Networking 4 inference

Petr Lapukhov

Disclaimer:

- opinions are presenter’s and not of the employer.
- Information from open sources.

Agenda

- LLM inference overview
- KV caching
- Distributed inference
- Scale-up networks
- The latency/throughput tradeoff frontier
- Disaggregated inference
- The inference fabric picture

Teaser: scale-up, scale-out and so on

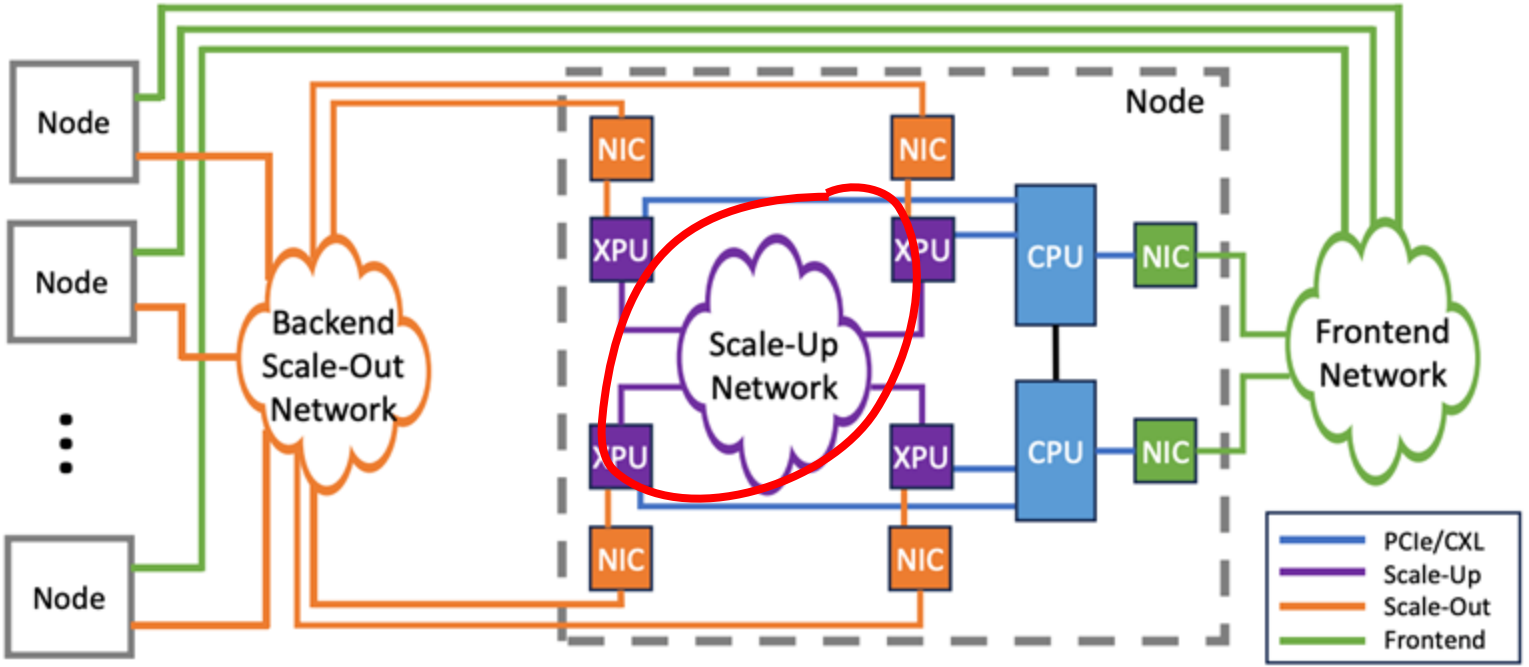


Image from UEC Spec: https://ultraethernet.org/wp-content/uploads/sites/20/2025/06/UE-Specification-6.11.25.pdf

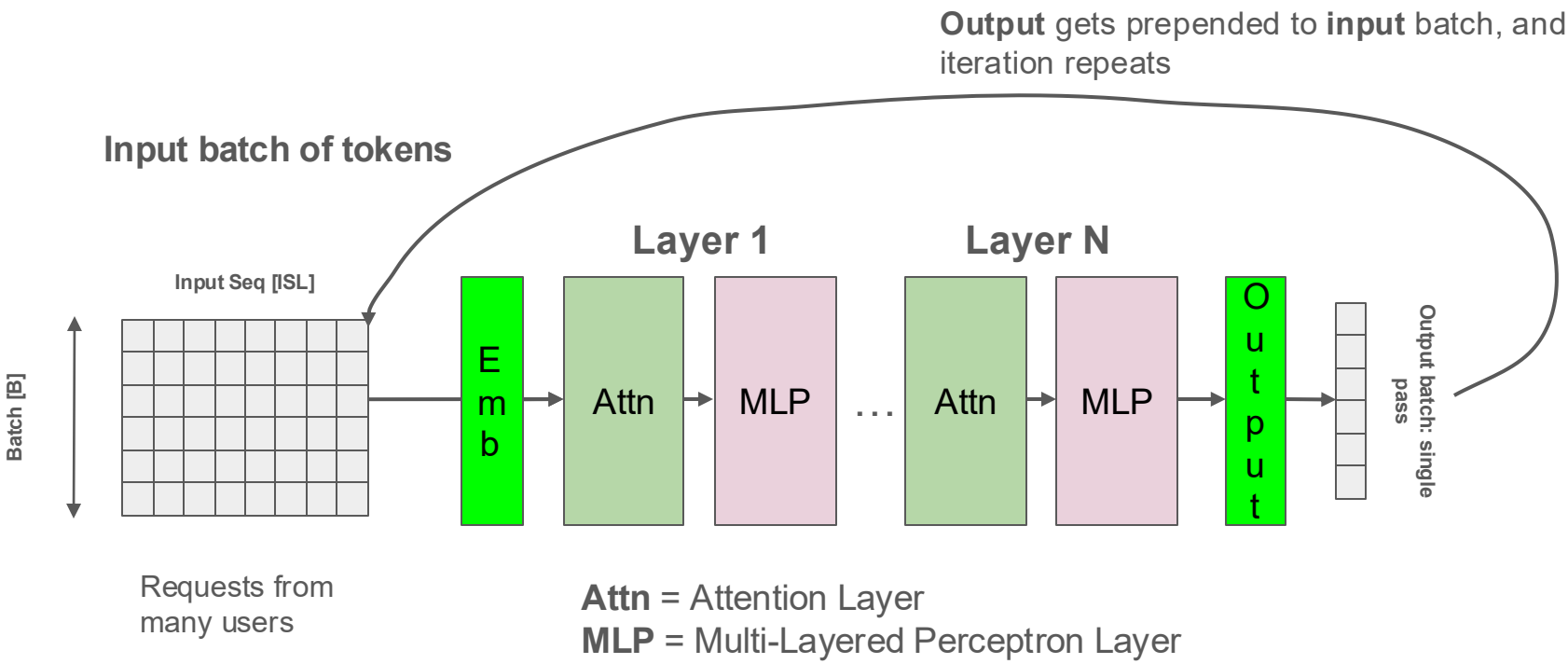
LLM: large language model

[1] Given: a deep (=many layers) **neural network** - think 70B - 671B “weights” - represents lots of matrices

[2] **Inference:** supplying text **input** to the model and obtaining its “continuation” - output

NOTE: the model “infers” new text based on its “stored knowledge” (weights) + “input context” (combine input + some other context if available)

LLM inference overview: a single pass



A bit on terminology

[1] Batch, sequence, tokens ← We just covered

[2] **Activation** - a token converted into **vector** representation for matrix ops

[3] Vector size - “hidden dimension” of the model (e.g., 8192, 7168)

[4] **Embedding/Output** - First and Last layers of model; Convert tokens to activations and activations to tokens (Vocabulary-sized layers). Often re-used.

- **Between** the Layers we pass activation matrices (sized to batch)
- **Inside** the layers we **multiply weights by activations** and apply non-linear transformations

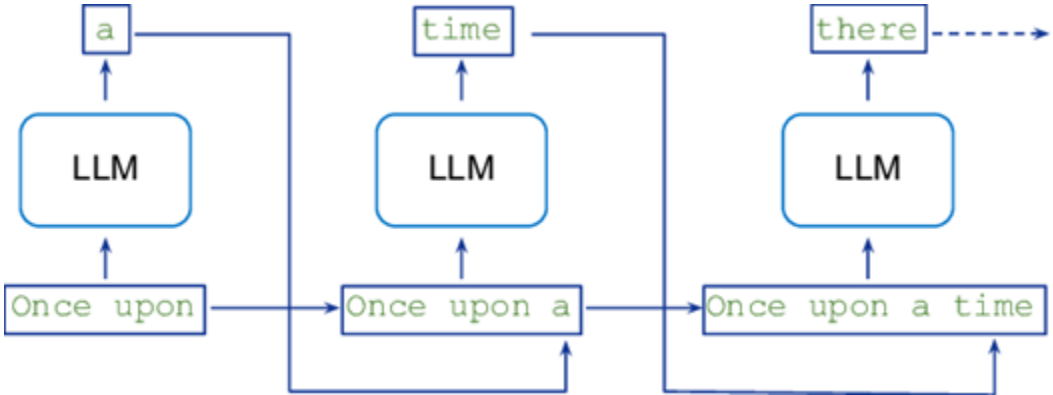
The auto-regression challenges

[1] The “initial” token batch is large - we process **full sequences**

[2] But then we iterate by emitting **one token at time** per sequence

[3] Majority of input remains the same as we loop and prepend!

[4] We are bound by token-by-token “generation” time



From
https://arxiv.org/abs/2305.16367

KV Caching

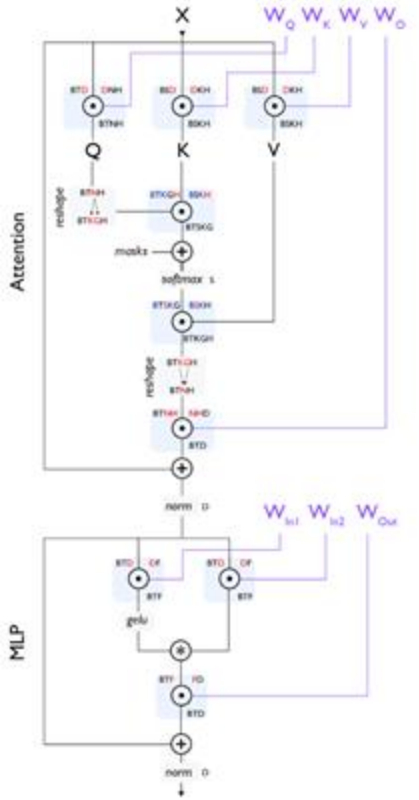
[1] **Attention** layers models how tokens interact in input sequence

[2] When calculating attention, every token “projects” in three vectors: **Key, Query, Value** using the weight matrices found in the Attn layer

[3] K & V vectors **remain the same** during computation, and can be cached [for each inputs sequence → saves a ton of compute time

KV cache is **materialized** when we process the input batch and remains during full pass

How big are the number? LLAMA3-70B example



symbol	dimension
B	batch
L	number of layers
T	sequence length (query)
S	sequence length (key value)
V	vocab
D	d_model, embedding dimension
F	MLP hidden dimension
H	attention head dimension
N	number of query heads
K	number of key/value heads
G	q heads per kv head = N // K

LLAMA3-70B Dimensions

L = 80
T = 1
S = 4K
V = 128256
D = 8192 (=NH)
H = 128
F = 28672 (3.5 * 8192)
N = 64
K = 8
G = 8

LLAMA3-70B Params

Attn = L (2DHN + 2DKH) = L * **151M** (Q + KV matrices)
FFN = L 3DF = L * **704M**
Emb = 2VD = **2.1G**

KV/token = L * 2KH = **160K elements**

ATTN: **151M elements per layer**

MLP: **704M elements per layer**

Emb: 2.1G (one layer)

KV cache: **160K per token!**

From: https://jax-ml.github.io/scaling-book/

Application and ISL/OSL

ISL = input sequence length; OSL = output sequence length

App	ISL	OSL
Chat	Small (~128)	Small (128)
Deep Research	Small (~128-1K)	Large (~128K+)
Coding	Large (128K-1M)	Small-Med (128-4K)
Summarization	Large (128K-1M)	Small (128-1K)

With long sequences: KV cache is very large - consumes most of memory

A detour on memory demands

Using LLAMA3 70B as a simple example:

[1] 70G weights (if using FP8)

[2] 160K per token for KV cache (using FP8)

[3] B200 HBM3e memory is ~192GB

–

Single GPU can fit 750K tokens + model weight, which seems good, huh?

But you also need **more GPUs to scale decoding speed** + fit larger models (405B)

Distributed inference & scale-up network

Splitting the model among GPUs

- Size: Larger models won't fit on single GPU: the size problem

Parallelism:

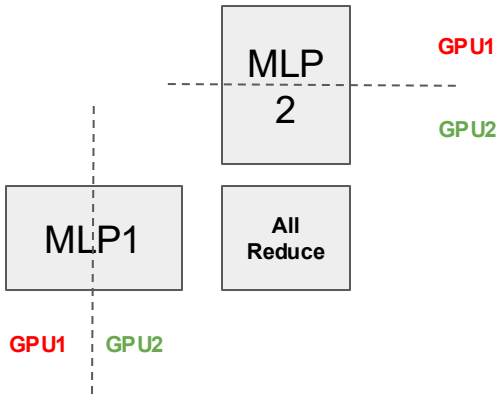
- Splitting model among more GPUs enabled more **compute**
- Splitting model weights enables **higher memory bandwidth** per weight

Parallelism types:

- **Data Parallel:** split the batch (sequences) among model **replicas**
- **Model Parallel:** distributed model **weights** among GPUs

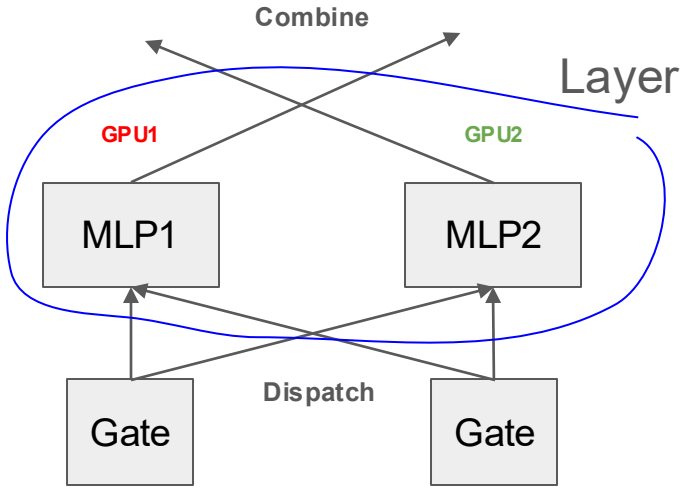
Model parallelism(s): enable more compute and mem b/w

Tensor Parallel (TP)



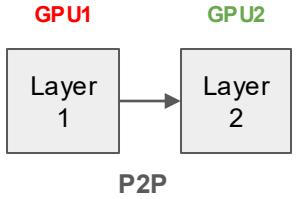
- Split **weights** of matrices
- **Per-GPU** matrix multiplication
- All-reduce to gather partial results across GPUs

Expert Parallel (EP)



- Split MLP weights too
- Dispatch “tokens” (activations) to experts
- Multiply activations by weights locally
- Combine afterwards

Pipeline (PP)



- Split layers of LLM among GPUs
- Pass activations along (whole batch)
- Improves throughput, but not latency

Distributed inference & network

It’s desirable to map EP/TP parallelisms onto **scale-up network**

- In **model parallelism** we exchange activations
- Latency of exchange is critical for token generation rate

```
Latency ~= Propagation Delay + Batch / Bandwidth
```

Scale-up networks: a bit of history

Precursors

- **CPU to CPU** interconnects: UPI, QPI, HyperTransport …

^^ Cache coherent and hence highly non-trivial to scale

- **GPU to GPU:** NVLink (first in Pascal generation)

^^ Bandwidth-oriented, no global cache coherency, local coherency only

Why not just use PCIe (NTB or CxL to scale)?

NVLink evolution

- Pascal was P2P links only
- Volta added first “NV switch” (still supported P2P)
- NV switch - routes memory transaction packets

NVlink directly extends internal GPU memory subsystem with memory addr translations

- Load/Store operations (Mem2Reg, LDG/STG) ← **sync**
- TMA (Tensor Memory Accelerator) ← **async**
- Copy Engines (think bulk data movements) ← **async**

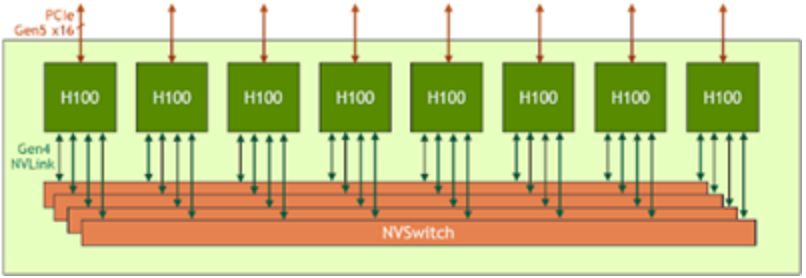
Special “stuff:” Multicast operations and in-network reductions (Hopper and onwards)

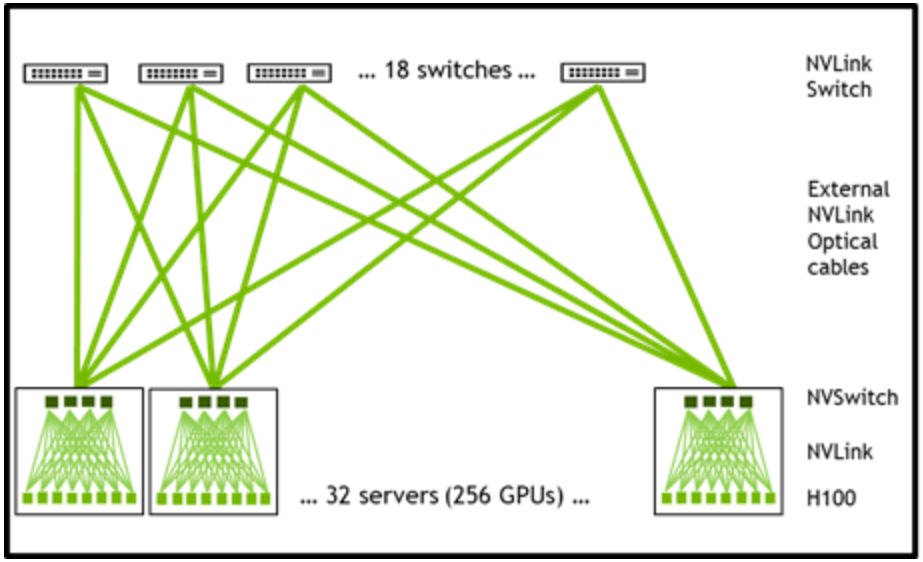
Scale-up “fabric” dimensions

	Ampere	**Hopper**	**Blackwell**
Lane speed	50Gbps	100Gbps	200Gbps
GPU NVL bandwidth	300GB/s	450GB/s	900GB/s
Fabric scale	8 (board)	8 (board)	72 (rack, 18 boards)

bandwidth is in unidirectional units

Scaling-out the scale-up?





NVLink switches can stack up if needed

Contrast: scale-up vs scale-out?

Scale-out = RDMA (RoCE/IB)

[1] NVLink ~9x more bandwidth per GPU (e.g., 900GB/s vs 100GB/s)

[2] Transport “in silicon” - die/package area is of paramount concern

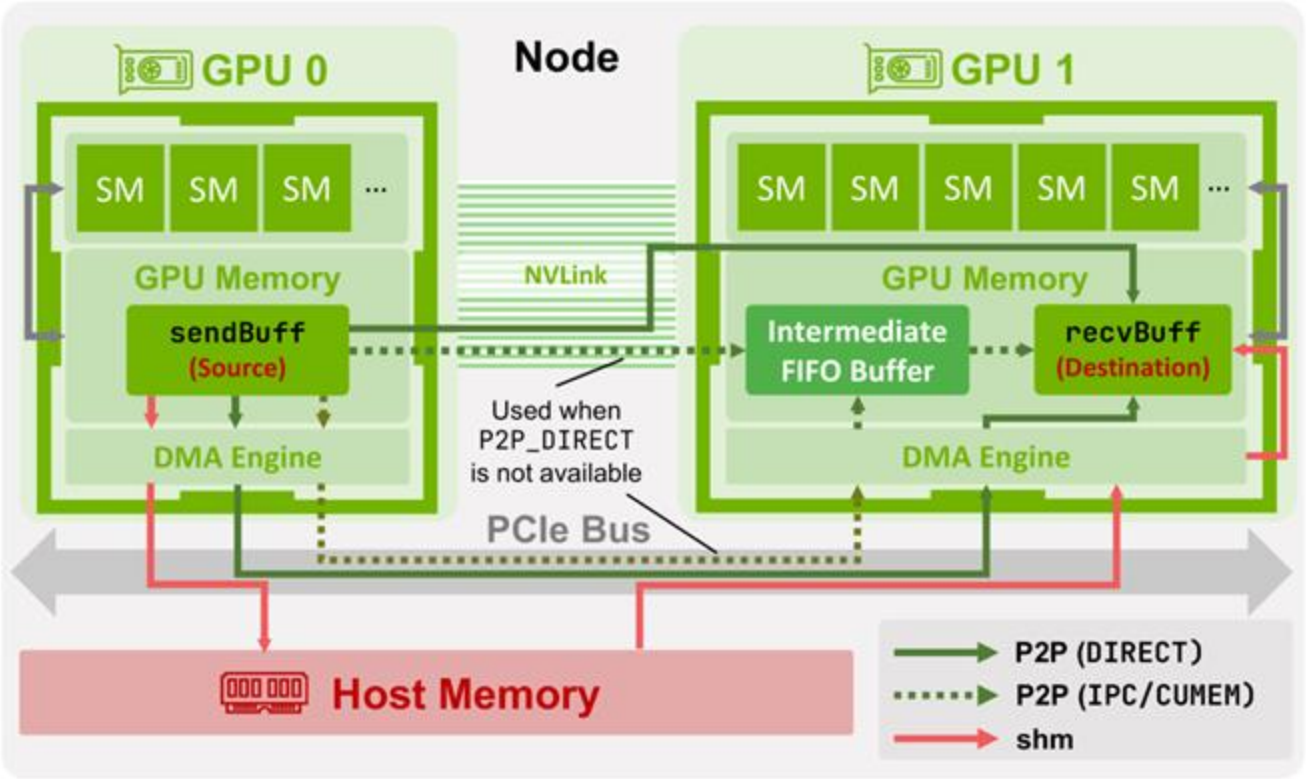
[3] Direct memory access via pointer-based operations - easier to program (?)

[4] Power efficiency is essential, hence “dense scale” and copper

[5] Collective offload (reductions, multicast) ← c.f. InfiniBand

Primary difference: bandwidth, latency of direct operations, power efficiency

NCCL over NVLink

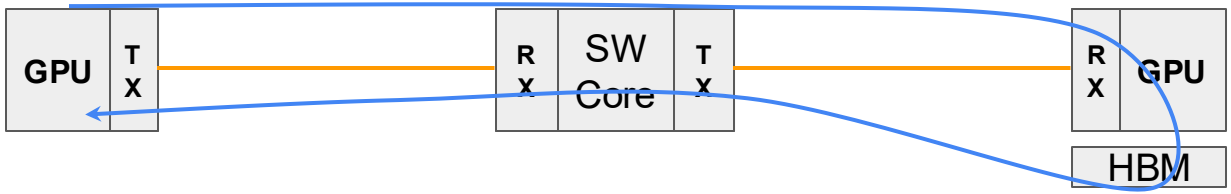


- Low-latency comms are driven by SM (streaming multiprocessors)
- CUDA kernels launched by NCCL library push and poll data between GPUs
- Producer/consumer model with synchronization

From: https://arxiv.org/html/2507.04786v1

The hard part: latency

- Static: propagation delays, data-link layer
- Dynamic: queueing and FEC effects
- Error propagation: non-trivial in memory-semantic protocols
- Comm-type: reads require round-trip; write & poll is faster



- 2x GPU die crossing (from SM to remote L2 cache/HBM)
- 2x TX/RX FEC processing
- 2x wire propagation
- 1x switch core crossing and queueing

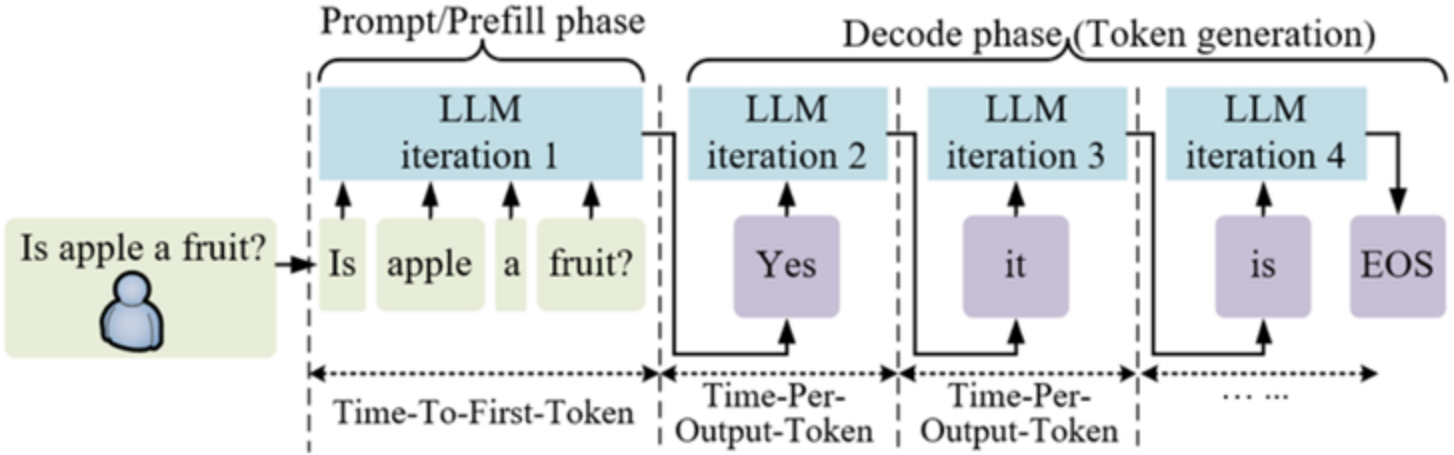
What’s besides NVLink?

- **UALink** standard (no implementations yet?) → Ethernet SerDes
- Broadcom **SUE** → Ethernet based (WIP?)
- Both documented openly online

Tricky part: you need compute engine to implement the transport

So why is scale-up a big deal for inference?

Performance metrics for inference



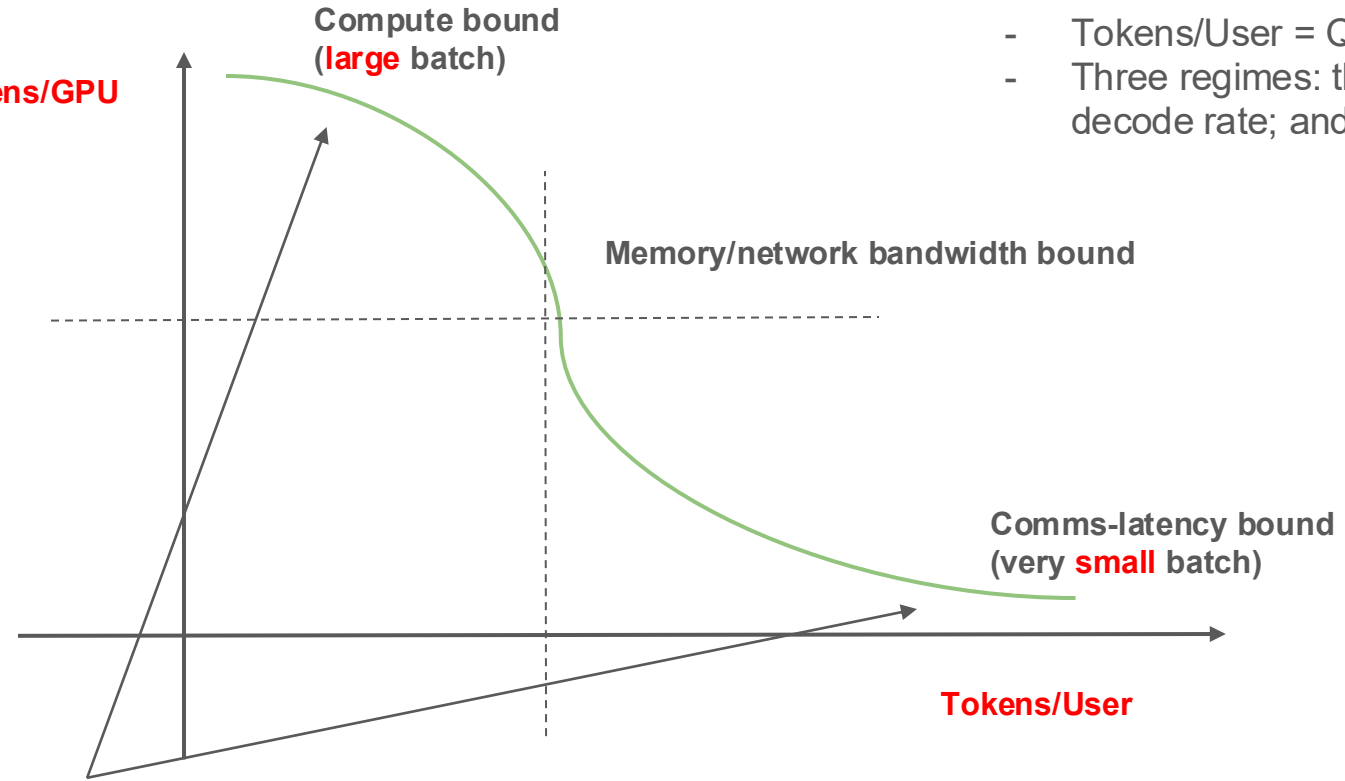
TTFT and TPOT (ITL) are critical inference metrics

Total query completion time = TTFT + OSL * TPOT

From: https://huggingface.co/blog/Kseniase/inference

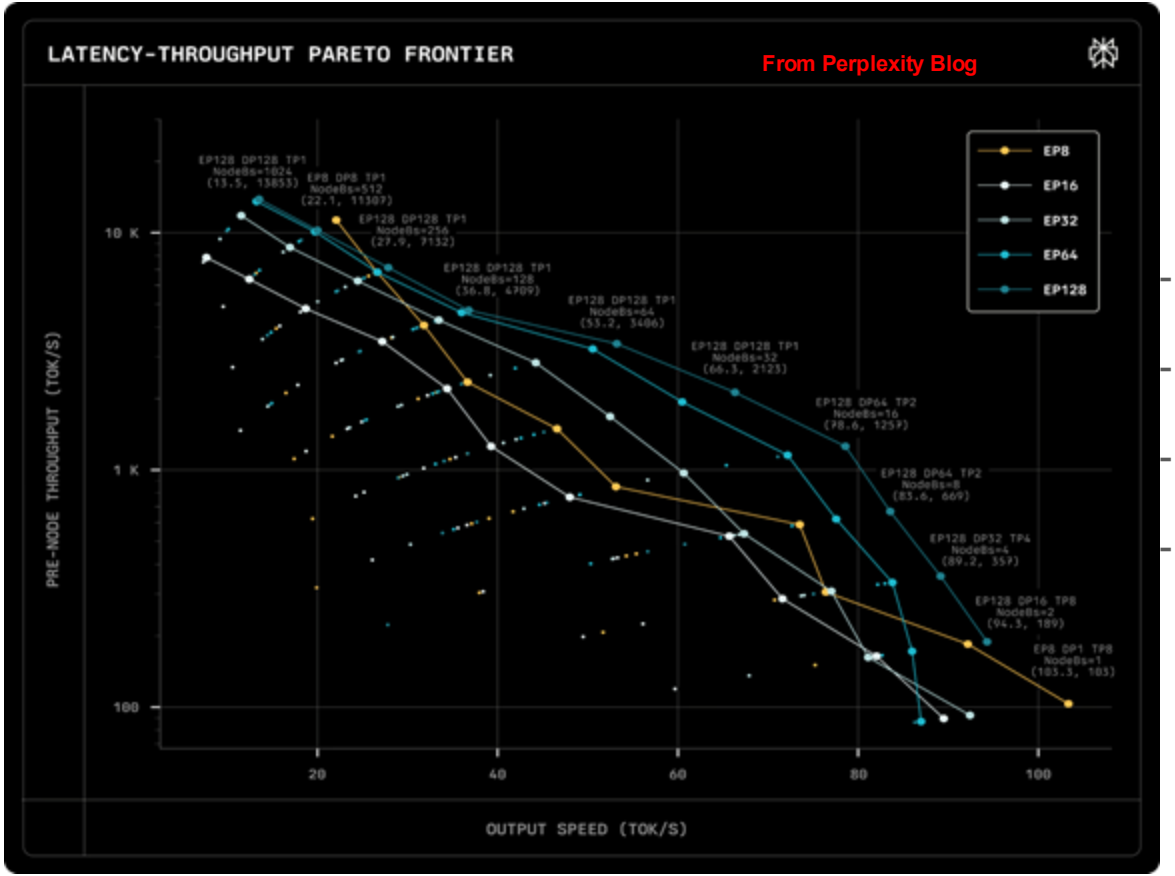
Latency/throughput tradeoff Pareto

- Tokens/GPU = throughput = $
- Tokens/User = Quality
- Three regimes: throughput vs decode rate; and the middle



Model parallelism types and batch size vary

Latency-throughput tradeoff



- This is for Hopper-based system - up to 16 H200 machines
- EP/TP parallelism traffic maps on the scale-up or scale-out fabric
- Highest decode rate when we map on scale-up (EP8/TP8) only
- Reason: bandwidth and latency

https://www.perplexity.ai/hub/blog/lower-latency-and-higher-throughput-with-multi-node-deepseek-deployment

So how scale-up helps… how?

- **Low-latency:** needed for all-reduce in the max(tokens/user) regime - for dense matrix multiplications
- **Larger radix/scale:** helps with sharding and boosts normalized memory bandwidth (=more shards)
- **High bandwidth:** needed for latency on mid-to-large batches

NOTE: more “sharding” means more exposed comms latency (less compute to hide)

Disaggregated inference

Prefill & Decode aka “Context” vs “Generation”

Recall inference has two phases

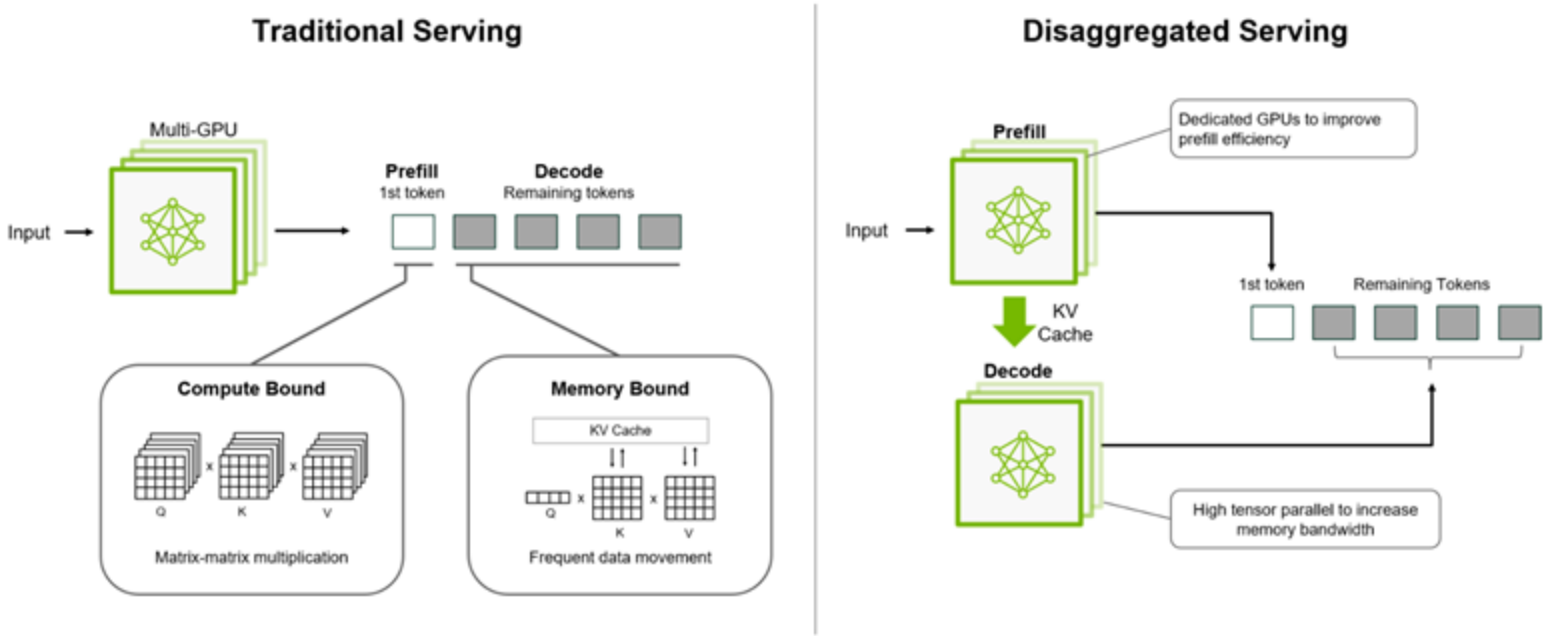
- Prefill aka Context: process the initial batch (attention)
- Decode/Generation: generate one token a time per sequence

Prefill is almost entirely **compute bound**

Decode requires higher memory bandwidth and low comm latency

It is not uncommon to see two phases “**disaggregated**”

Prefill/Decode (P/D) split



https://developer.nvidia.com/blog/introducing-nvidia-dynamo-a-low-latency-distributed-inference-framework-for-scaling-reasoning-ai-models/

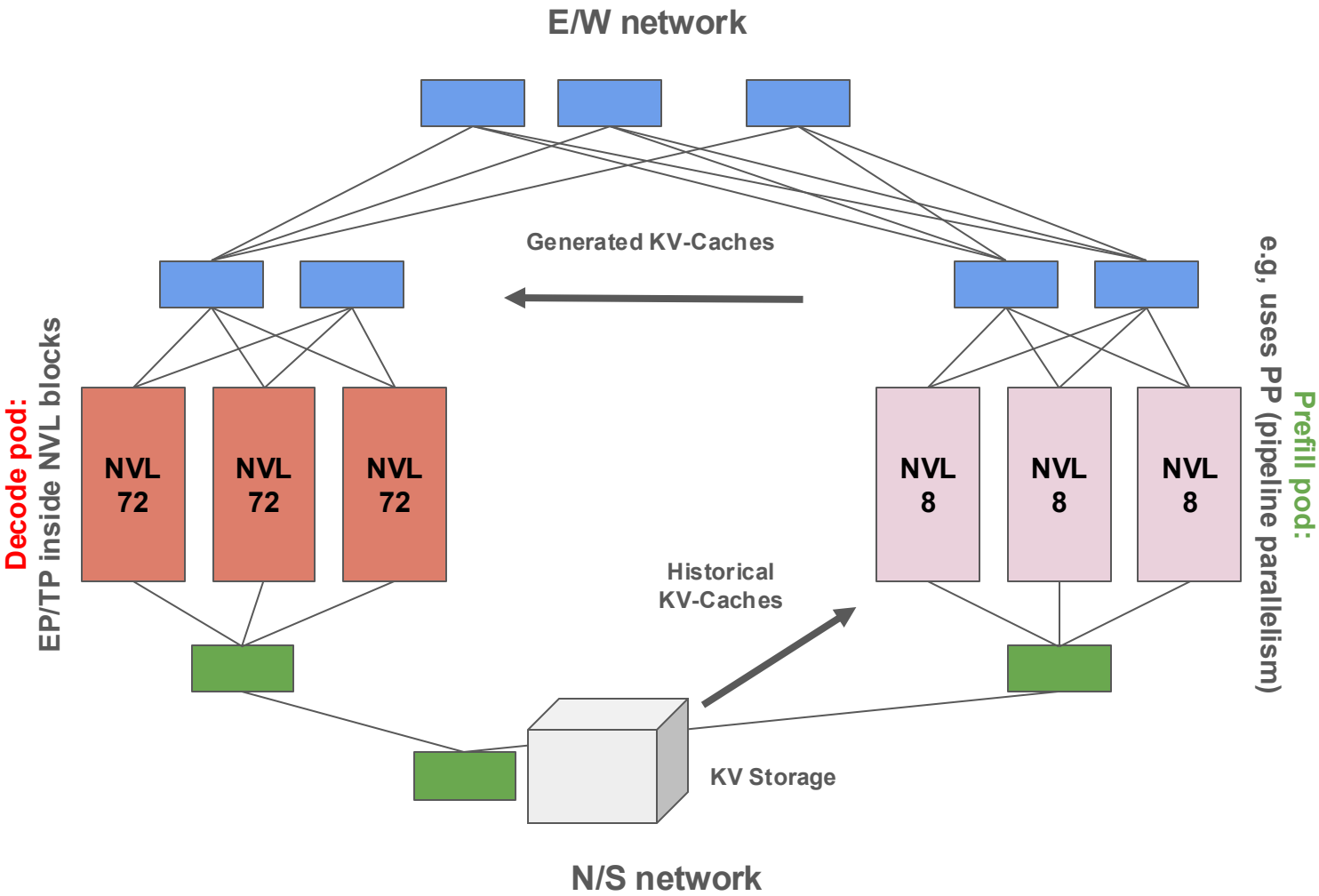
How does P/D disagg works again?

- Same model, but hosted twice
- Different distributed partitioning strategies
- Global scheduler (router) directs queries
- Prefill produces **KV cache**
- Decode takes the KV cache and emits tokens

What’s KV Cache storage?

- There are two kinds of “KV cache”
- First is the “active” that is used in running computations
- Second is “inactive” which encodes your old sessions
- Alternatively: can you your “indexed” code base

KV cache “offline” storage is critical for performance on very large contexts (e.g. 100K+ tokens of history or shared context)


E/W network
Generated KV-Caches
Decode pod:
EP/TP inside NVL blocks
NVL 72
NVL 72
NVL 72
NVL 8
NVL 8
NVL 8
Prefill pod:
e.g, uses PP (pipeline parallelism)
Historical KV-Caches
KV Storage
N/S network

Recap

- LLM inference is distributed matrix multiplication (TP/EP/PP)…
- KV cache creates memory capacity pressure
- Throughput vs. latency tradeoffs are non-trivial
- Distributed inference runs over scale-up - whether possible
- Disaggregated inference specializes “pods” for prefill/decode
- E/W network transfers KV caches
- Inactive KV cache storage becoming essential for large contexts