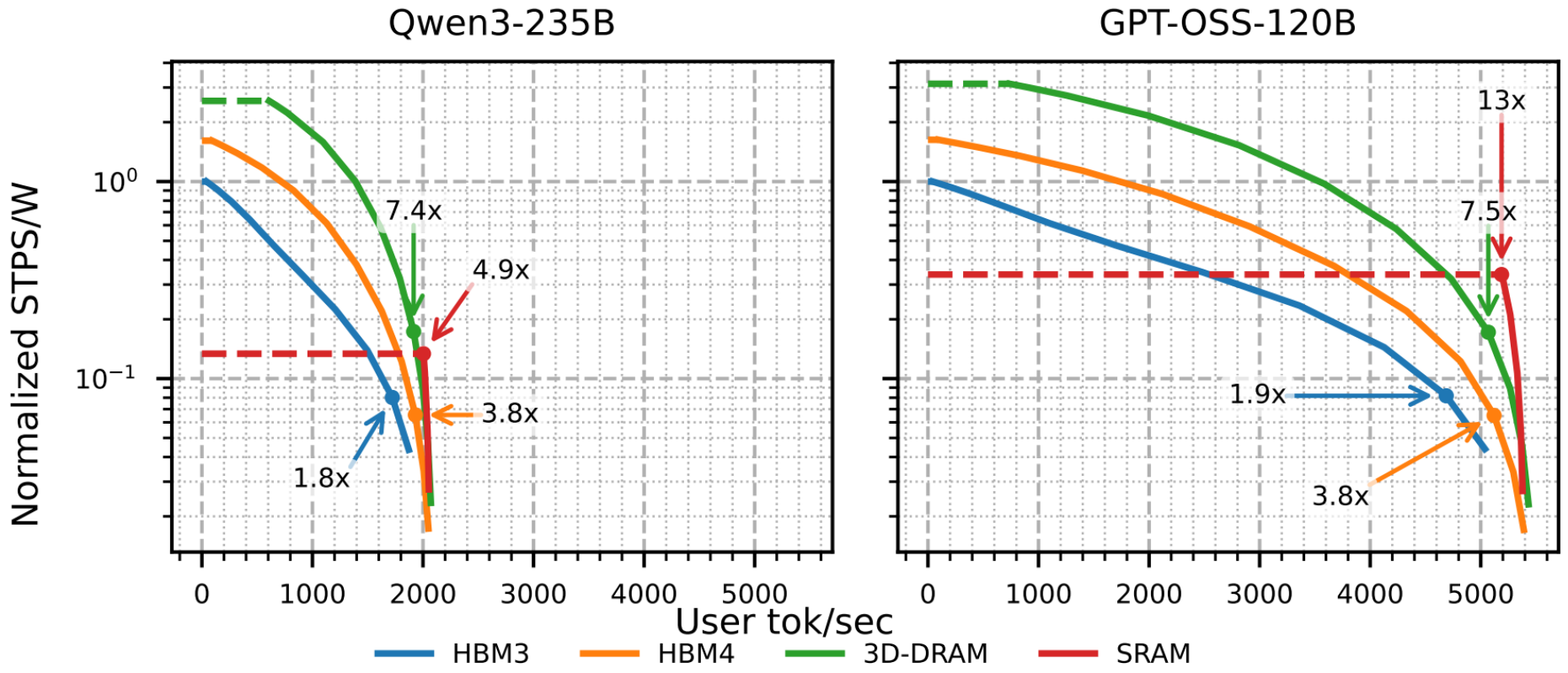
Performance can double or triple with more bandwidth…**BUT**

Synchronization Delays



Synchronization latencies dominate performance at large BW

What are some “easy” tradeoffs?



Substantial efficiency gains (7X) with marginal tradeoff in user responsiveness (10%)

Summary

- LLMs stress hardware's **capacity, bandwidth, compute and collective communication** capabilities.
  - *Architect's Dream or Nightmare workload*.
  - *HW must provide caching, prefetching, high bandwidth, fast synchronization*
- Established technologies can easily get to 5000 user tokens/second
  - Large power/energy savings by trading off very little user latency
  - 5% to 10% "slower" gets you >5X more efficiency
- Path to 50,000 and 100,000 tokens/second needs:
  - New algorithms and **flexible** hardware – caching, prefetch, synchronization (within and between chips)
  - For the research community: focus less on what AI chips can do today

Validation of Liminal