



CONFERENCE

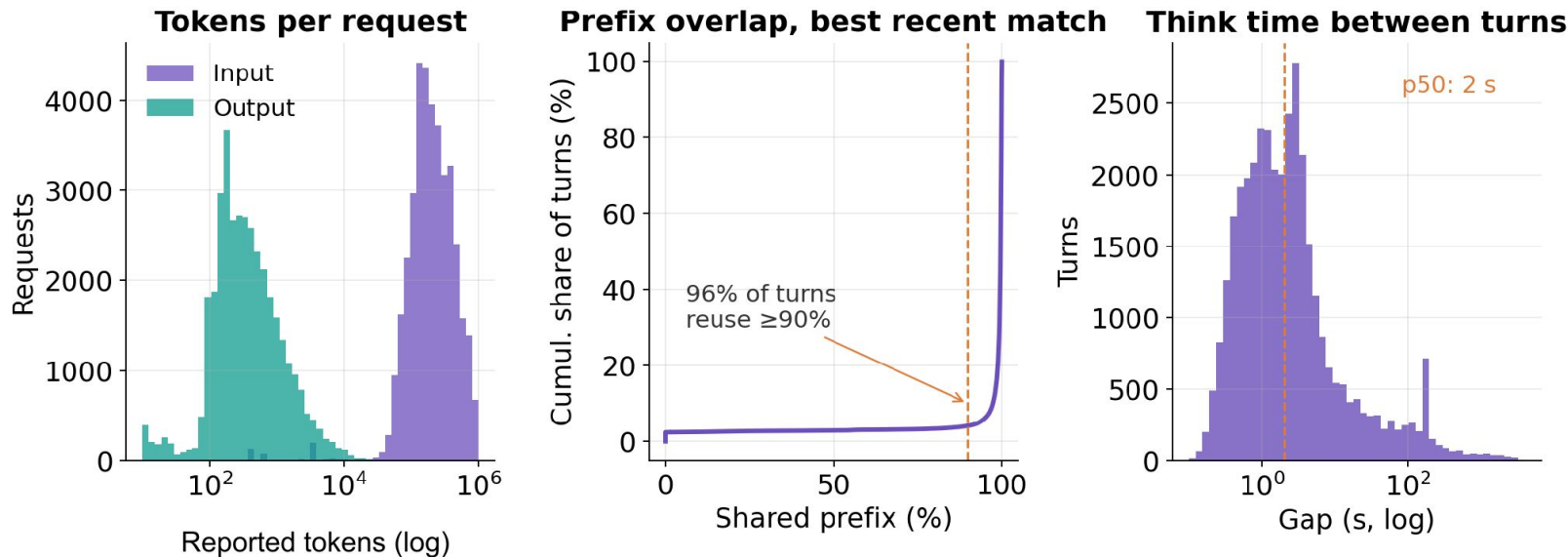
—NORTH AMERICA 2026—

Making Enterprise Agentic Inference Production Ready

Tyler Michael Smith & Joseph Groenenboom

What do agentic workloads look like?

Long context, multiturn, large MoEs

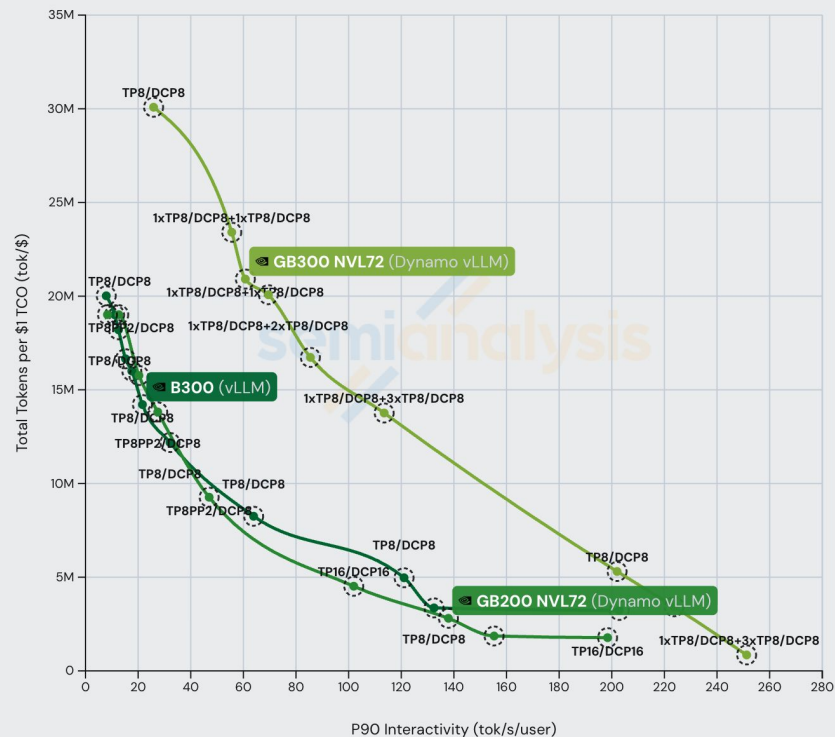


Kimi K3 2.8T Agentic Total Tokens per \$1 TCO vs. P90 Interactivity

Cost Tier: Owning at Large Hyperscaler Volume Updated: 2026-10-15 Source: SemiAnalysis InferenceX™

TCO \$/chip/hr: B300: 2.26 GB200: 1.86 GB300: 2.31

Source: [SemiAnalysis Market July 2026 Pricing Surveys & AI Cloud TCO Model](#)



inferencex.semianalysis.com/inference/kimi-k3

What matters for agentic inference?

- Operations
- Cost
- Interactivity
- Correctness
- Development Velocity

Cost Tier: Owning at Large Hyperscaler Volume Updated: 2026-10-15 Source: SemiAnalysis InferenceX™

Source: [SemiAnalysis Market July 2026 Pricing Surveys & AI Cloud TCO Model](#)

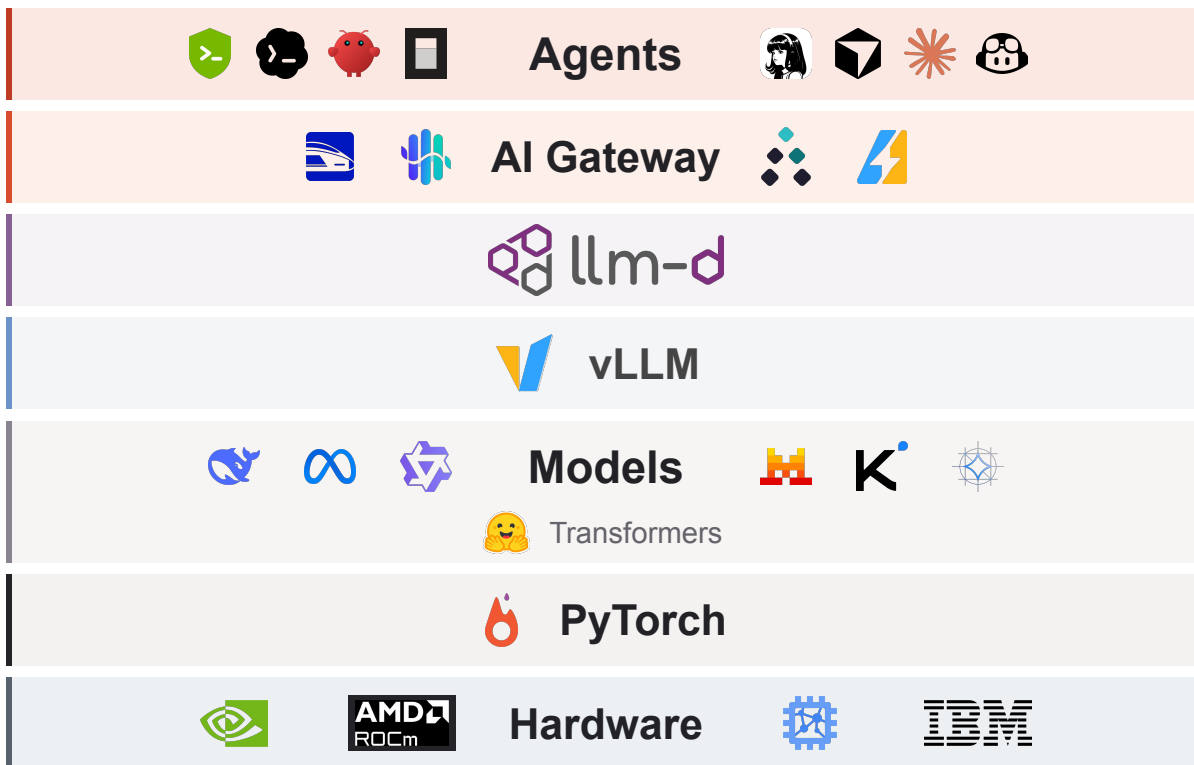


- Y-axis: Cost or throughput per GPU
- X-axis: Interactivity per request (how fast is an agent?)

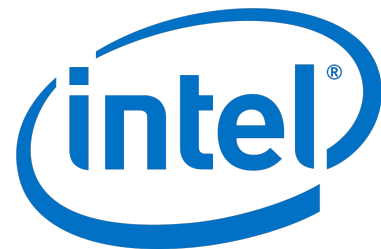
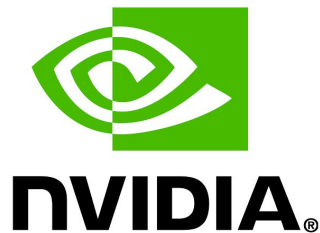
Compare different configs on the same chart

- Points have different configurations and different amounts of load

The stack: from hardware to agent



PyTorch Out of the Box Hardware Support





ExecuTorch

PyTorch Distributed

| Distributed APIs

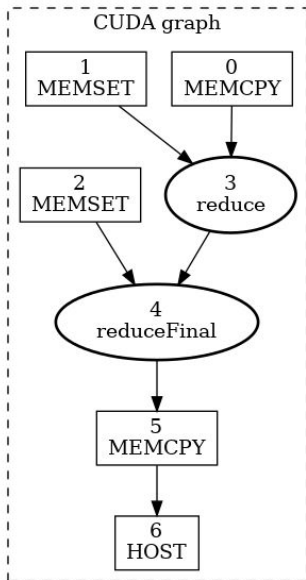
- Distributed Data-Parallel (DDP)
- Fully Sharded Data-Parallel (FSDP2)
- Tensor Parallel (TP)
- Pipeline Parallel (PP)

| Natively Supported Backends

- NCCL
- RCCL
- Gloo
- MPI

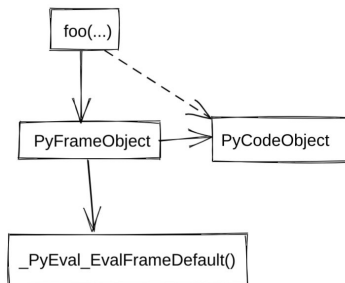
PyTorch Model Acceleration

- Graphs

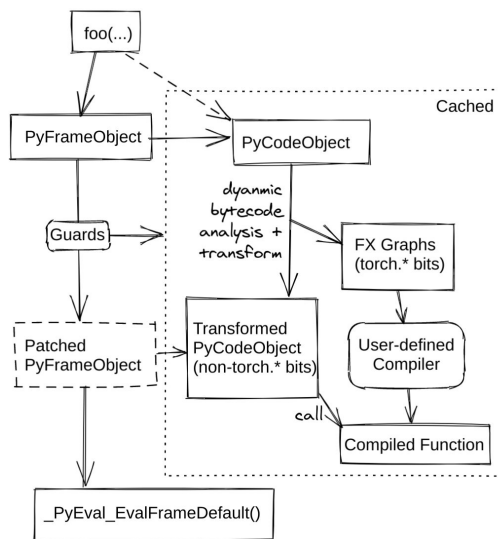


- Compile

Default Python Behavior



TorchDynamo Behavior



Hugging Face Transformers



 LLM

 SGL MLX

LLaMA 

Transformers

Transformers has become the **backbone of open-source AI**, giving everyone a standardized toolkit.



Standardized Toolkit

Provides consistent, high-level APIs and architecture definitions across NLP, computer vision, and multi-modal models.



Ecosystem Backbone

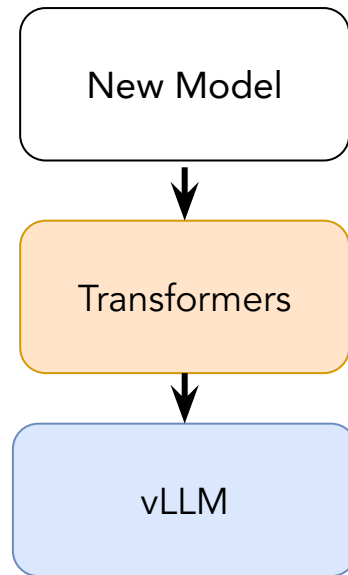
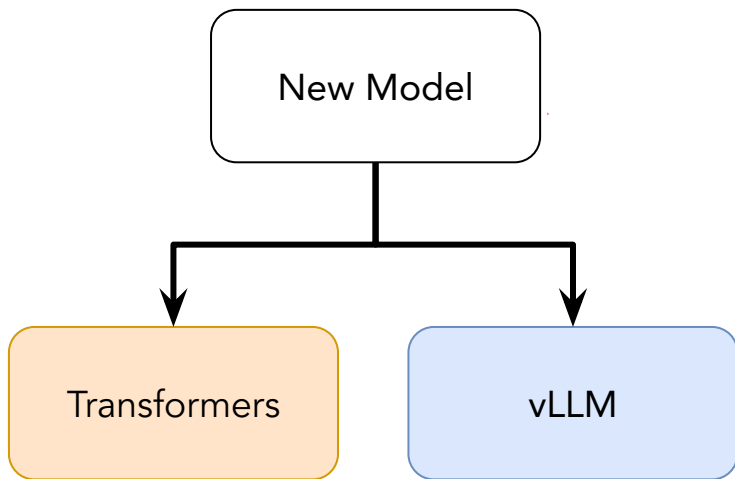
Seamlessly connects with PyTorch, vLLM, and enterprise inference frameworks for efficient deployment.



Open Innovation

Empowers global developer collaboration, rapid research iteration, and open-weights model distribution.

Transformers and vLLM



vLLM: From Research to Standard

In 3 years, vLLM has rapidly evolved from a UC Berkeley research project into the leading OSS engine

Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon^{1,*} Zhuohan Li^{1,*} Siyuan Zhuang¹ Ying Sheng^{1,2} Lianmin Zheng¹ Cody Hao Yu³
Joseph E. Gonzalez¹ Hao Zhang⁴ Ion Stoica¹

¹UC Berkeley ²Stanford University ³Independent Researcher ⁴UC San Diego

Abstract

High throughput serving of large language models (LLMs) requires batching sufficiently many requests at a time. However, existing systems struggle because the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically. When managed inefficiently, this memory can be significantly wasted by fragmentation and redundant duplication, limiting the batch size. To address this problem, we propose PagedAttention, an attention algorithm inspired by the classical virtual memory and paging techniques in operating systems. On top of it, we build vLLM, an LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage. Our evaluations show that vLLM improves the throughput of popular LLMs by 2-4x with the same level of latency compared to the state-of-the-art systems, such as FasterTransformer and Orca. The improvement is more pronounced with longer sequences, larger models, and more complex decoding algorithms. vLLM's source code is publicly available at <https://github.com/vllm-project/vllm>

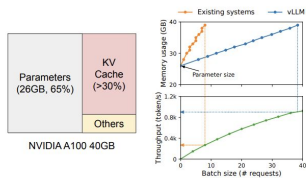
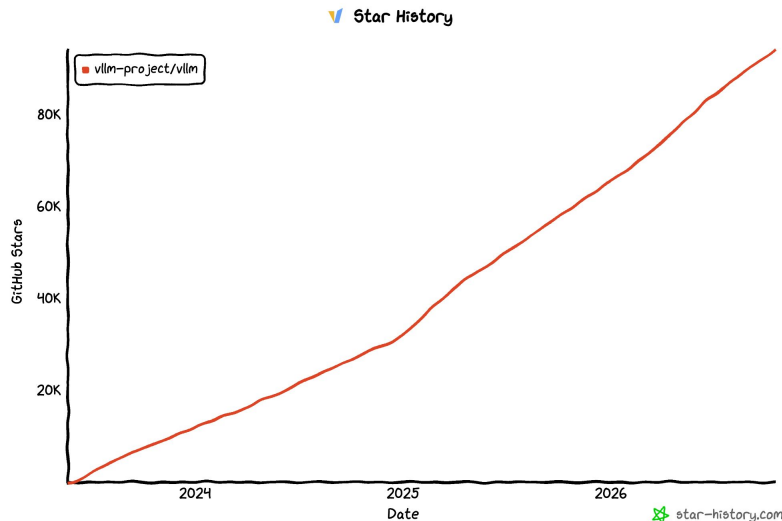


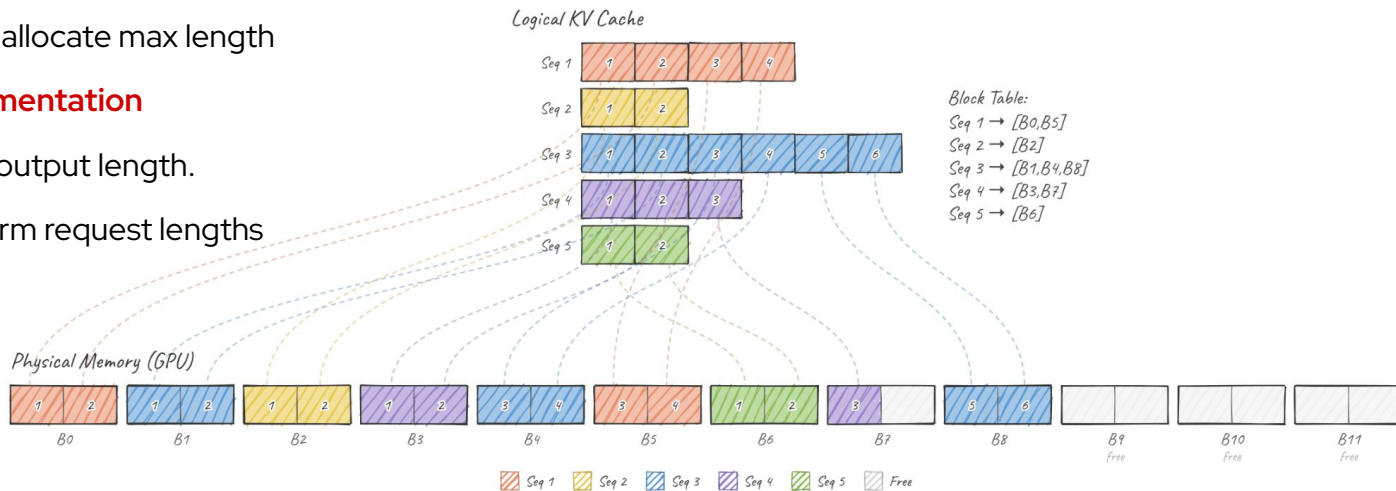
Figure 1. Left: Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. Right: vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.



Paged Attention

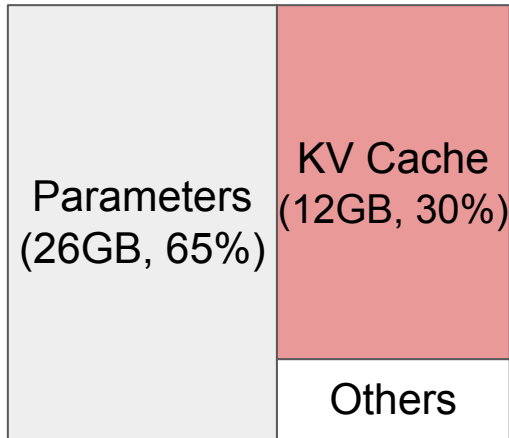
Paged Attention manages KV cache like virtual memory

- Previous approaches **pre-allocated contiguous** memory
 - Bad: must allocate max length
- **Resulted in fragmentation**
 - **unknown** output length.
 - Non-uniform request lengths



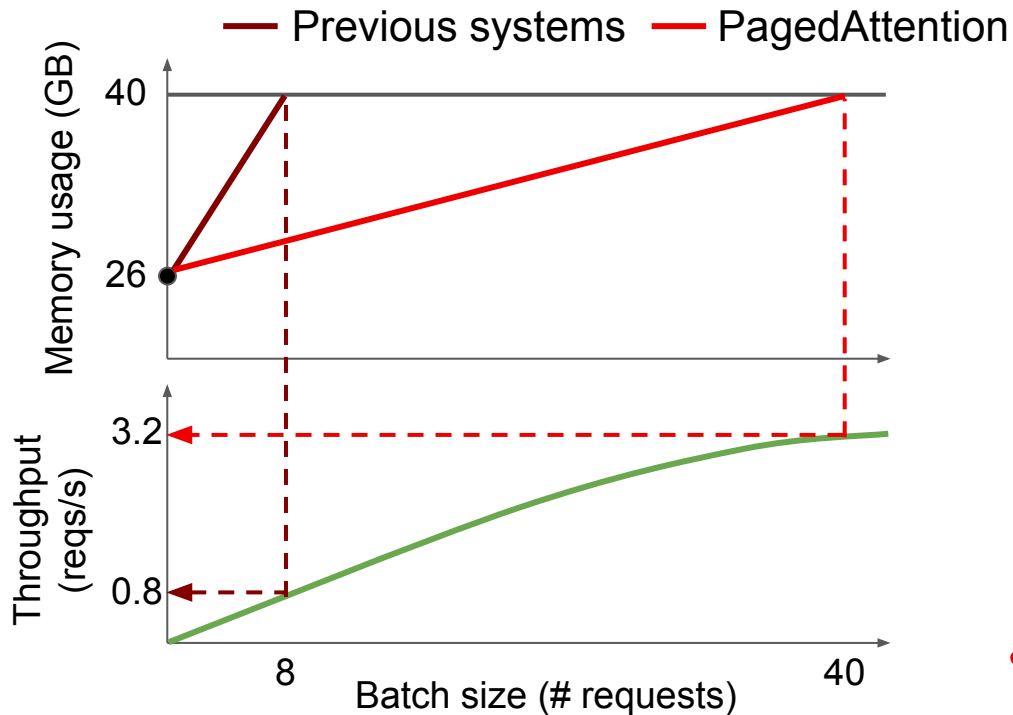
Key insight

Efficient management of KV cache is crucial for high-throughput LLM serving



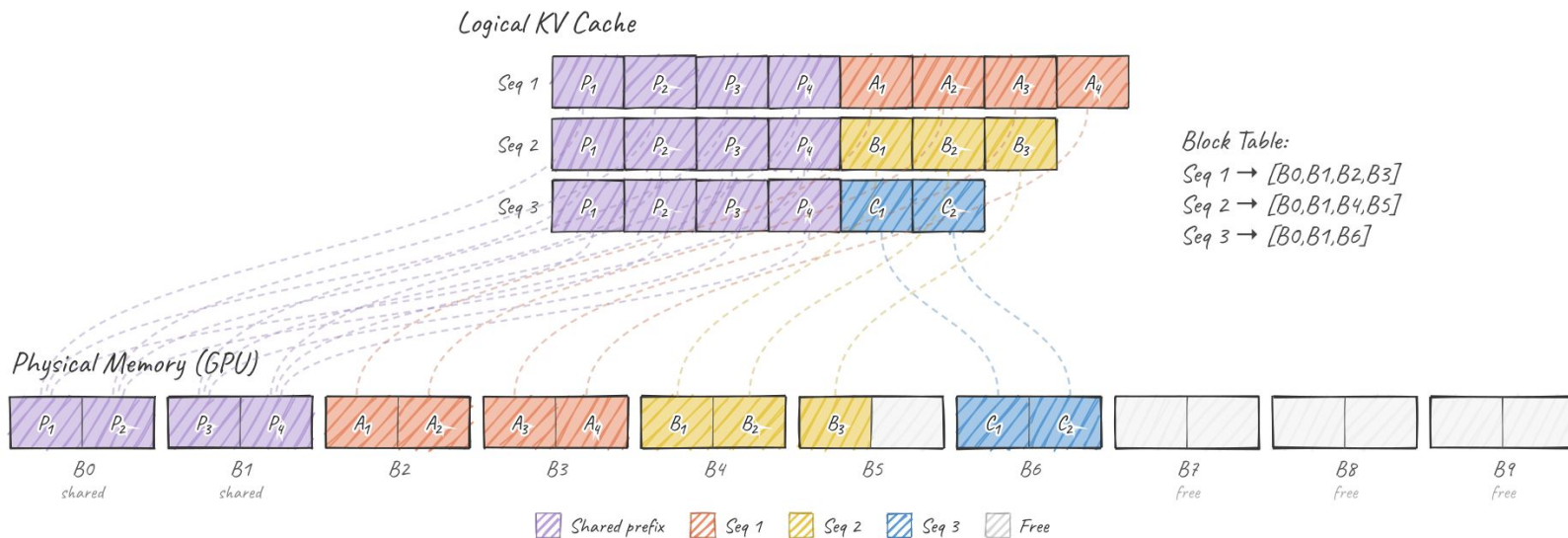
13B LLM on A100-40GB*

*Common setup in 2023



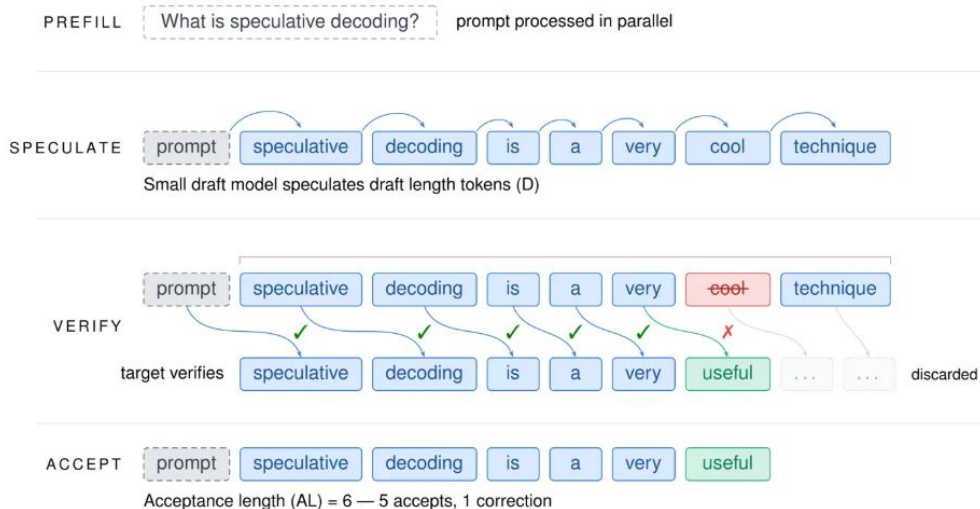
Multi-turn and Forking Agentic Workloads: Prefix caching

Matching prefixes can share the same KV blocks



Speculating allows more work per forward pass

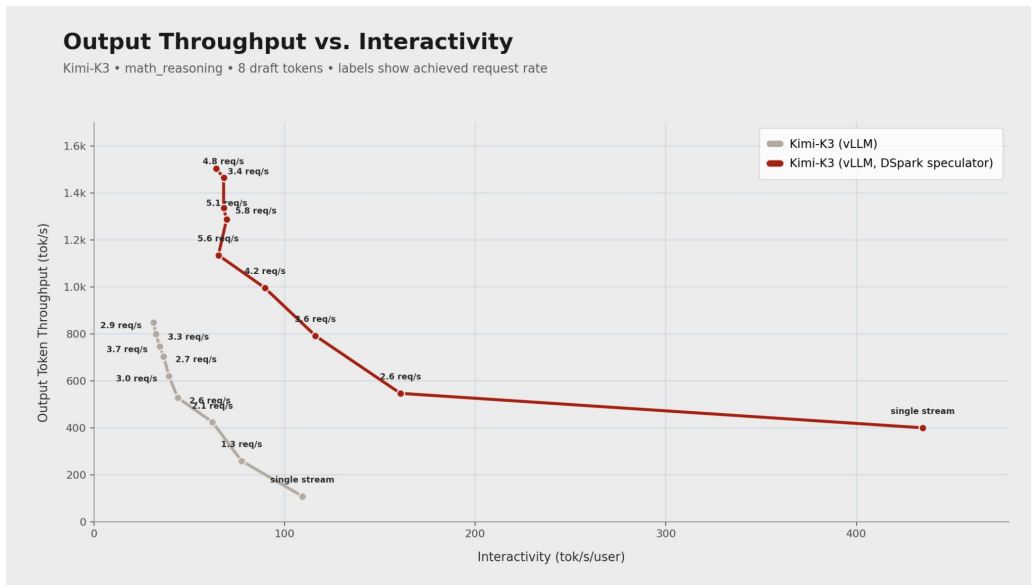
Speculative decoding *speculates* tokens that the main model *verifies*



- Lets the model generate more than one token per forward pass.
- Especially helpful for:
 - Low concurrency, high interactivity servers
 - Sparse MoE models
- Active area of development
 - DFlash, DSpark
 - Today, extra layers rather than separate drafter model

Speculative decoding is now table-stakes for agentic serving

Speculative decoding *speculates* tokens that the main model *verifies*



- Greatly increases maximum interactivity
- Shifts entire Pareto curve to the right
- Large MoEs run at small batch sizes except at far left side of Pareto

Problem: There is a lot to configure!

Use vLLM recipes to make deployment easy

- Quantization
- Parallelization
- Speculative Decoding
- Backend selection
- KV Connectors
 - KV offloading
 - P/D Disaggregation
 - HiSparse

Solution: recipes.vllm.ai

HARDWARE	NVIDIA	H100 80G/GPU	H200 141G/GPU	B200 180G/GPU	GB200 NVL4 192G/GPU	B300 268G/GPU	GB300 NVL4 288G/GPU
	AMD	Mi300X 192G/GPU	Mi325X 256G/GPU	Mi355X 288G/GPU	Mi350X 288G/GPU		
VARIANT	NVFP4 353 GB FP8 604 GB BF16 1204 GB						
	ModelOpt NVFP4 on the routed experts; all other weights BF16, W4A4 on Blackwell; W4A16 through Marlin on Hopper. Recommended on Blackwell and H100. 🔗 reflection/Beam-1.0-501B-A23B-NVFP4 ↗						
STRATEGY	Tensor Parallel ⚡ Tensor + Expert Parallel Data + Expert Parallel						
	Single-node tensor parallel. Splits the model across all local GPUs. TP size is set to the GPU count at deploy time. The simplest multi-GPU strategy — works for all model architectures. Latency oriented						
KV OFFLOAD	Off Simple Offloading Mooncake LMCash						
NODES	Single-node Multi-node (example: 2)						
FRONTEND	⚡ Rust Python						
	The experimental Rust frontend can improve throughput and latency, especially under high concurrency. Switch to Python if you encounter unsupported features or compatibility issues.						
FEATURES	Reasoning Tool Calling DFlash2 (for low latency & small batch size)						
ADVANCED — performance tuning							



vLLM brings you new models, **ready on day 0**



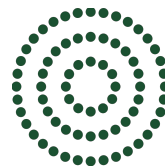
Support when new weights launch.

recipes.vllm.ai

Pick your model and GPUs.
Copy the launch command.



Muse Spark Open



Reflection Beam



Kimi K3



DeepSeek-V4.1

Flash



Qwen3.8



Nemotron 3.5



GLM-5.2



MiniMax M3



Gemma 4



gpt-oss

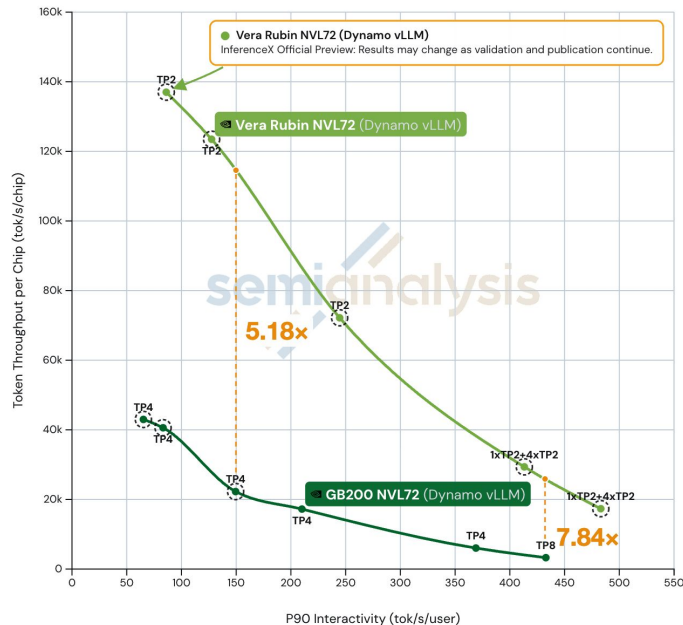


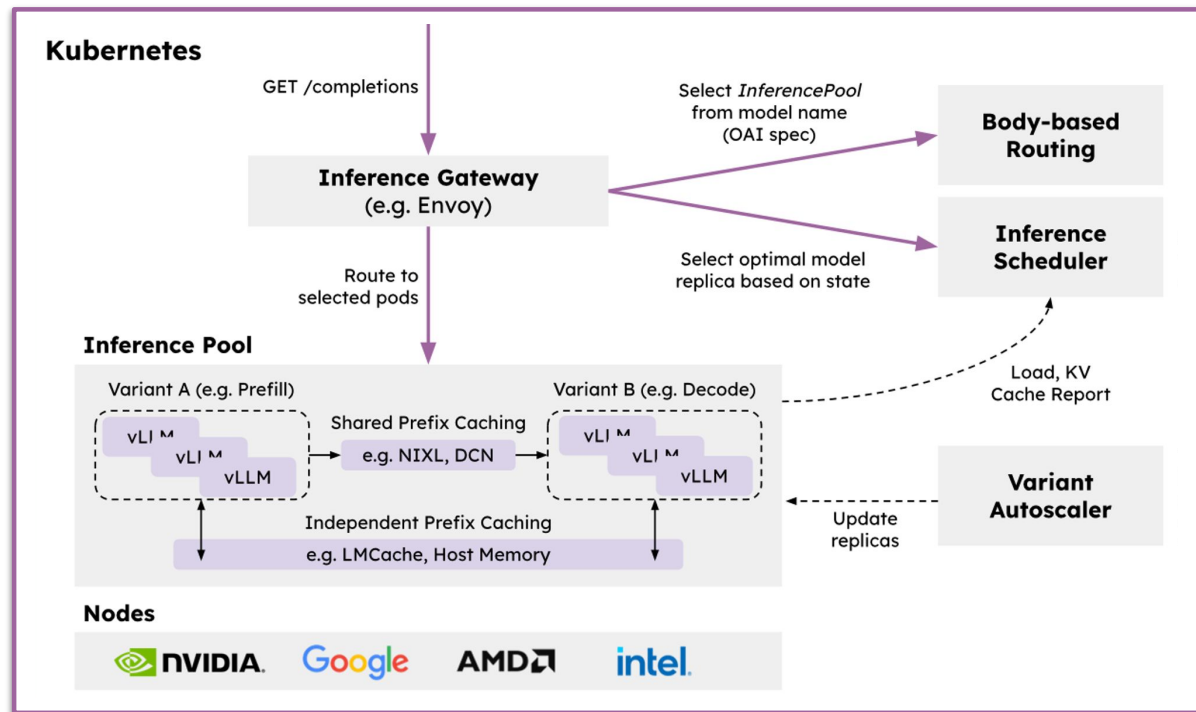
Day-0 Support for Vera Rubin

The vLLM community works with hardware to prepare early

- The vLLM community prepared early for Rubin
 - Joint: NVIDIA, Red Hat, Inferact
- Preview image available now:
 - `vllm/vllm-openai:cu134-nightly`
- Much easier than Hopper -> Blackwell
 - FlashInfer: NVIDIA's investment means there are already open and optimized kernels available
 - Smaller architectural differences made it easier this time

MiniMax M3 428B Agentic Token Throughput per Chip vs. P90 Interactivity
Updated: 2026-10-08 Source: SemiAnalysis InferenceX™



**Operationalizability****Flexibility****Performance**

“Well-lit” Paths

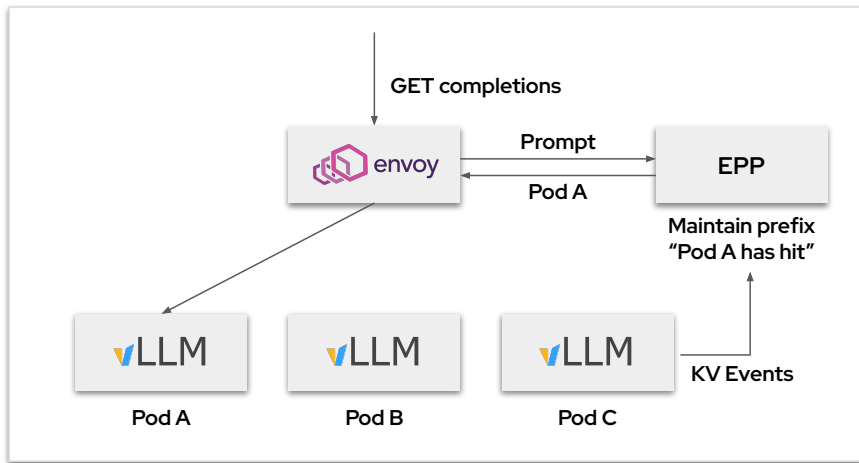
llm-d provides “well-lit” paths for running LLM inference workloads

- ▶ Intelligent Inference Scheduling
- ▶ KV Cache Management
- ▶ P/D Disaggregation
- ▶ Wide Expert Parallelism
- ▶ Request Prioritization

Intelligent Inference Scheduling

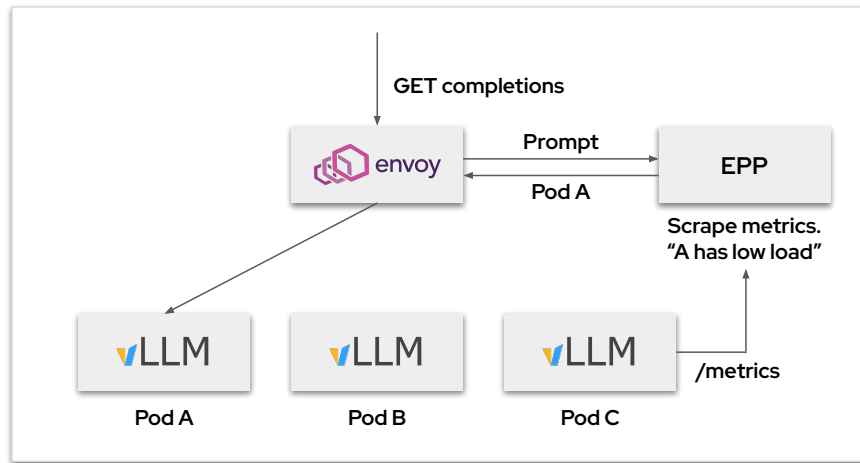
vLLM-aware load-balancing enables smarter request routing that improve SLOs

Prefix-Aware Routing



Dramatically increase prefix-cache hit rate

Load-Aware Routing

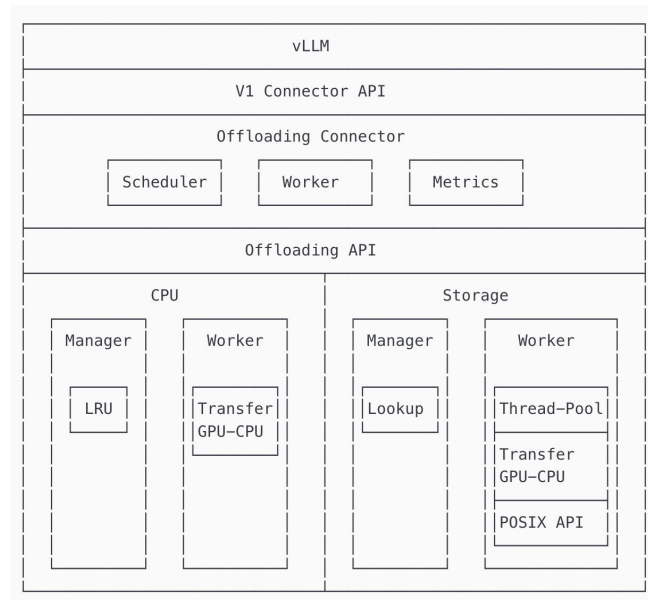


Load-balancing based on actual replica state

KV Offloading

Reuse more prompts in KV cache to avoid recomputation

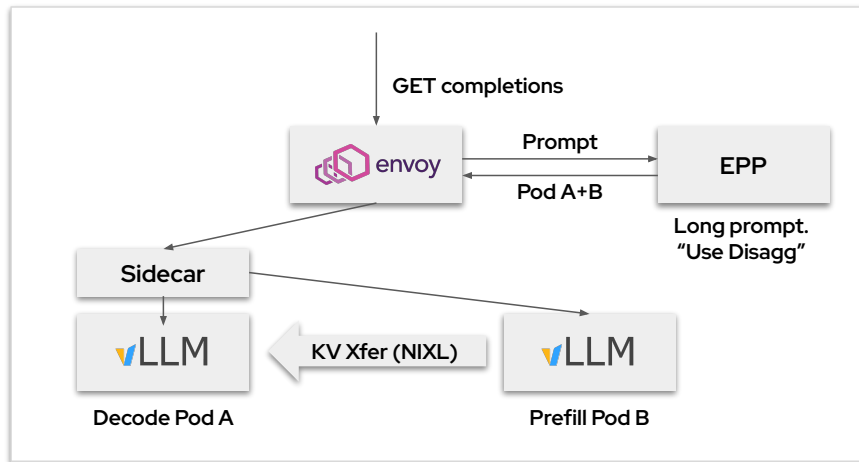
- vLLM natively supports KV offloading via the KV Connector API
- Tiered offloading - KV cache hierarchy
 - CPU memory
 - Local NVMe
 - Remote storage
- Filesystem and object storage
- Offloading allows larger prompt caches but not higher batch size
 - Doesn't help decode throughput



P/D Disaggregation

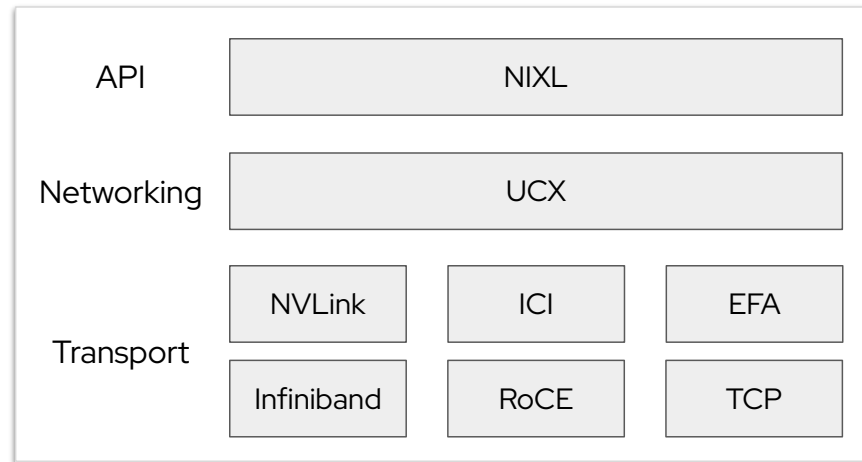
P/D disaggregation is a key optimization for demanding inference workloads

Disaggregated Serving



Specialization of pods for prefill and decode phase

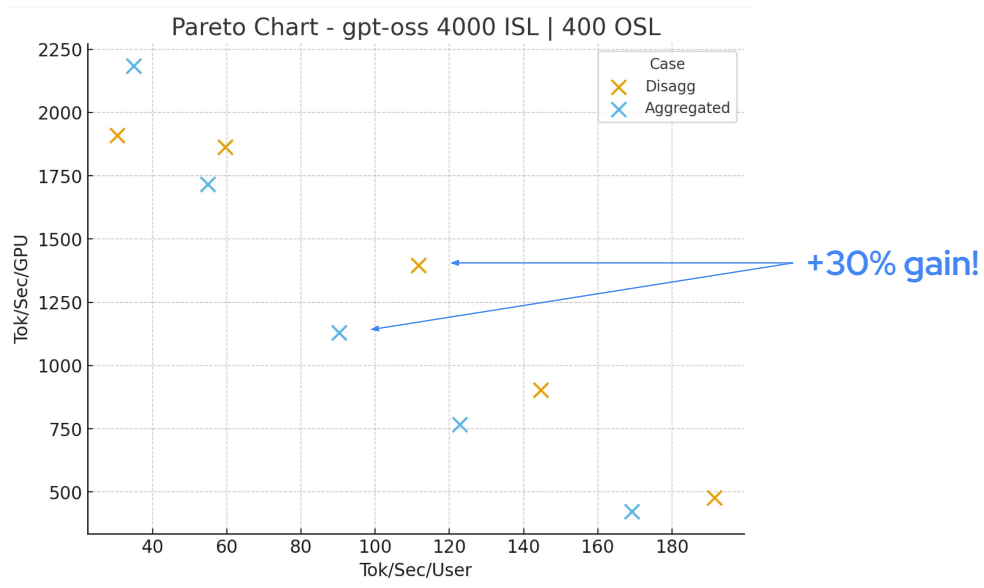
Heterogenous Transfer Protocols



Support variety of transports via UCX

P/D Disaggregation

P/D disaggregation drives significant performance improvements for long context workloads

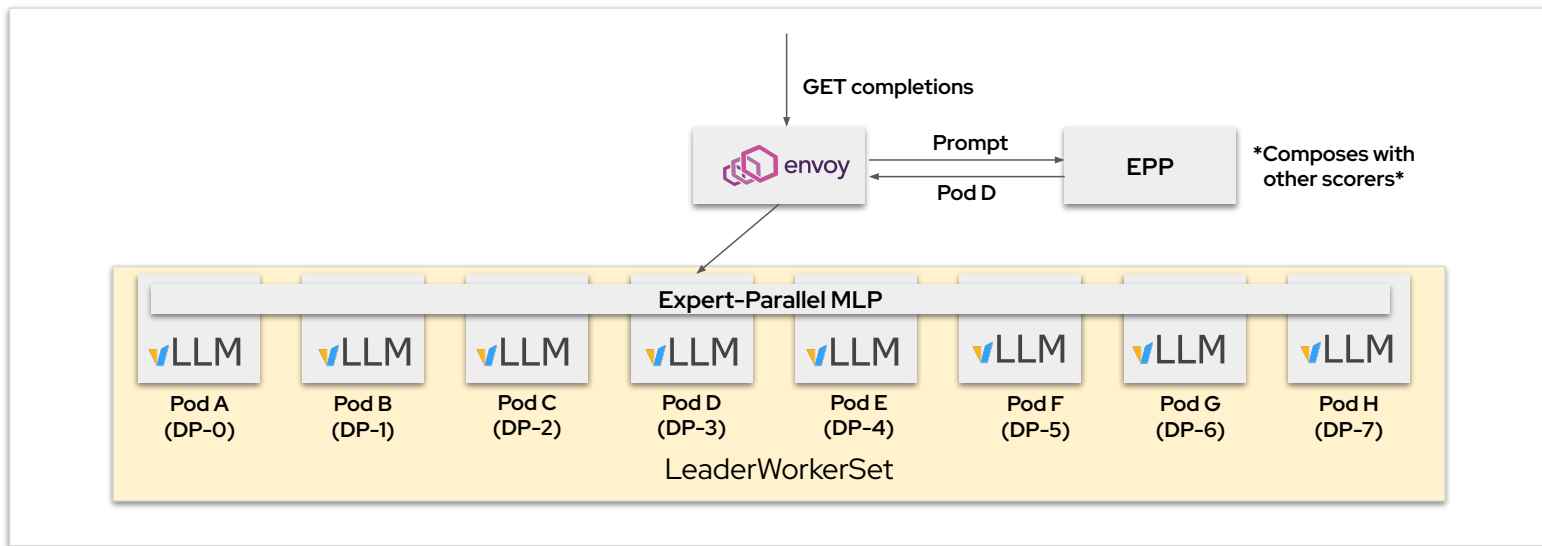


Key Insights

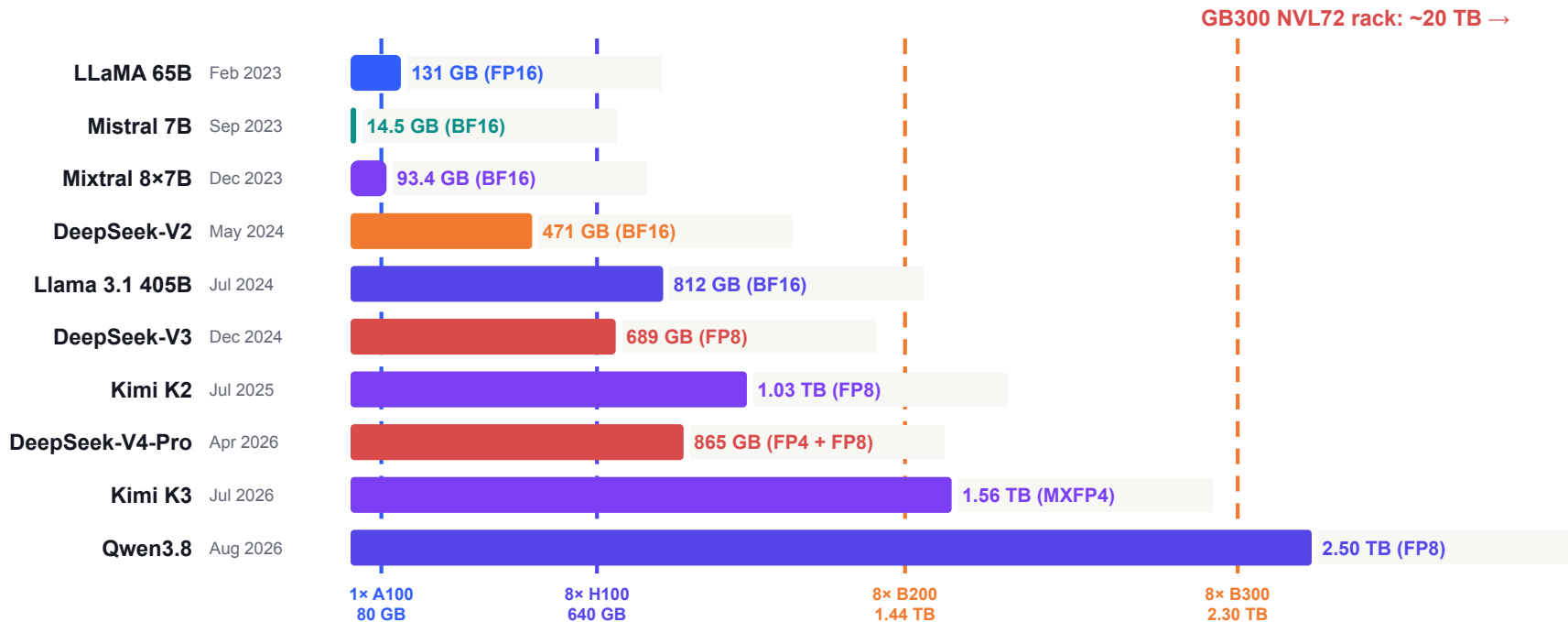
- Important for long context
- Disaggregation unlocks specialization for phases of inference
 - Decode scales better with parallelism than Prefill
 - Decode needs more KV Cache space
- Need to tune xPyD ratios for workload

Wide Expert Parallelism

llm-d's K8s-native design composes the EP implementation with the rest of the llm-d system

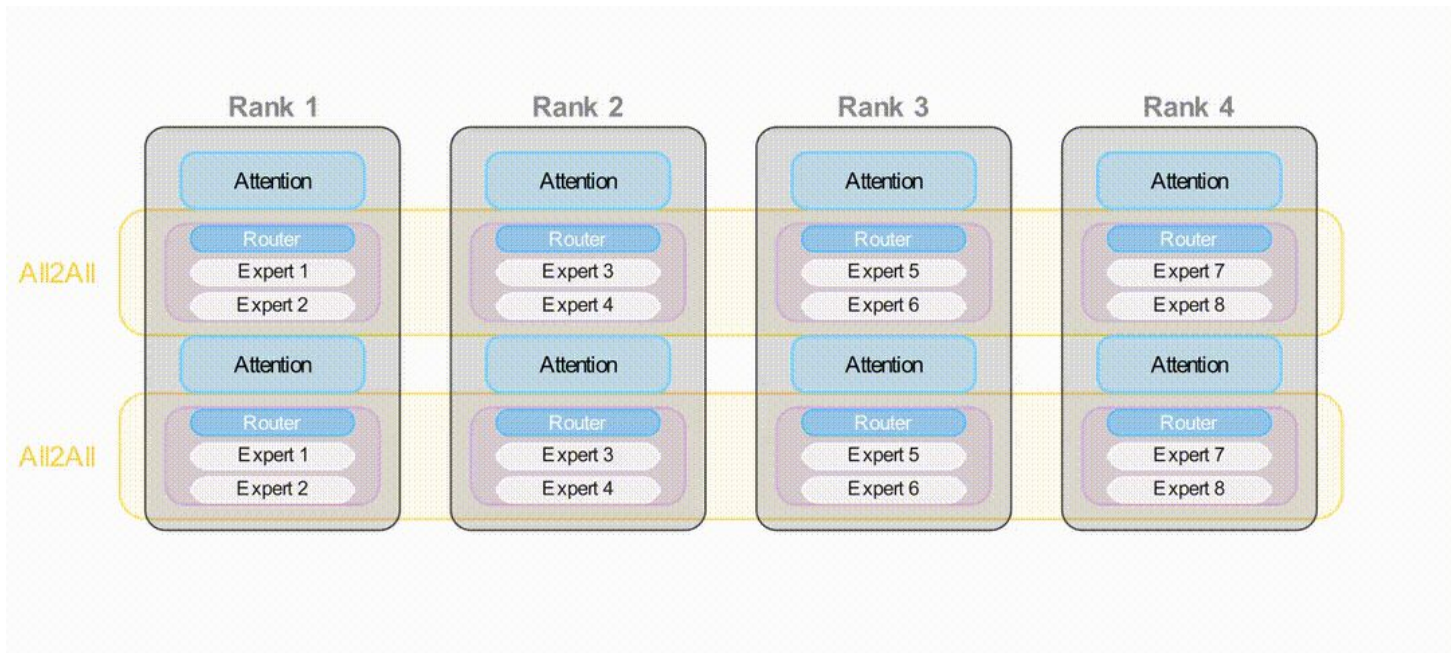


Large-Scale Serving: Model weights now reach terabytes



Released weights still outgrow a node, even at low precision.

Wide expert parallelism: attention data-parallel, experts spread across ranks



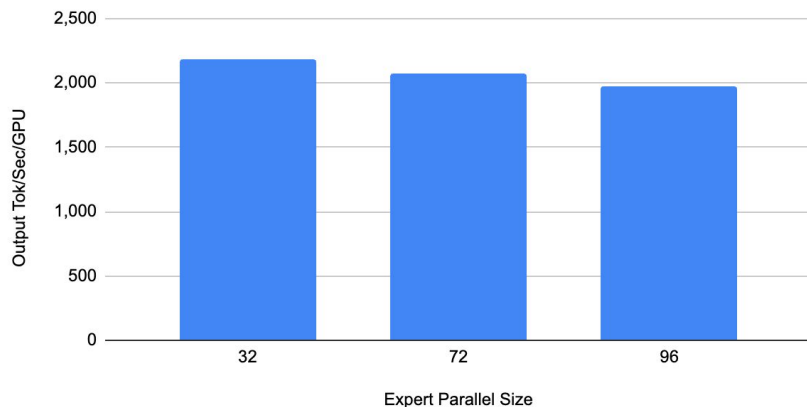
Wide EP supports thousands of concurrent requests

DeepSeek-R1, H200: 256 requests/rank $\times$ 96 ranks = **24,576** requests

Reached 2.2k decode tokens/s **per H200 GPU**

Decode Throughput

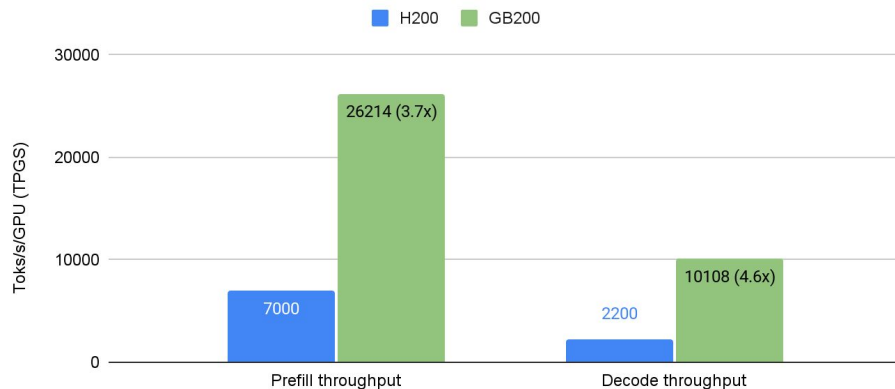
2000 ISL, 2000 OSL (avg), 256 Concurrent Requests / EP Rank



Reached 10K decode tokens/s **per GB200 GPU**

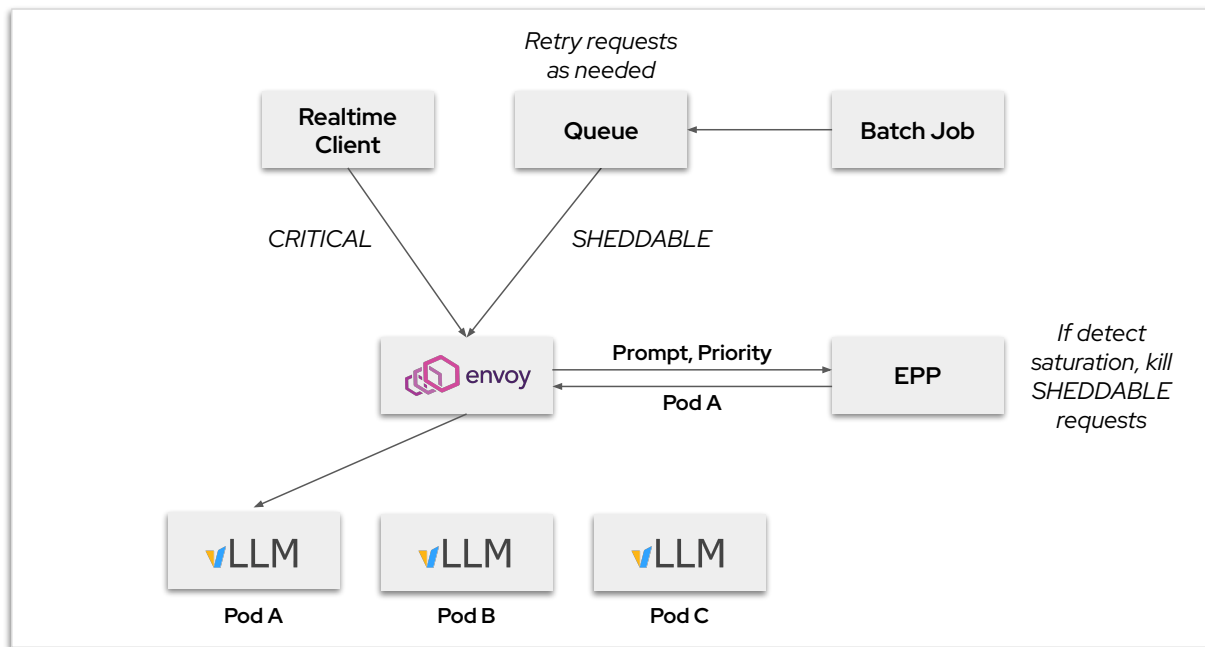
Prefill and Decode Throughput Comparison between H200 and GB200

2000 ISL, 2000 OSL



Request Prioritization

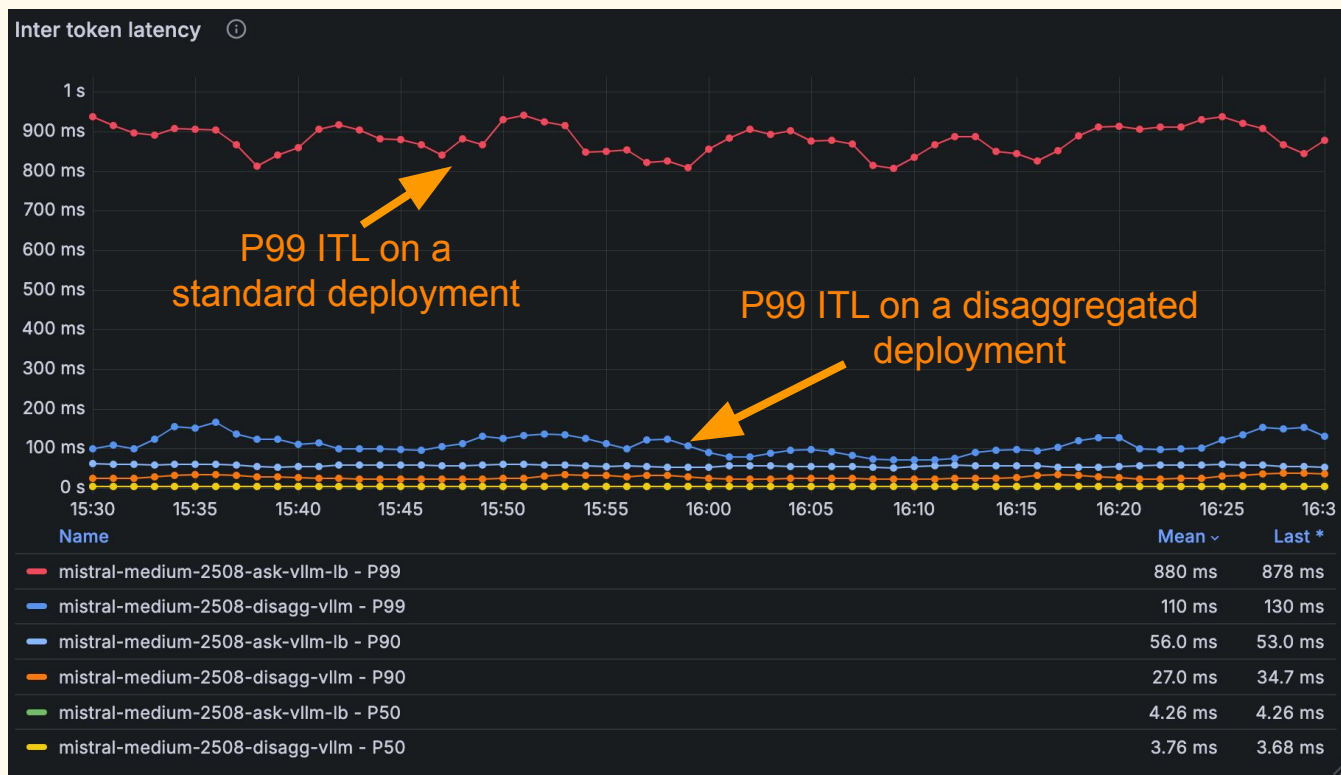
llm-d enables multiplexing of different request classes onto the same model deployment



Thank you!

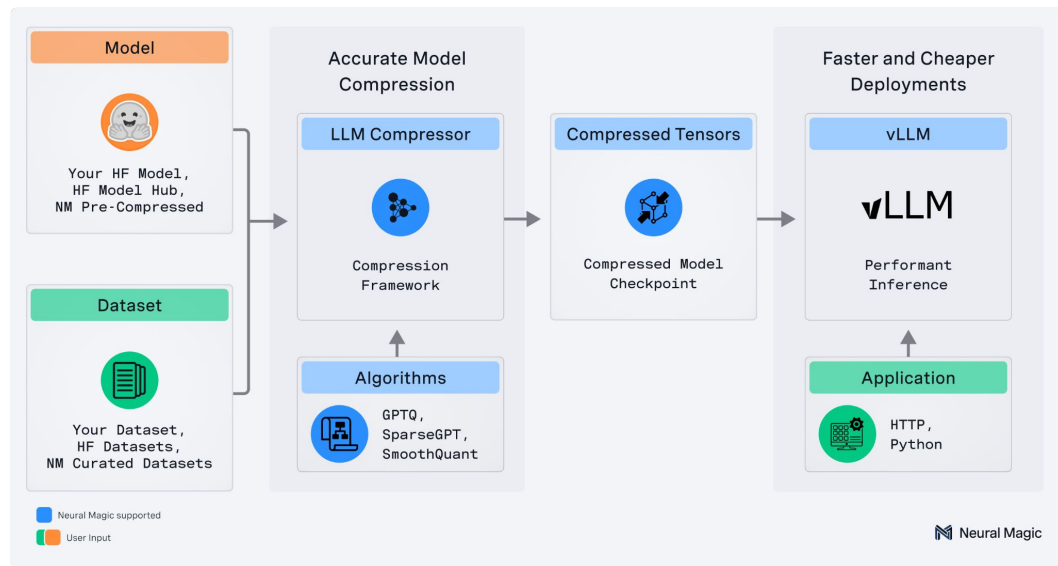
Backup Slides

The goal:



vLLM's llm-compressor quantizes models

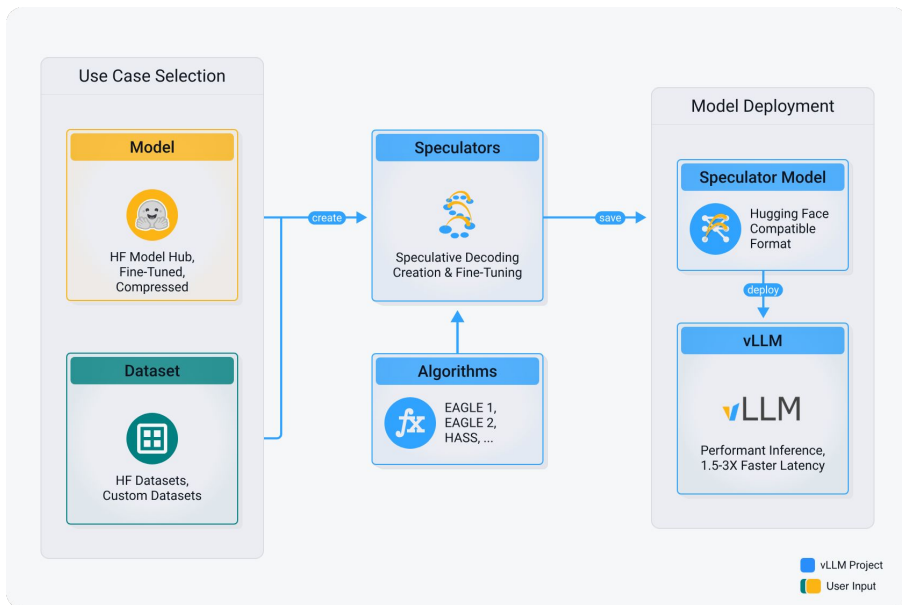
Quantization uses lower precision weights to shrink footprint and compute faster



- [vllm-project/llm-compressor](#)
- Library for compressing models via quantization
- Red Hat releases many quantized checkpoints
- Different hardware need different data types
 - CPU - int8 and uint8
 - NVIDIA Blackwell - NVFP4
 - Hopper - FP8

vLLM's Speculators trains speculator models

Speculative decoding *speculates* tokens that the main model *verifies*



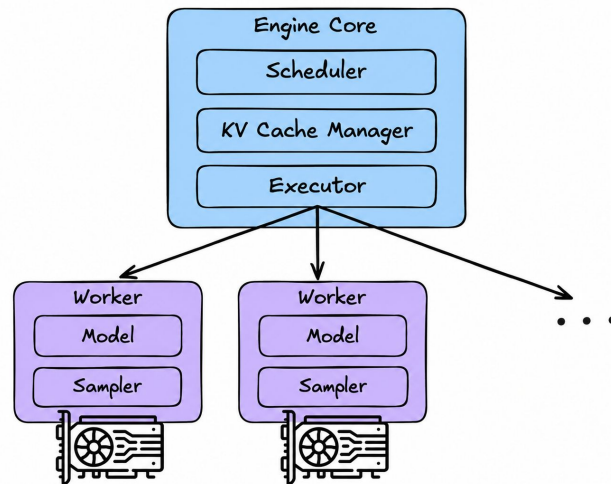
- [vllm-project/speculators](https://github.com/vllm-project/speculators)
- Library for training speculator models
- Many speculators-trained models available
 - [RedHatAI/GLM-5.3-speculator.dspark](#)
 - [RedHatAI/Kimi-K3-speculator.dspark](#)
 - [Inferact/Muse-Spark-Open-DSpark](#)

vLLM Optimizes for Throughput and Latency

vLLM aims for the speed of light

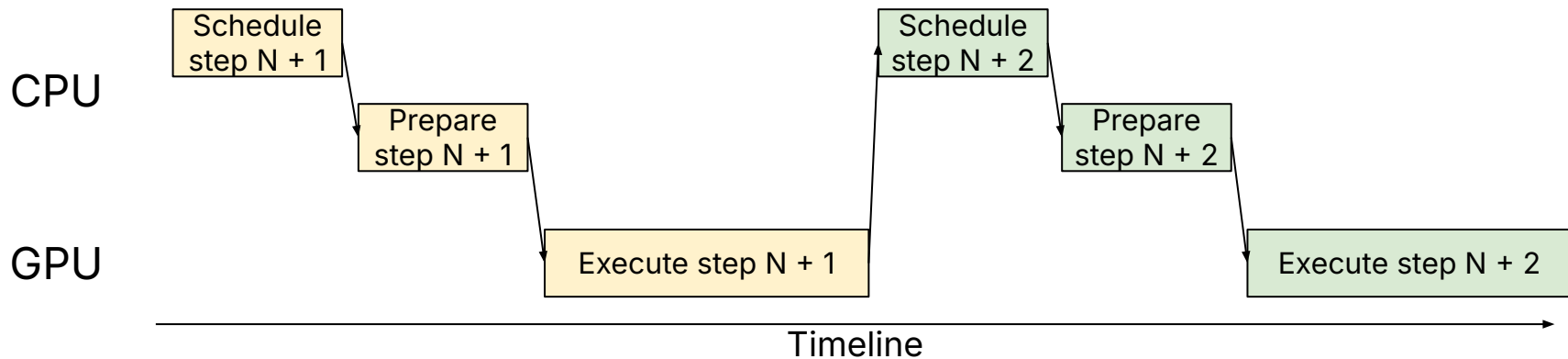
Reaching the speed of light requires optimizing the entire engine

- Eliminate CPU overhead
- Optimize the model execution
- Quantization
- Speculative decoding



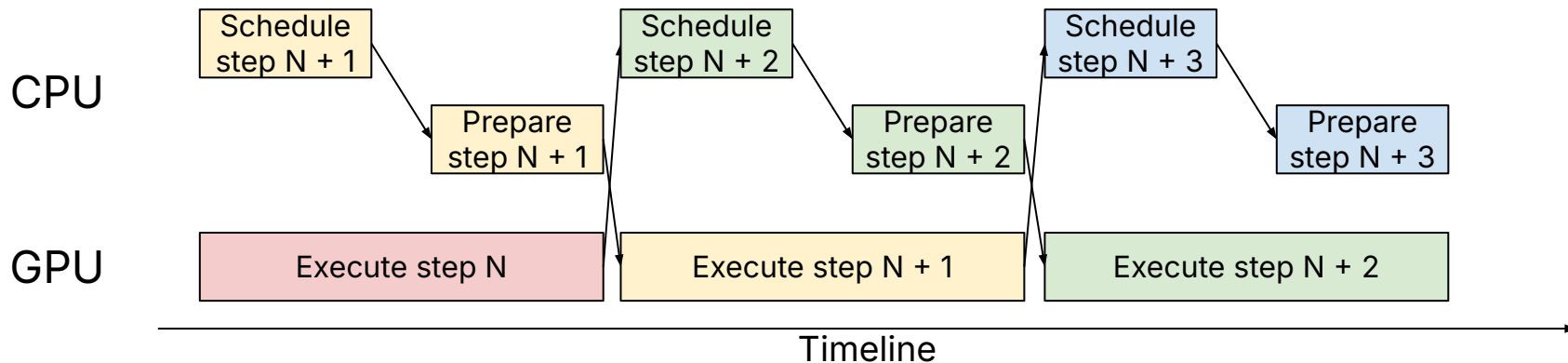
Synchronous Scheduling

Scheduling & input preparation happens on the critical path, making GPU idle



Core Optimization: Asynchronous Scheduling

vLLM overlaps the host overheads with model execution



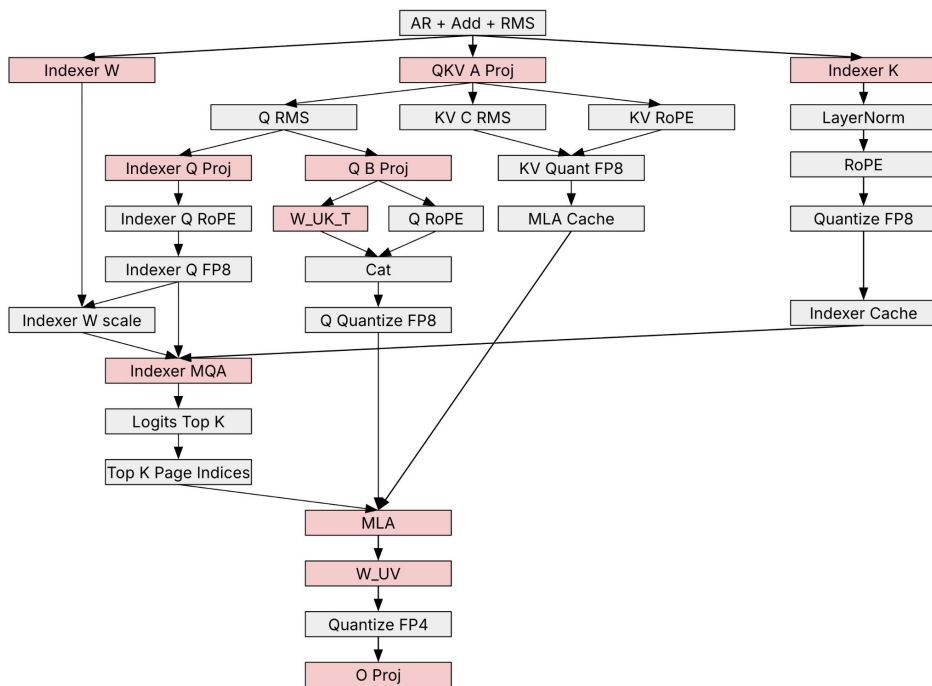
- Core idea: “schedule & prepare the next batch one step ahead”
- vLLM ensures *zero synchronization*,
 - The GPU is never stalled by the CPU in *any case*



Model Optimization: Asynchronous Scheduling

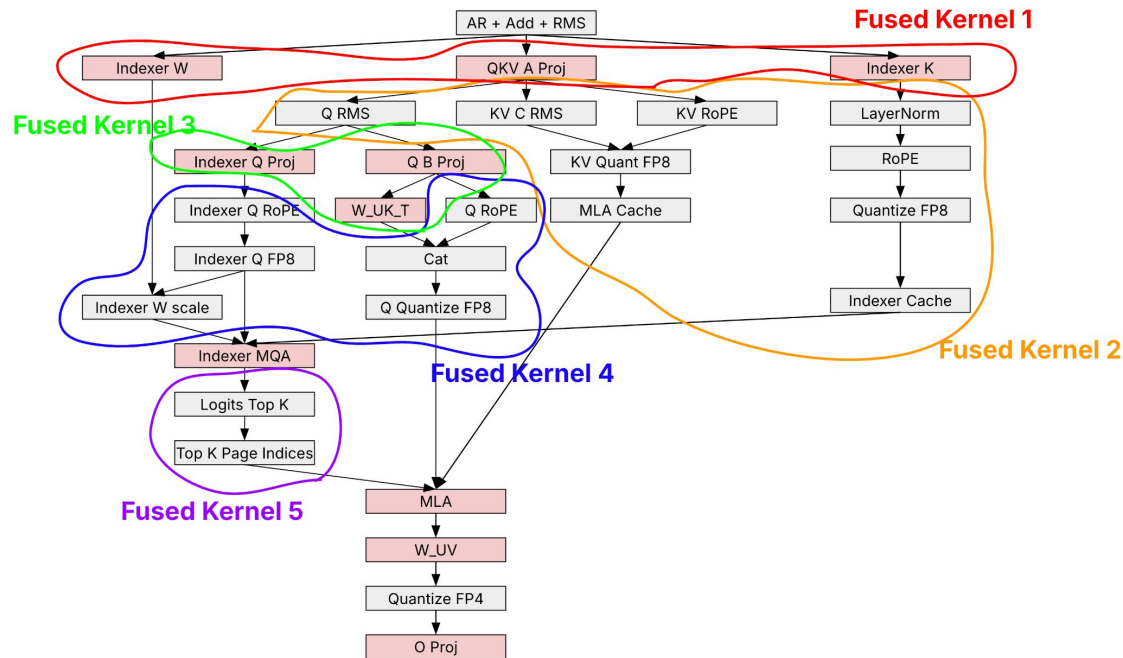
Fusion avoids kernel startup overheads and allows overlapping of different costs

Example: DeepSeek V3.2

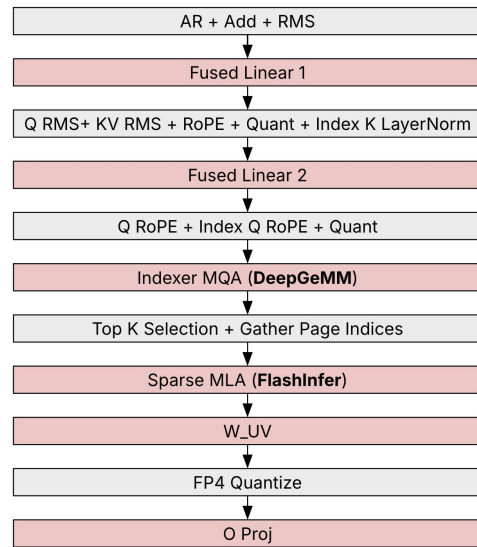


Model Optimization: Asynchronous Scheduling

Fusion avoids kernel startup overheads and allows overlapping of different costs



Example: DeepSeek V3.2

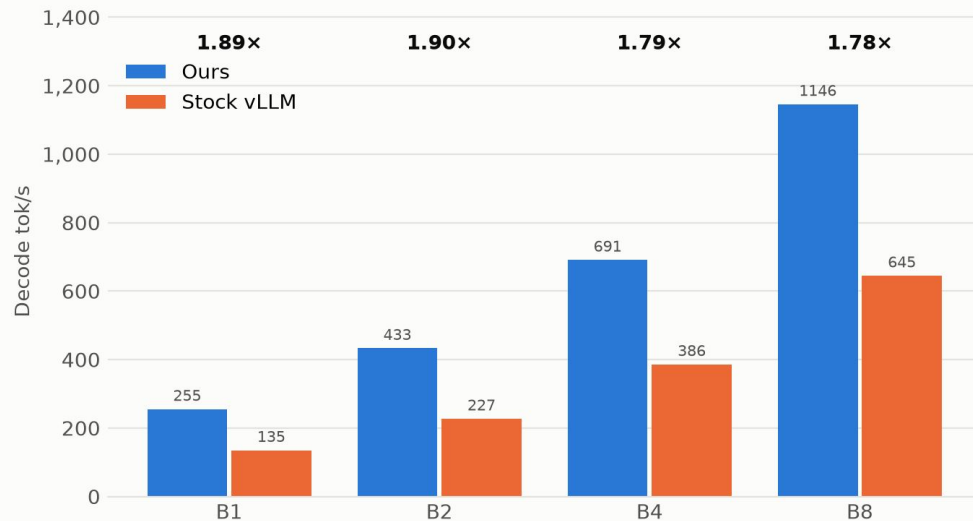


Frontier fusion: Mega-Kernels

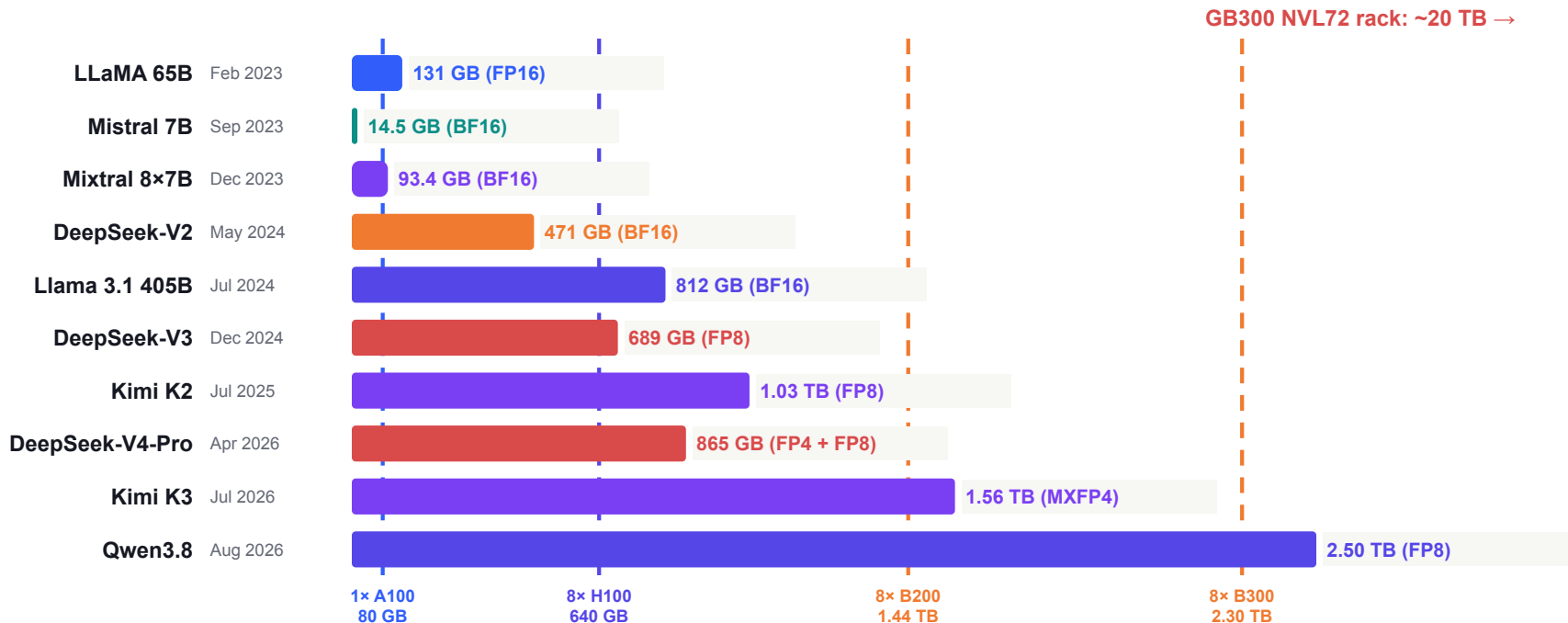
Implement the entire forward pass in one kernel

- Mega-kernels take the idea of fusion to the extreme
- Huge interactivity wins for Kimi K3 decode

TP16, no spec decoding, ISL 1, greedy



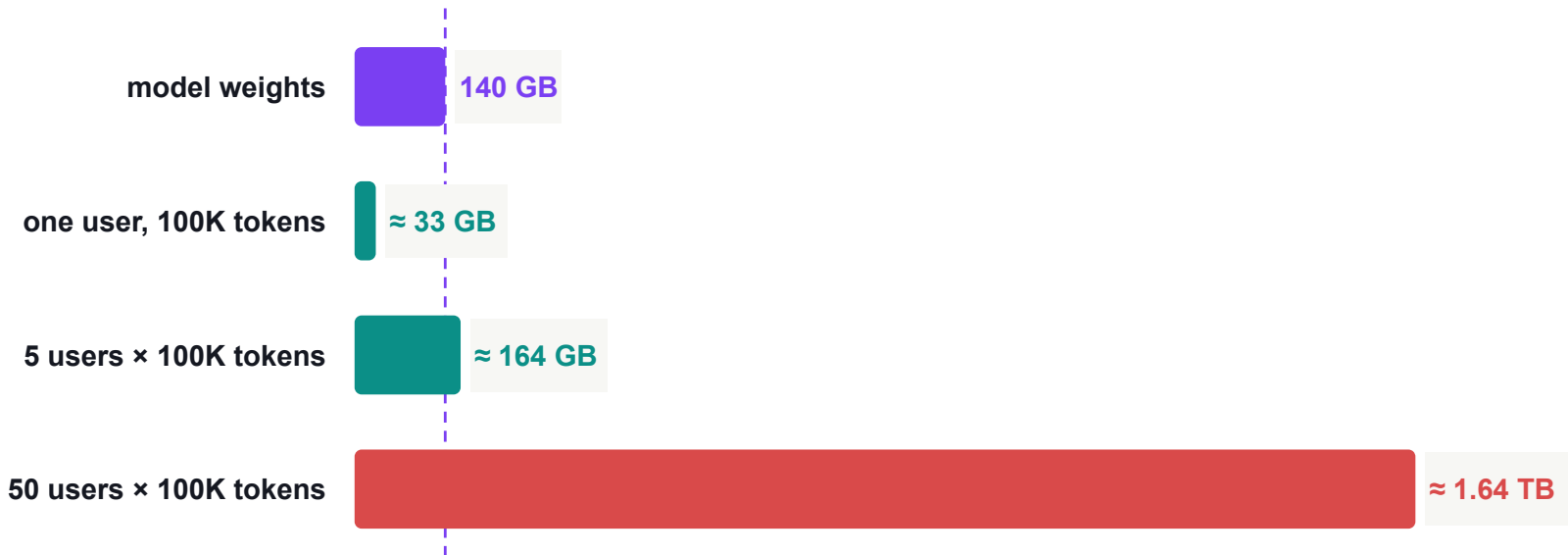
Model weights now reach terabytes



Released weights still outgrow a node, even at low precision.

Serving many users can need more KV memory than the model weights

Llama 3.1 70B, BF16, 80 layers with 8 KV heads + head size 128 → 320 KiB of KV cache per token

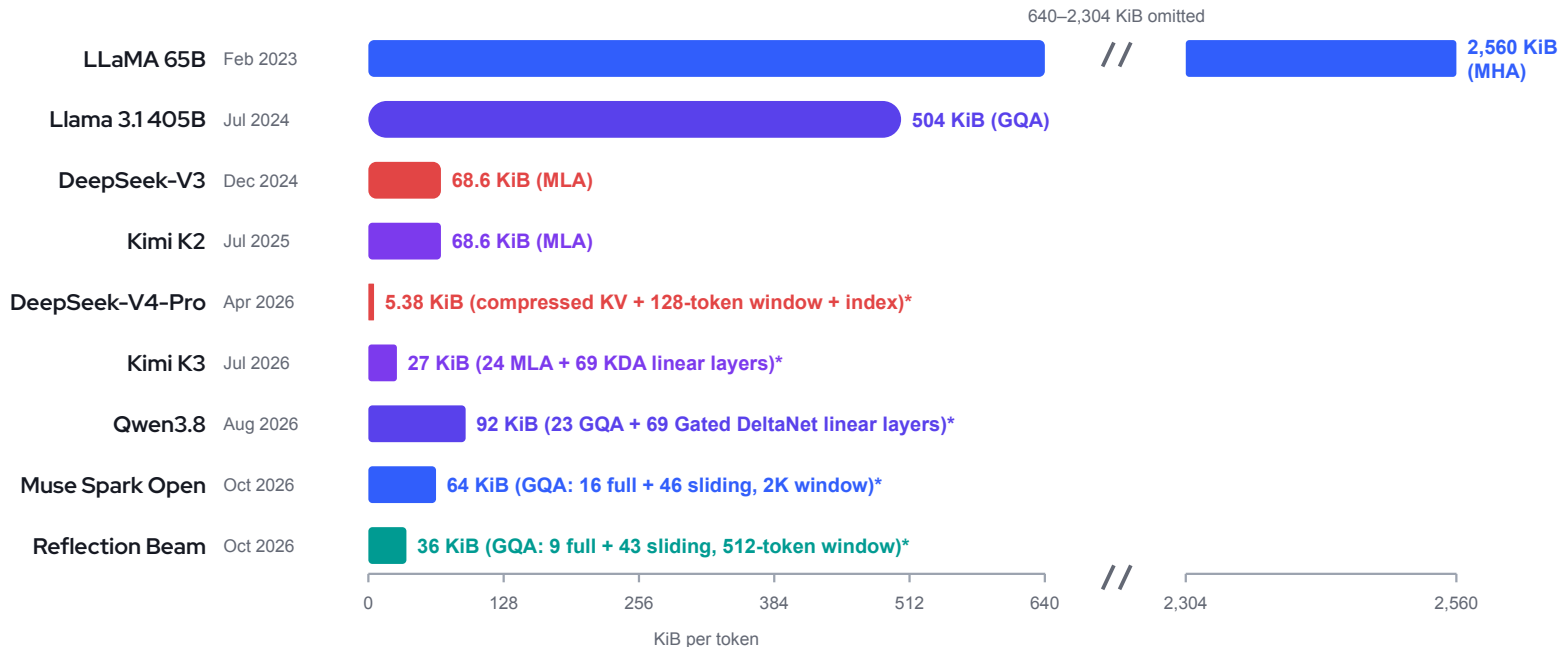


Efficient decode means batching many users, so the live state outgrows the model itself.



KV cache size and attention are heterogeneous

Growing KV payload across all layers, one cache copy. BF16 except DeepSeek-V4's packed FP8.



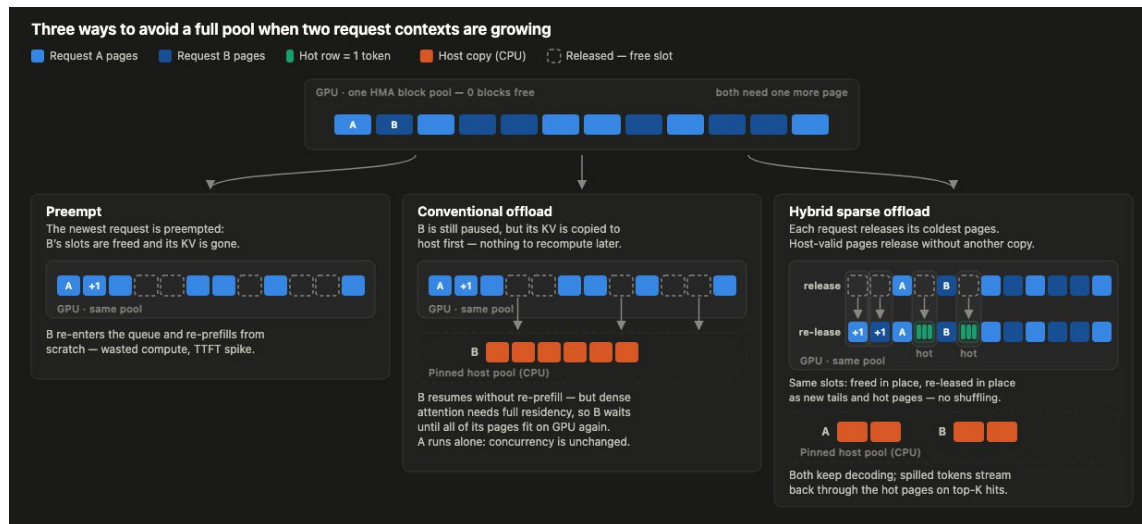
MLA and hybrid attention keep token state small as model weights grow.



Hybrid HiSparse

Sparse attention lets us maximize the KV cache space

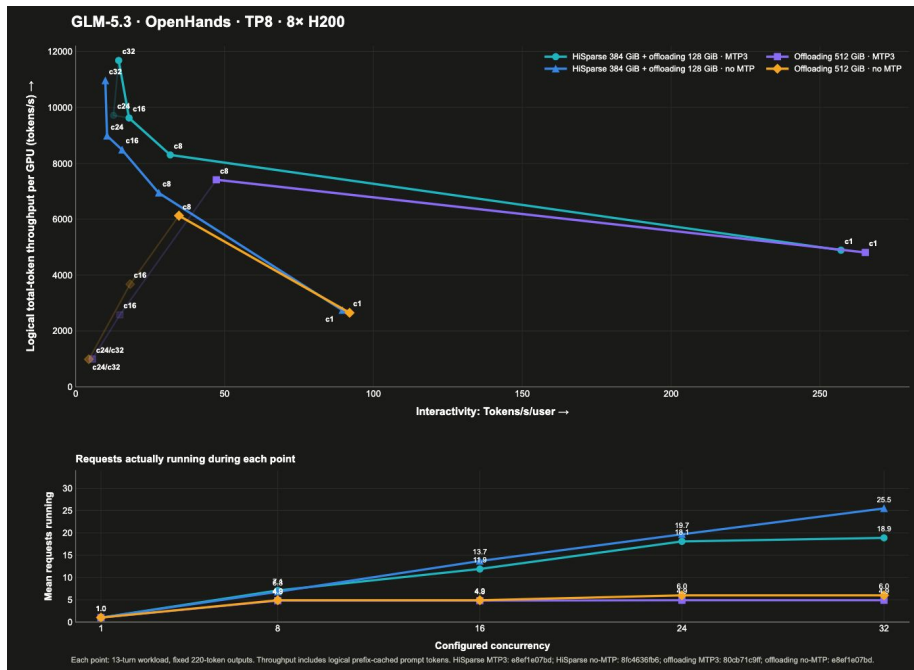
- Models like DeepSeekV3.2 and GLM 5.2 have sparse attention layers with indexers
- Idea: Keep KV cache in CPU memory, stream tokens selected by the indexer



Hybrid HiSparse

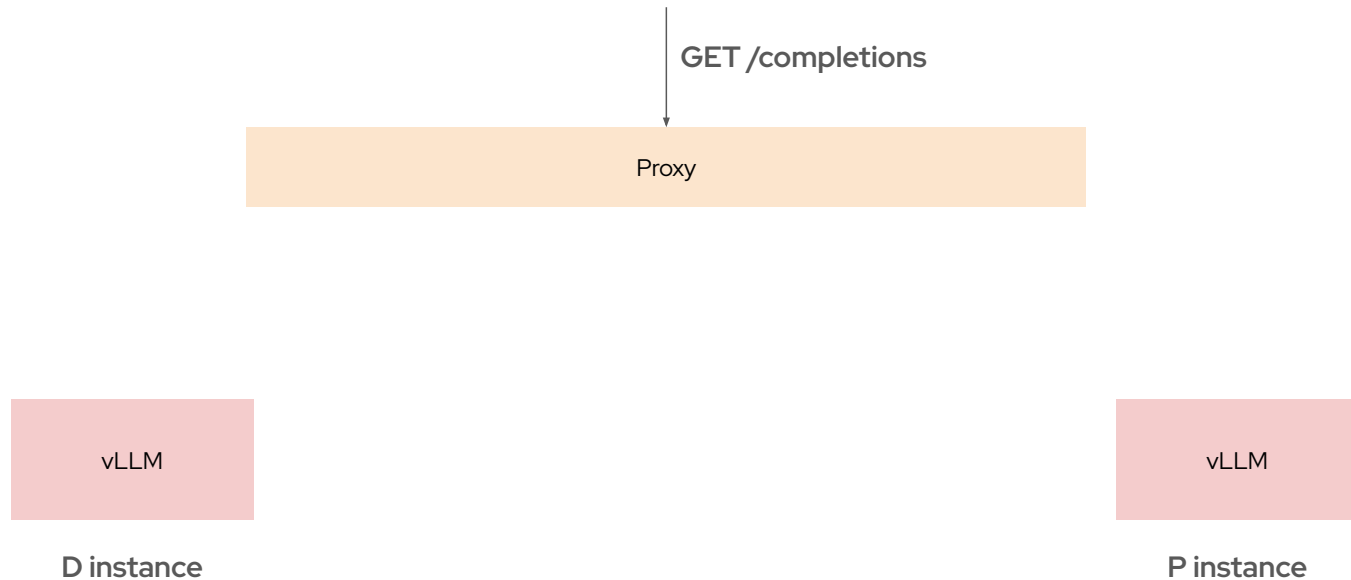
Sparse attention lets us maximize the KV cache space

- HiSparse keeps most of the KV cache in CPU memory.
- Increase the number of tokens a server can fit in KV cache
- For GLM 5.2, HiSparse plus DCP allows 1M context even on H100s (at EP32)
- **Downside:** This is a model-specific optimization



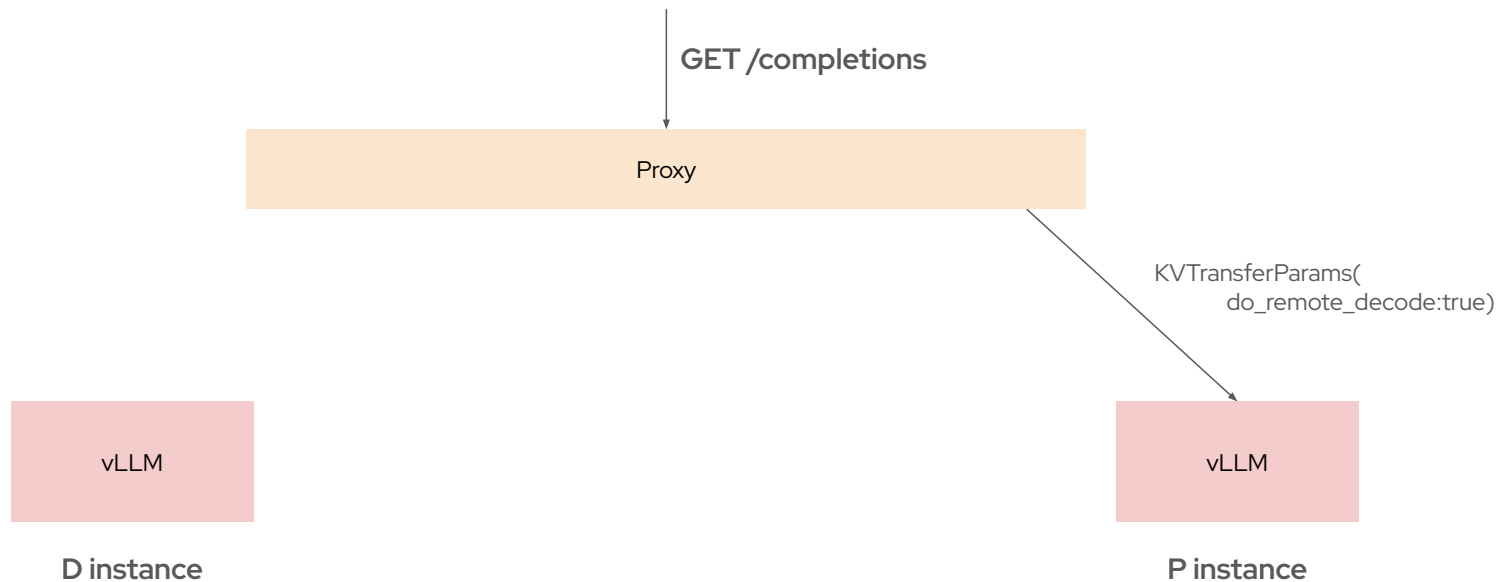
PD Disaggregation

- Use separate Prefill and Decode replicas to process a request



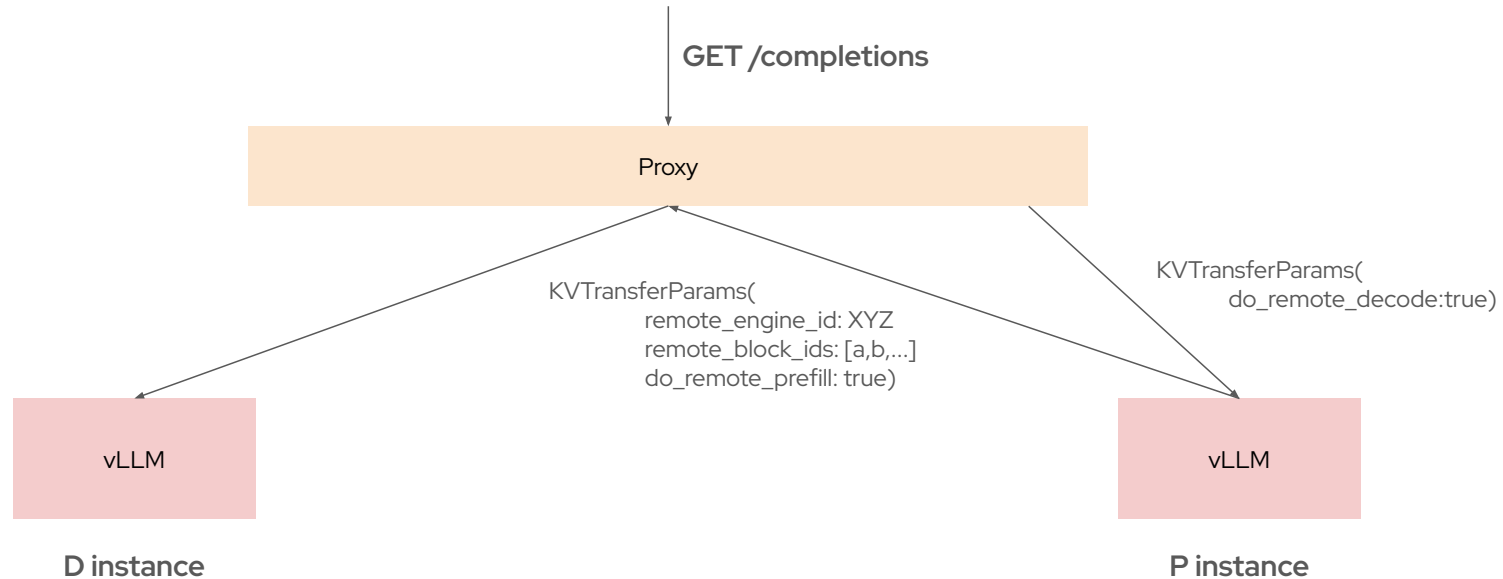
PD Disaggregation

- Use separate Prefill and Decode replicas to process a request



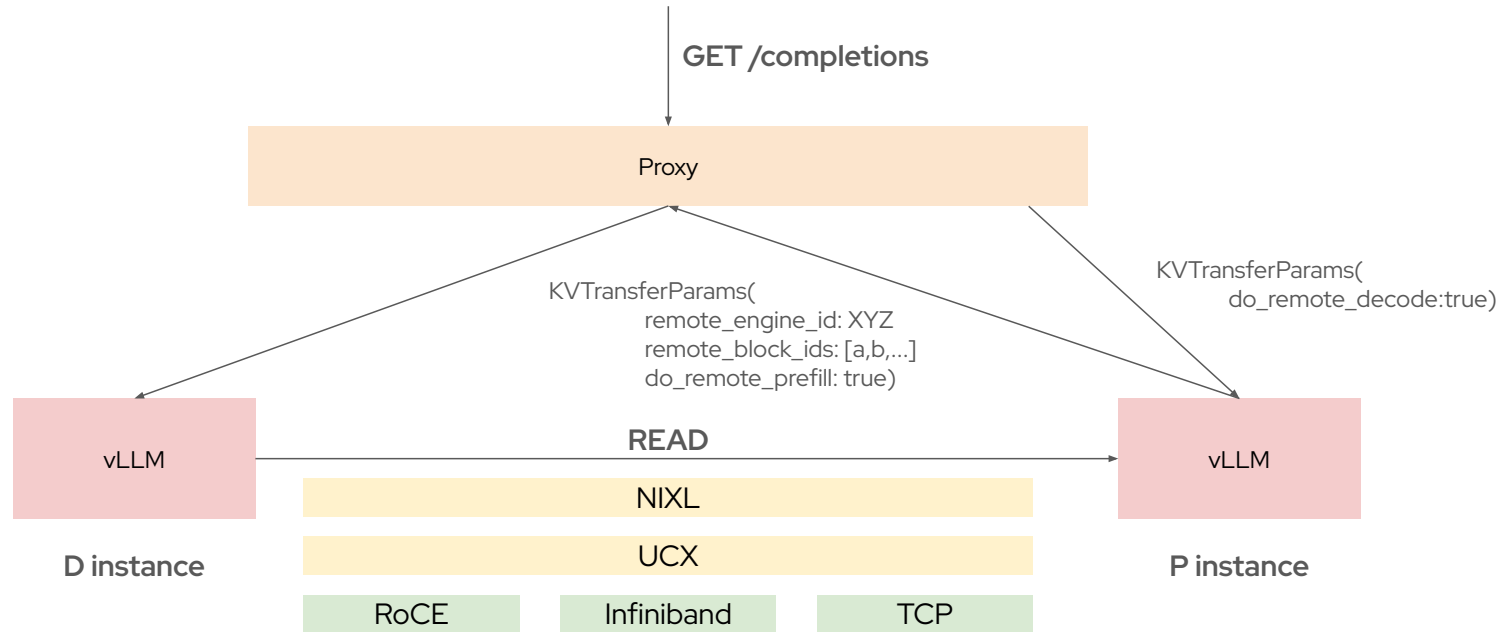
PD Disaggregation

- Use separate Prefill and Decode replicas to process a request



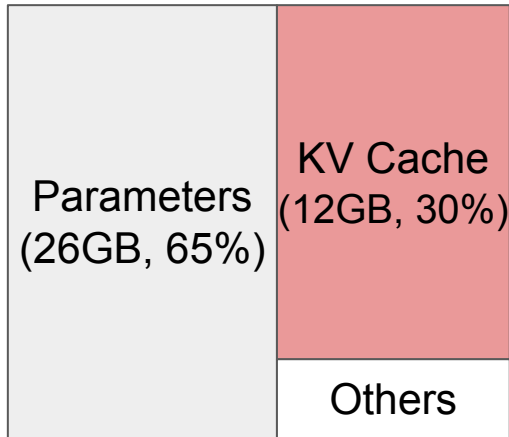
PD Disaggregation

- Use separate Prefill and Decode replicas to process a request



Key insight

Efficient management of KV cache is crucial for high-throughput LLM serving



13B LLM on A100-40GB*

*Common setup in 2023

