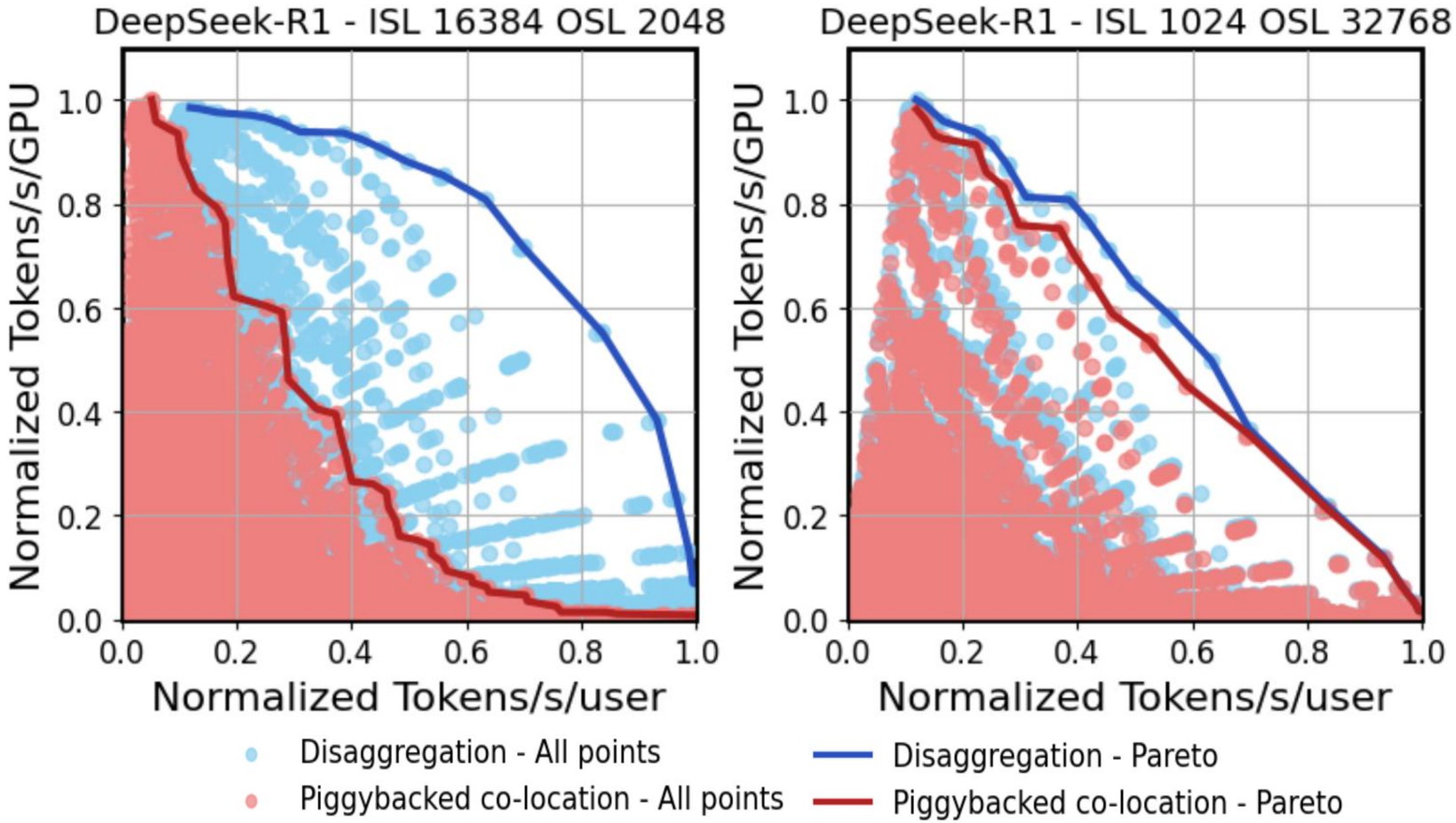
A Pragmatic Take on Inference Disaggregation

MLSys 2026 · NVIDIA

Tiyasa Mitra · Ritika Borkar · Nidhi Bhatia · Shivam Raj · Hongkuan Zhou · Yan Ru Pei · Vishwanath Venkatesan · Kyle Kranen · Ramon Matas · Dheevatsa Mudigere · Ritchie Zhao · Maximilian Golub · Arpan Dutta · Suresh Nambi · Sailaja Madduri · Dharmesh Jani · Brian Pharris · Itay Neeman · Bita Darvish Rouhani

Inference Serving Performance: The Throughput–Interactivity Pareto Frontier

Benefit of disaggregation varies heavily with serving scenario – calls for a systematic design space exploration



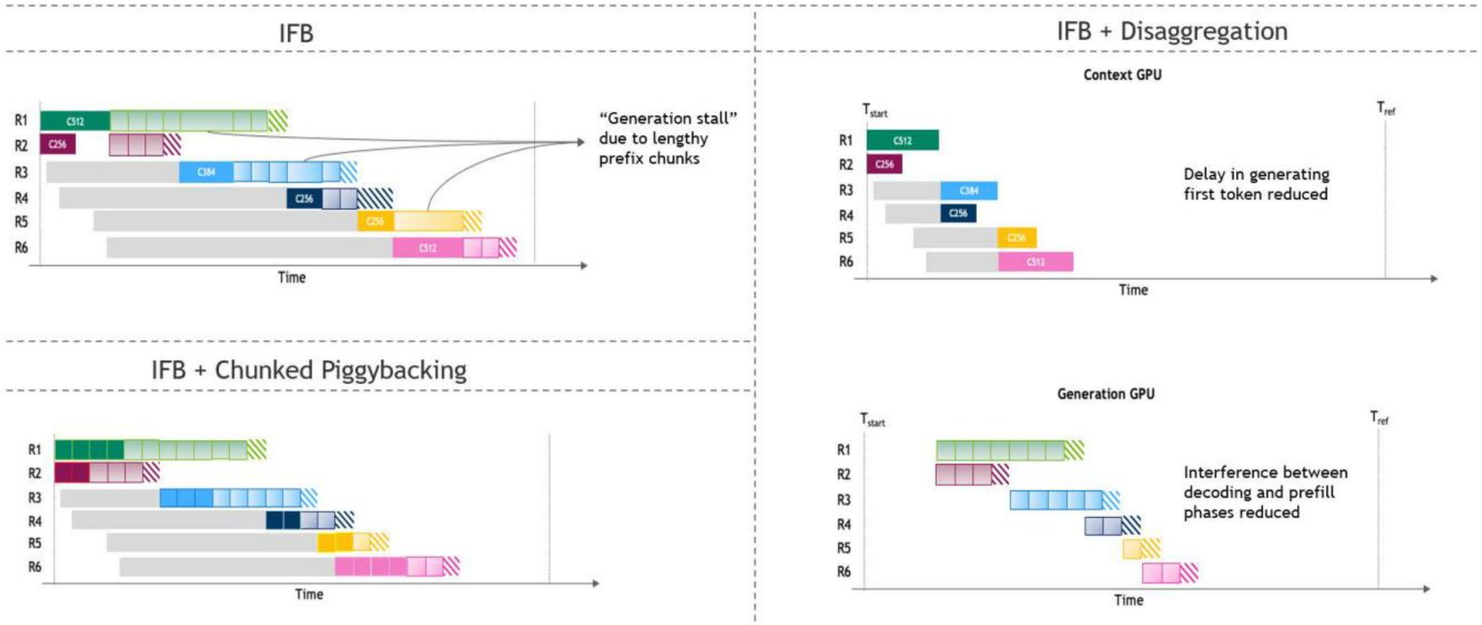
Pareto frontier for DeepSeek-R1. Left: prefill-heavy (ISL>>OSL) — disaggregation wins. Right: decode-heavy — co-located with piggybacking is competitive.

LLM Inference Serving Fundamentals and Datacenter SLAs

Prefill and Decode Phases have distinct compute patterns and SLA targets

- **Colocation with in-flight batching**
  - Prefill and decode phases served on the same model instance
  - Introduce a new sample into batch as soon as another request is retired
  - Generation Stalls!
- **Piggybacking**
  - Chunk prefill into smaller blocks to avoid generation stalls
  - Prefill and decode still served on the same instance → suboptimal mapping
- **The promise of disaggregation**
  - Each pool independently optimizes parallelism, batching, and hardware mapping

Phase	Characteristics	SLA constraint	Typical Value
Prefill	Parallel compute	**First Token Latency (FTL)**	100ms – 10s
Decode	Autoregressive	**Token-to-Token Latency (TTL)**	1ms - 50ms



Disaggregated serving decouples prefill and decode phases, enabling independent optimization towards FTL and TTL constraints

Design Space Exploration

Millions of design points simulated using a high-fidelity, kernel-aware GPU performance simulator
Rate matching between prefill and decode pools for full Pareto comparison against co-located serving

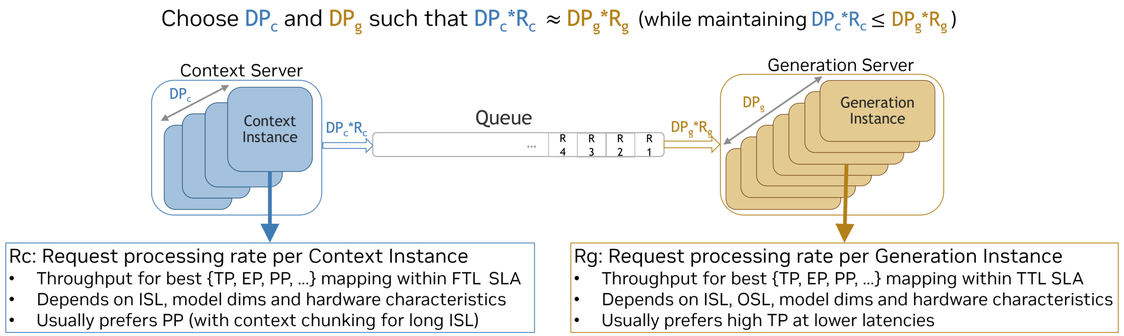
Experimental Setup

- **Models:** DeepSeek-R1 (671B, MoE), Llama 3.1 (8B/70B/405B, dense)
- **Traffic:** ISLs: IK-16K, OSLs: 2K-32K
- **Hardware:** Blackwell Systems, NVLink Domains 8, 72
- **Precision:** NVFP4/MXFP4
- **Partitioning Strategies:**
  - TP — Tensor Parallelism: splits weight matrices across GPUs
  - EP — Expert Parallelism: shards MoE experts across GPUs (MoE-specific)
  - PP — Pipeline Parallelism: stages layers across GPUs
  - CPP — Chunked Pipeline Parallelism: prefll specific
  - TEP — Tensor Parallel attention + EP FFN (MoE-specific hybrid)

Simulator Details

- Device-level model: memory hierarchy, compute core implementation, interconnect latencies, optimized kernel
- System-level model: NVLink + Ethernet, optimized communication collectives
- Validated against silicon across multiple GPU generations

Rate Matching Prefill and Decode Pools



Algorithm (2 steps)

1. Prefill config selection: highest throughput/GPU to meet FTL SLA
2. For each decode config, integer-solve for α = #prefill / #decode instances
   - Prefill req rate = R_C, Decode req rate = R_G
   - decode_instances:prefill_instances, $\alpha = R_C/R_G$

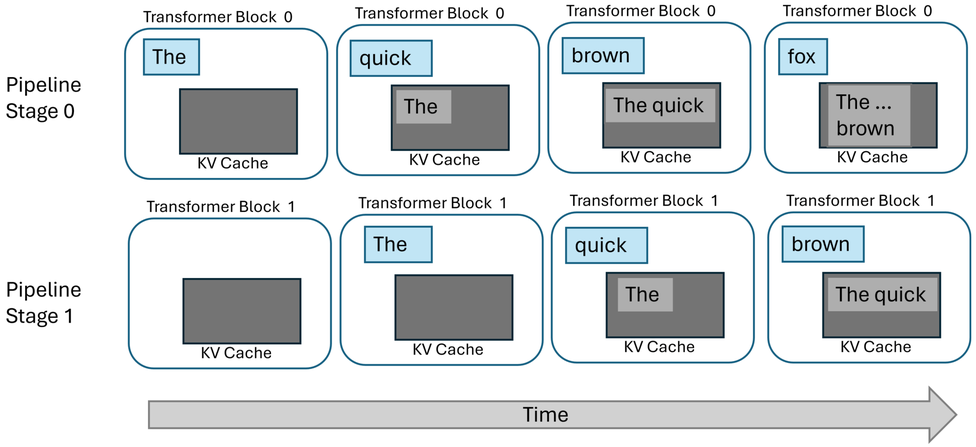
GPU budget minimization: choose smallest α satisfying both SLAs

KV cache + weight capacity constraints accounted for

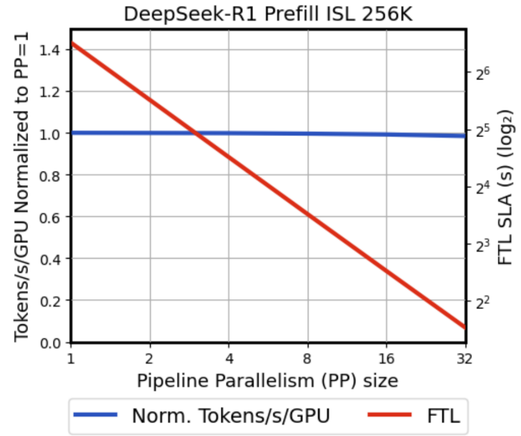
Optimizing Prefill Performance: Chunked Pipelining for Tight FTL SLAs

CPP dramatically reduces communication overhead vs. Tensor Parallelism at long context lengths

- **The FTL challenge**
  - Context lengths reach 100K–1M tokens: FTL SLA is still a few seconds
  - Model Sharding comes with communication overhead:
    - Tensor Parallelism introduces expensive all-reduce
    - Expert Parallelism, when available, introduces all-to-all cost and imbalance
- **How CPP works**
  - Split input into chunks, process each independently using prior KV cache
  - Pipeline: overlap earlier layers of new chunk with later layers of previous chunk
  - No output dependency on previous chunk — only KV cache dependency
- **Communication advantage**
  - TP: 2 × ISL × d_model × N_layers × bytes (all-reduce per layer)
  - CPP: ISL × d_model × (N_pp − 1) × bytes (send-recv only)
  - N_pp << N_layers and send-recv << all-reduce → CPP wins at long ISL



CPP — chunks enter pipeline early, overlapping layers.

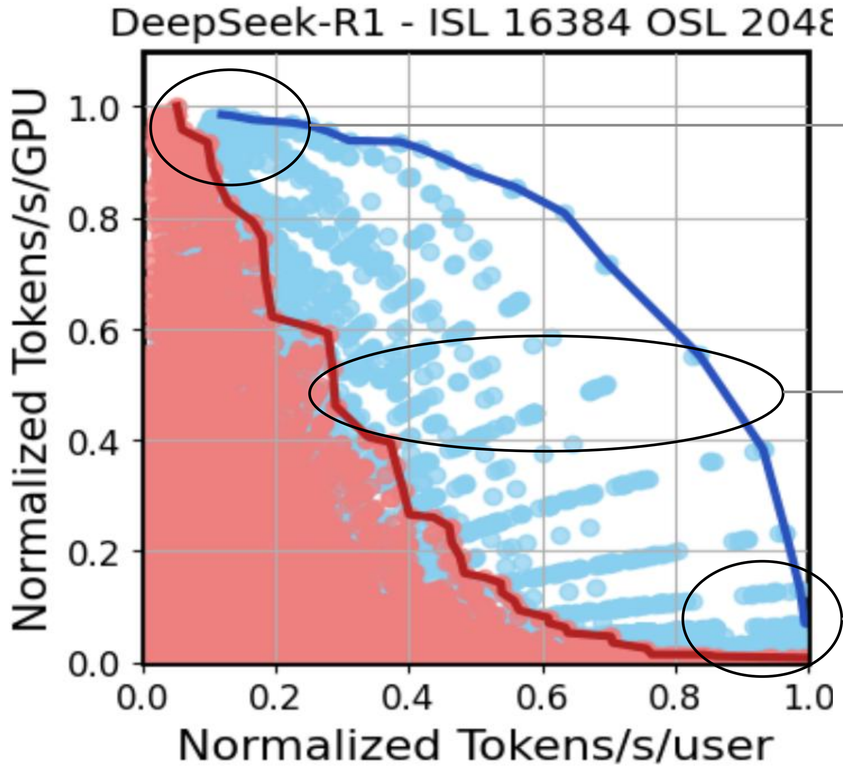


Increasing PP stages reduces FTL while keeping throughput high.

Optimizing Decode Performance: Bottlenecks Change With TTL SLAs

Optimizing for a single design point can lead to sub-par cross-Pareto performance

TTL SLAs govern where the system should operate on the decode (generation) pool
Moving right along the Pareto frontier → tighter TTL SLAs(higher tokens/s/user)
As TTL tightens, smaller batch sizes and greater tensor parallelism are preferred



High-throughput Region: **Colocation is fine!**
large batches → good compute utilization on GEMMs
Prefers fewer GPU mappings while ensuring large batches

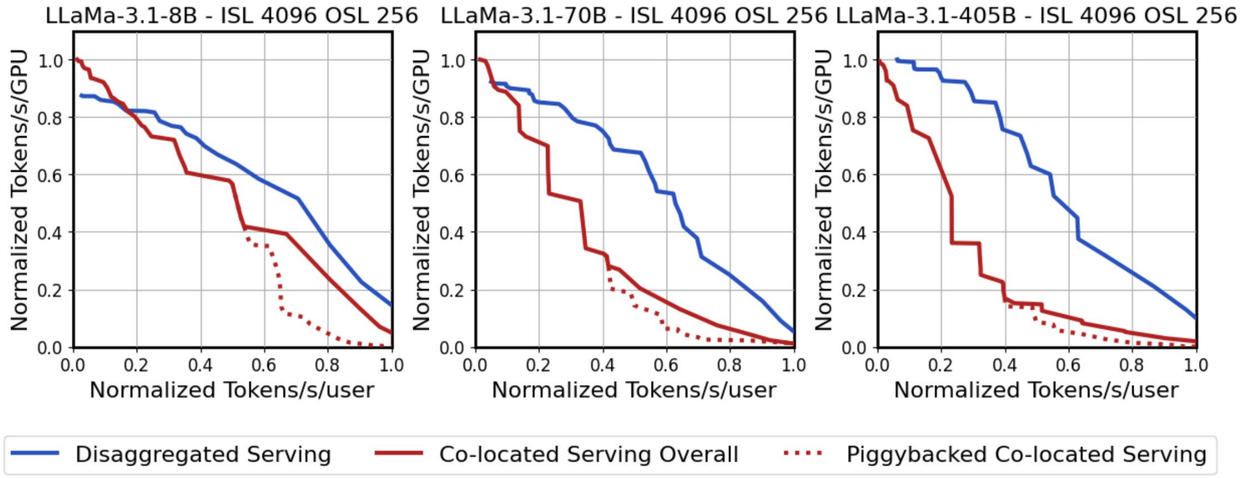
Mid-Latency Region: **Disaggregation is great!**
smaller batches → memory and communication bandwidth bound
Larger GPU footprint to improve latency →TP and wide EP
TP increases as we move right on the Pareto

Low-Latency Region: **Disaggregate until you hit batch 1 ☺**
Very small batches → latency bound
Within chip: kernel launch latency, datapath and memory latencies
At system level: communication can be latency bound

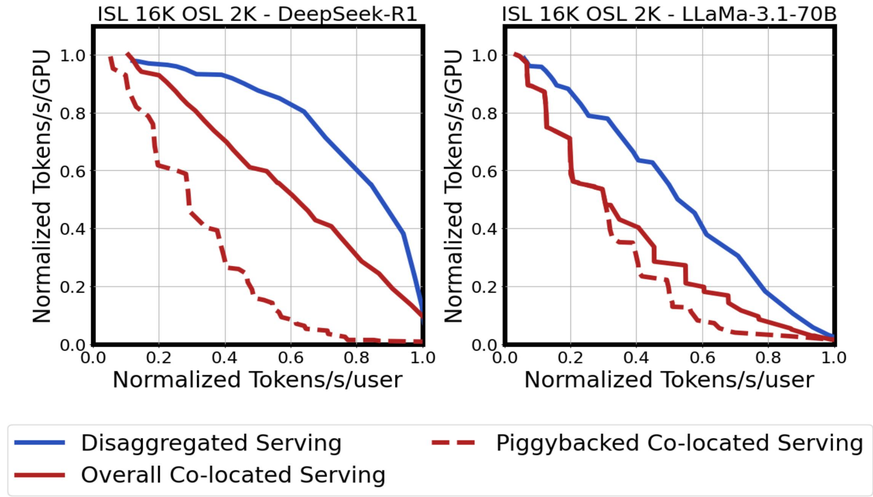
Model Architecture and Size Sensitivity

MoE models gain more from disaggregation due to Expert Parallelism
Larger models gain more from disaggregation, due to more GPUs in the performant mapping search space

- **Model Arch: DeepSeek-R1 (MoE) vs. Llama-3.1-70B (dense)**
  - MoE has an extra Expert Parallelism (EP) dimension to exploit
    → More benefit from disaggregation in the medium latency region
- **Model Size: Llama-3.1 Model Family**
  - Smaller models prefer fewer GPU mappings to ensure good utilization
    → Leads to smaller mapping search space for disaggregation



Disaggregated vs. co-located for Llama 8B / 70B / 405B. The performance gap widens with model scale.



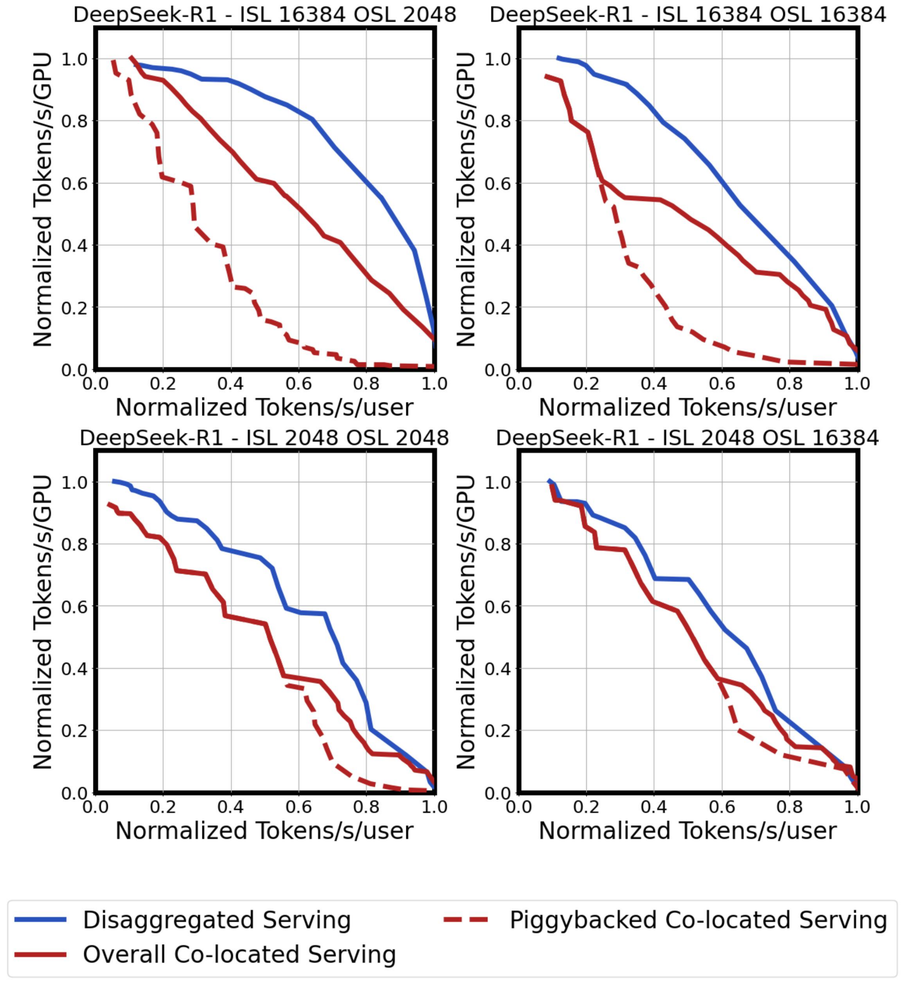
Disaggregated vs. co-located for DeepSeek-R1 and Llama-70B.

Frontier models are trending sparser and larger

→ the promise of disaggregation is growing!

Traffic Pattern Sensitivity: When is disaggregation worth it?

There must be enough prefill traffic to reap the benefits of independent prefill & decode serving strategies

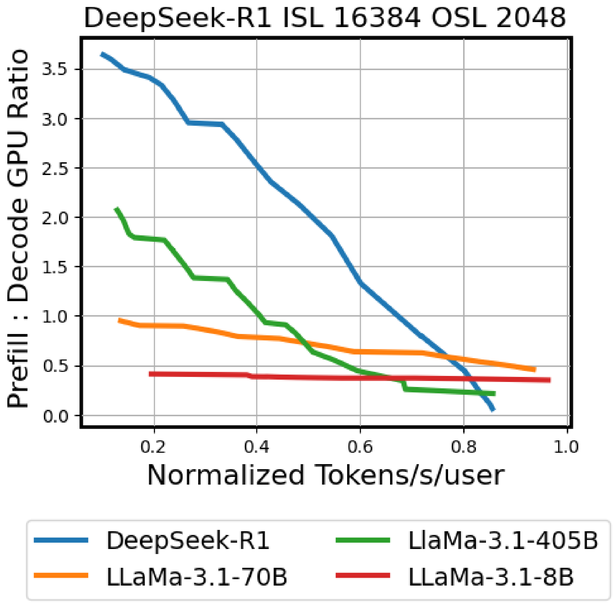


- **Prefill-heavy (ISL >> OSL):** Disaggregation wins decisively
  - Prefill dominates compute; decode stalls with co-location
  - Prefill and decode prefer very different mapping strategies
- **Decode-heavy (ISL << OSL):** Co-located serving is competitive
  - Decode work dominates, prefill interference does not significantly slow down decoding
  - Suboptimal prefill mapping does not hurt end-to-end performance
- **Balanced Traffic (ISL ~ OSL):** Still benefits from disaggregation!
  - 1:1 ISL:OSL ratio still spends more time in decoding due to the autoregressive nature of token generation
  - Note: ISL16K-OSL16K is more “prefill heavy” than ISL 2K-OSL 2K
    - → Prefill attention grows quadratically with sequence length; decode attention grows linearly

Pareto for DeepSeek-R1 across 4 traffic patterns. ISL/OSL={16K/2K, 16K/16K, 2K/2K, 2K/16K}.

Dynamic Rate Matching: Fixed Ratios Leave Goodput on the Table

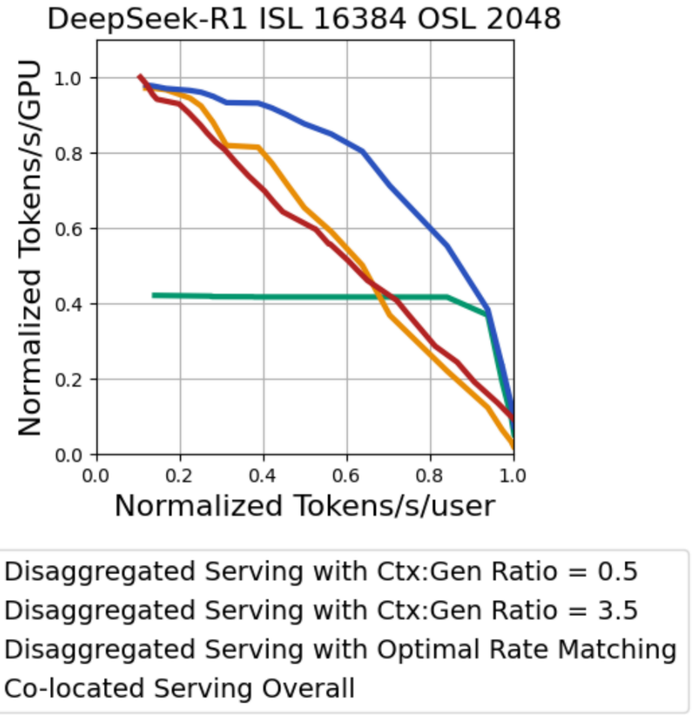
The optimal Ctx:Gen GPU ratio varies substantially with model, traffic, and latency target



Optimal Ctx:Gen ratios vary widely with workload and SLA targets.

Many serving conditions and choices can make the ratio shift:

Tight TTL: more decode GPUs → ratio decreases
Larger ISL: more prefill work → ratio increases
Prefix caching: fewer prefill GPUs → ratio decreases
Speculative decoding: fewer decode GPUs → ratio increases



Fixed Ctx:Gen ratio (0.5 or 3.5) degrades vs. dynamic optimal at different latency targets.

Consequence of wrong static ratio:

Ratio 3.5: great at relaxed latency, terrible at tight TTL
Ratio 0.5: great at tight latency, wastes GPUs otherwise

Shifting workload patterns in the real world: Dynamo SLA-Aware Planner

Continuous adaptation of Ctx:Gen allocation — 2× better goodput vs. best static; 8× vs. inefficient ratio

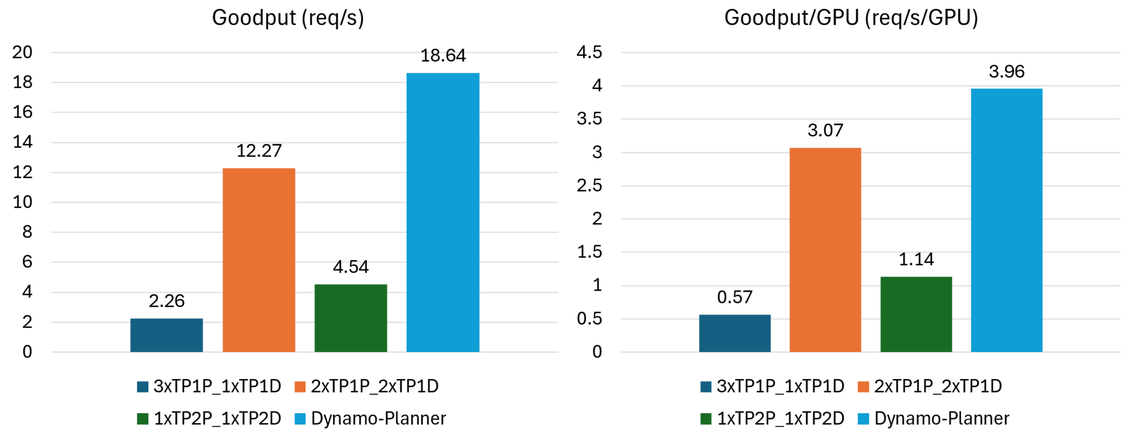
- **How it works (3 phases)**
  - 1. Sweep: evaluate TP/PP/EP/TEP configs to find best engine mapping
  - 2. Profile: measure FTL vs. ISL and TTL vs. in-flight KV usage per config
  - 3. Runtime: monitor ISL/OSL/QPS; time-series forecast to scale ahead
  - Moving-average correction: KV prefix reuse, prefill queueing, ISL/OSL variance

- **Benchmark (DeepSeek-R1-Distill-Llama-8B, H200 GPUs)**
  ISL=3K · OSL=300 · QPS=5–45 req/s · FTL ≤ 200ms · TTL ≤ 10ms

8× better goodput vs. inefficient ratio (3:1 Ctx:Gen, TP1)

4× better goodput vs. wrong mapping (1:1 Ctx:Gen, TP2)

1.5× better goodput vs. best static (1:1 Ctx:Gen, TP1)



Goodput and goodput/GPU across serving configs. Dynamo SLA-Aware Planner dominates.

NVLink Domain Sensitivity

Larger NVLink domains consistently improve disaggregated serving performance

Why larger NVLink domains help

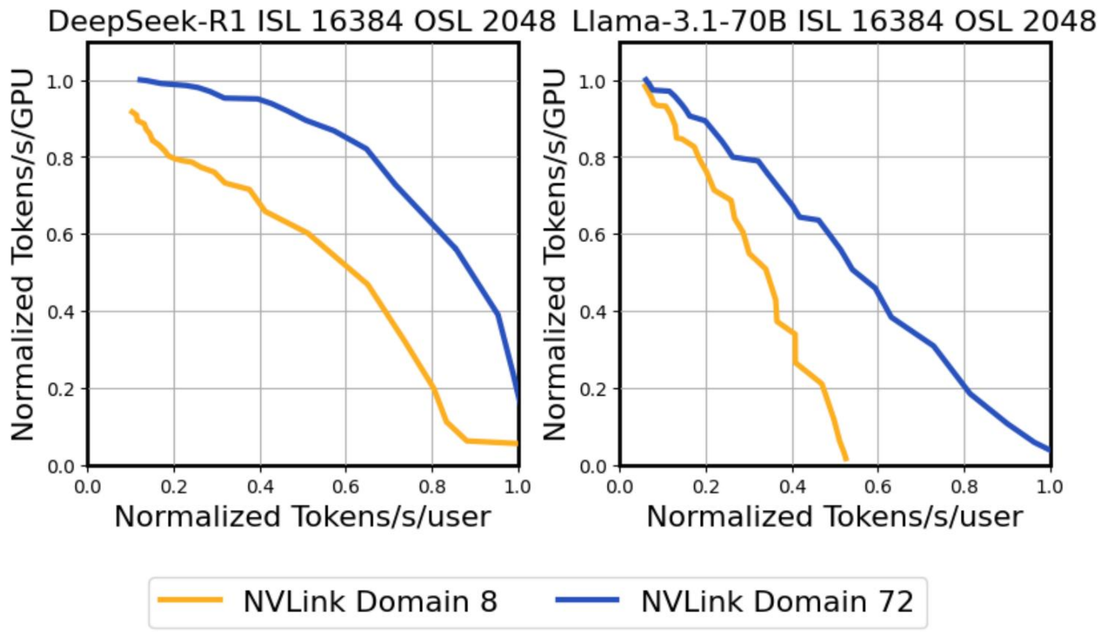
NVLink provides high-bandwidth, low-latency communication within prefill and decode instances

Larger NVLink domains = wider parallelism options per pool

- **DeepSeek-R1 (MoE)**
  - Larger domain enables higher Expert Parallelism (EP) within the decode pool
  - Faster All-to-All enables workload distribution across more GPUs
- **Llama-3.1-70B (dense)**
  - Larger domain enables higher Tensor Parallelism (TP) within the decode pool
  - Efficient All-Reduce enables lower per-token latency at tight TTL SLAs

NVLink domain grows with each GPU generation

→ improved performance opportunity for disaggregated serving!

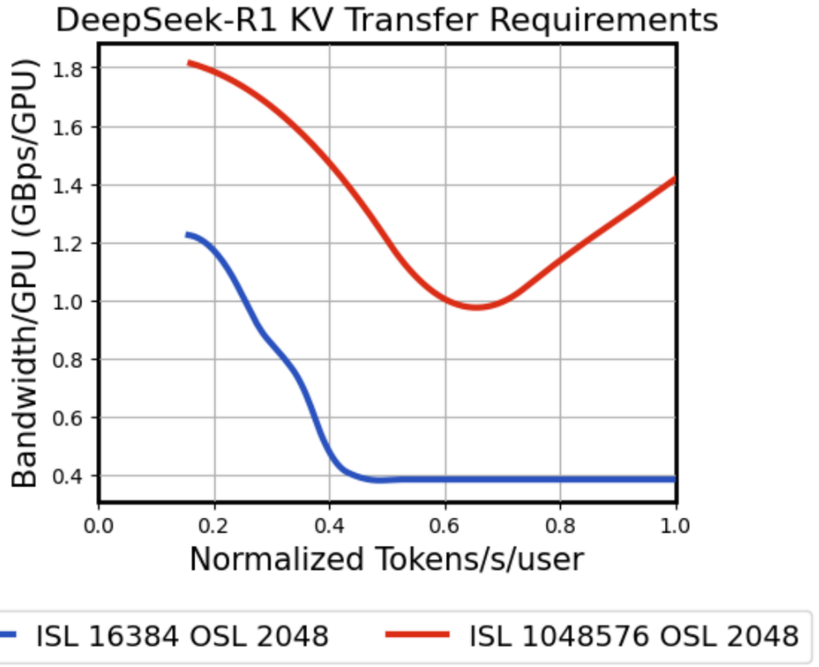


Pareto for two NVLink domain sizes. Larger domain (right) shifts the frontier outward for both models.

Deployment: KV Cache Transfer

Bandwidth requirements suggest that transfer over Datacenter Ethernet/IB is viable
NIXL provides maximum flexibility for asynchronous KV transfer

- **Bandwidth requirements**
  - KV is transferred in layer-grouped bursts to saturate interconnect and hide latency
  - BW_ctx = (N_L·B_ctx·ISL·d_h·N_kv·bytes·S_util) / (FTL·NumGPU_ctx)
  - BW_gen = (N_L·B_gen·ISL·d_h·N_kv·bytes·S_util) / (TTL·OSL·NumGPU_gen)
  - **Existing datacenter BW is sufficient; egress decreases as ISL grows**
- **What stresses bandwidth?**
  - **Shorter FTLs:** FTL grows superlinearly with ISL; KV size grows linearly
    → more time to transfer
  - **Shorter OSLs:** Shorter OSLs mean more frequent KV cache ingestion on decode GPUs
  - **Relaxed TTL SLAs:** Larger batch and fewer GPU mappings at high throughput
    → more BW per GPU
  - **Larger KV cache size:** MLA models require less BW than GQA models due to smaller KV cache footprint



Max(egress, ingress) BW vs. TTL. Existing datacenter BW is sufficient.

NVIDIA Inference Transfer Library (NIXL)

- Non-blocking, async P2P KV transfer - inference engine uninterrupted during transfer
- Allows for dynamic allocation and deallocation of workers. New workers join by exchanging metadata only, avoiding global sync
- Auto-selects optimal path across RDMA (InfiniBand/Ethernet), NVLink, or GPUDirect Storage
- Integrated with Dynamo and widely adopted across open-source inference frameworks

Deployment: KV Cache Layout and Cache Aware Routing

Efficient layout and cache-aware routing minimize transfer overhead at scale

KV Cache Layout

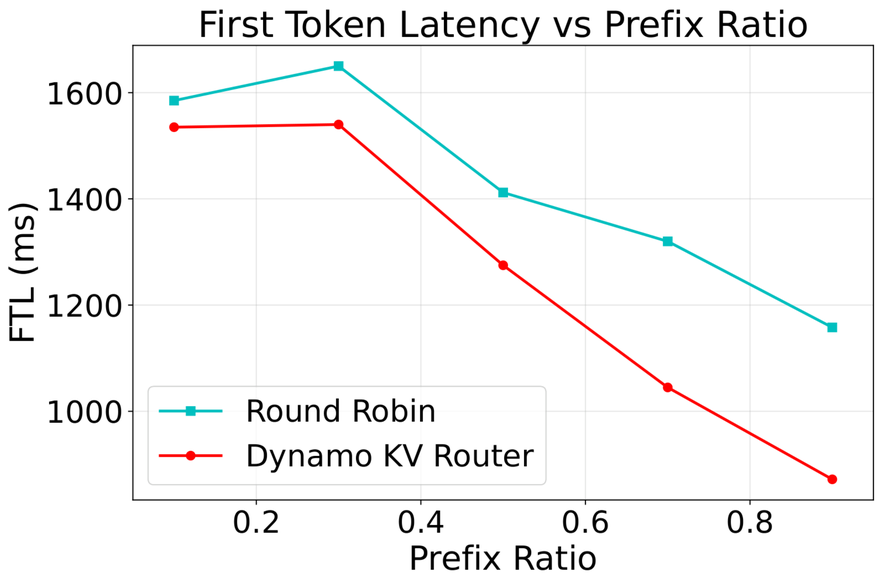
Goal: on-demand allocation, minimal fragmentation, variable ISL support

- Layer-wise layout: KV blocks grouped by transformer layer
  - Enables layer-by-layer KV transfer overlapped with prefill computation
- Head-wise blocking: KV cache organized per attention head for fine-grained allocation and management
- Selective transfer: only non-cached KV blocks transferred, minimizing BW

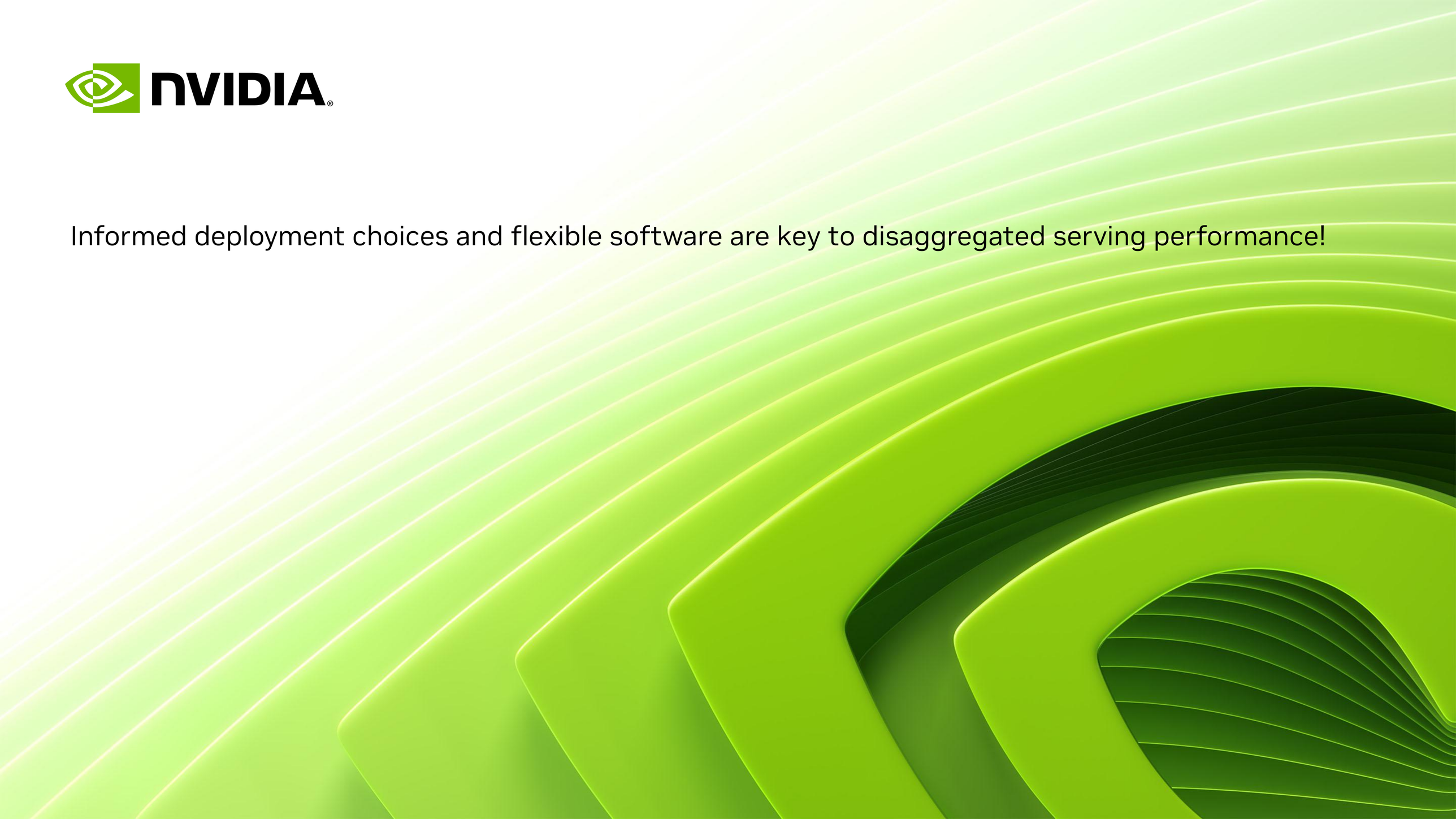
KV-aware Routing

Goal: Route requests to where KV cache resides to maximize performance

- **Cache-oblivious routing (round-robin):**
  - Fragments KV state across workers
  - Cache misses increase, resulting in longer FTLs.
- **Cache-aware routing (Dynamo KV Router):**
  - Routes each request to the worker with the highest prefix cache overlap
  - Maintains stable FTL as prefix-sharing ratio increases (validated on 8x L40S GPUs)
  - Reduces KV transfer volume: only non-cached blocks are transferred to decode workers



FTL vs. prefix ratio. KV-aware routing maintains stability vs. round-robin.


Informed deployment choices and flexible software are key to disaggregated serving performance!