Introduction to vLLM: High-Performance LLM Serving & Maximizing Performance on Limited GPUs

vLLM Thailand Meetup - Sept 2026

Tun Jian Tan (@tjanaa)
vLLM Maintainer
Principal AI Engineer, Embedded LLM
Slides co-prepared by Kuan-fu Liu, Embedded LLM

embedded

embedded

VISION

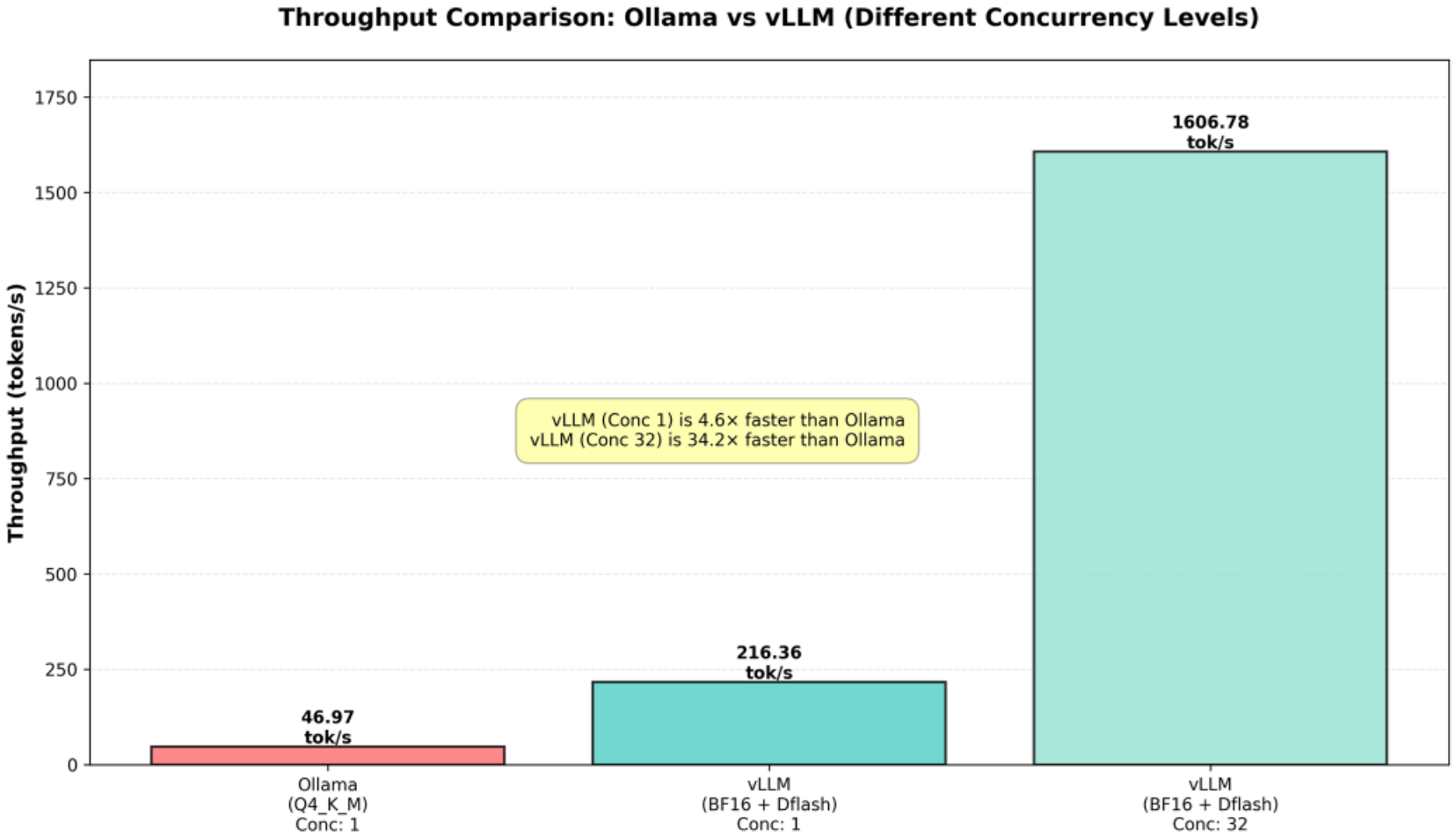
vLLM everywhere: the fastest engine on any hardware, powering an interoperable, federated AI utility.

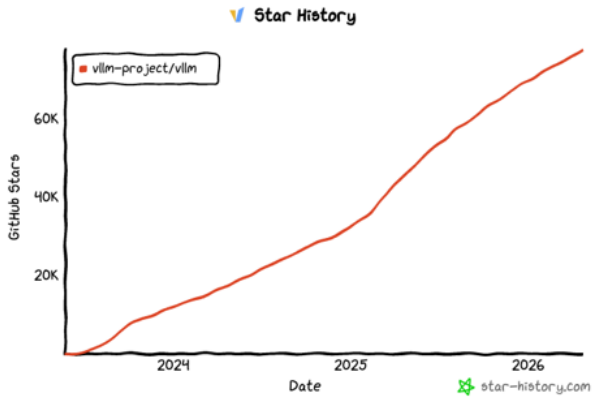
MISSION

A federated AI utility powered by vLLM: fastest on any hardware, interoperable by design, and aligned with the ecosystem so contributors and partners all benefit.

Different category of software

	Ollama	vLLM
Target	Single user, laptop	Concurrent, production
Batching	1 (batch size by default)	Continuous batching
Scale-out	TP (dependent on platform)	TP/PP/DP/EP, PD Disaggregated Inferencing
Day-0 support	Lagging	Same day
Quantization	GGUF	FP8, INT4, INT8, MXFP4, NVFP4


Throughput Comparison: Ollama vs vLLM (Different Concurrency Levels)
Throughput (tokens/s)
0
250
500
750
1000
1250
1500
1750
46.97
tok/s
216.36
tok/s
1606.78
tok/s
vLLM (Conc 1) is 4.6× faster than Ollama
vLLM (Conc 32) is 34.2× faster than Ollama
Ollama
(Q4_K_M)
Conc: 1
vLLM
(BF16 + Dflash)
Conc: 1
vLLM
(BF16 + Dflash)
Conc: 32



Most Popular LLM Serving Engine

- **91K+** GitHub stars, **800+** PRs/month
- **500K++** GPUs deployed 24/7
- **12.5K+** members in slack.vllm.ai

>2000 Contributors from >50 major companies

Broad Model Support (>100 arches)

Llama · DeepSeek · Qwen · GPT-OSS · Gemma · Mistral · Kimi · Minimax · Inkling

Flexible Device Parallelism

Tensor, Pipeline, Expert, Data, Context Parallel, Disagg Prefill/Decode, Disagg Encoder

Wide Hardware Support

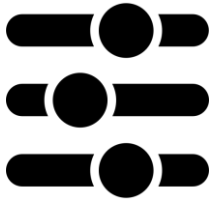
CUDA · ROCm · Gaudi/XPU · TPU · Neuron · Ascend · Metal · MACA · RBLN · Spyre · Monaka · MLU · Kunlun · CPU

Diverse Project Ecosystem

AGENTIC API · llm-d · vLLM-Omni · GuideLLM · vLLM Semantic Router · LLM Compressor · PECULATORS · Production Stack

Security
Your data never leaves your VPC. Sensitive prompts stay inside your trust boundary

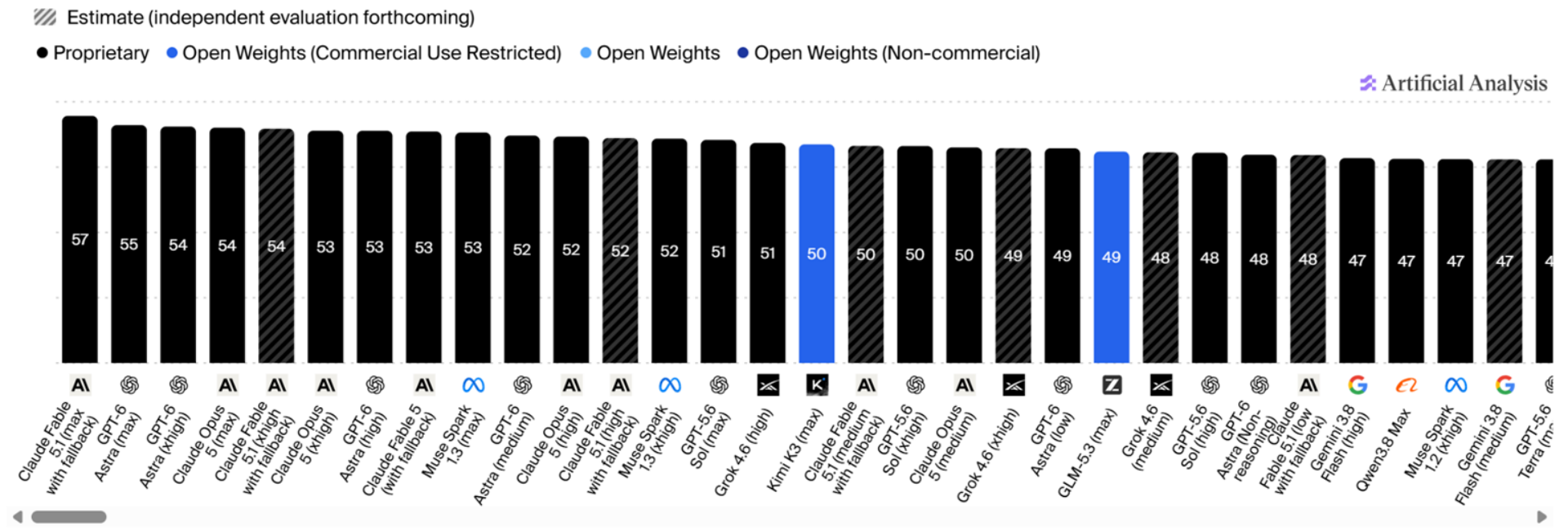
Privacy
No vendor telemetry on your prompts. No mystery retention policy. No black box.



Control
Your model. Your routing. Your SLAs. Your own pace on upgradings and policy.

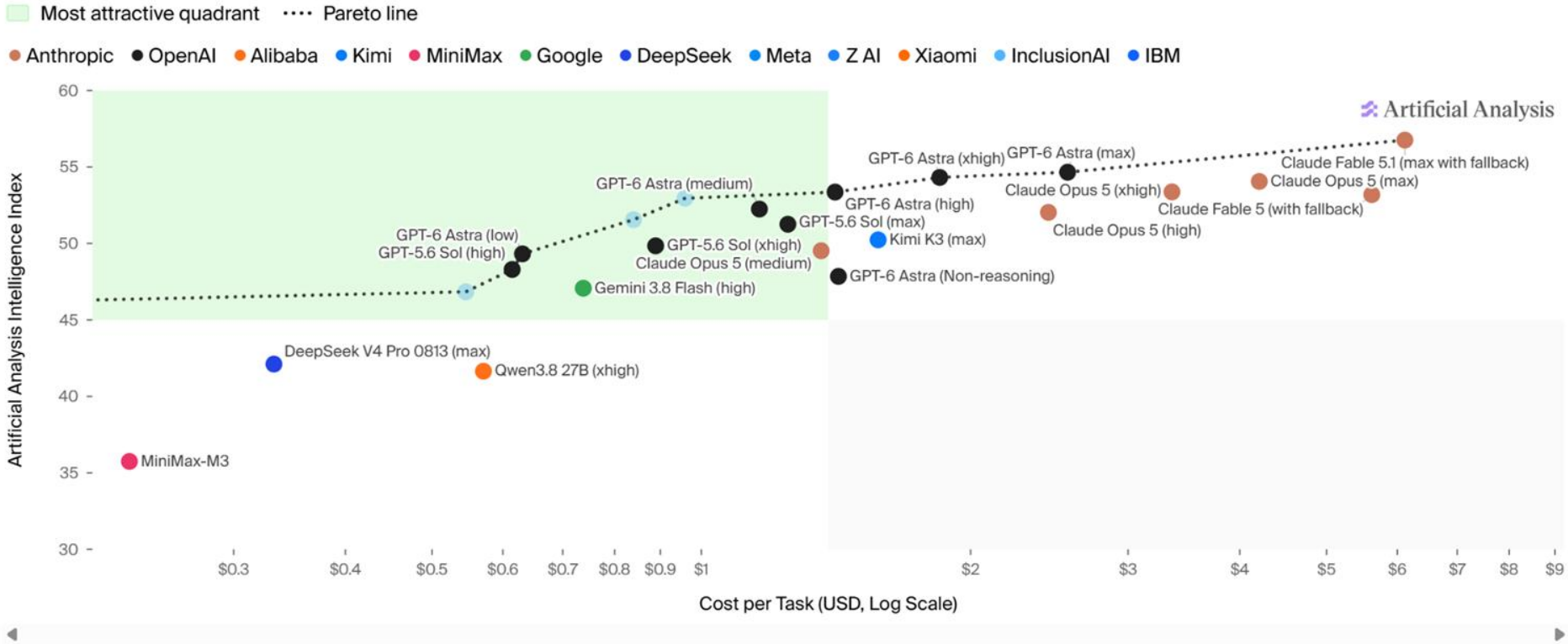
Artificial Analysis Intelligence Index by Open Weights / Proprietary

Artificial Analysis Intelligence Index v4.2 incorporates 10 evaluations: AA-Briefcase, GDPval-AA v2, τ^3-Banking, Terminal-Bench v2.1, SciCode, Humanity's Last Exam, GDP.pdf, CritPt, AA-Omniscience, AA-LCR v1.1



Intelligence Index vs. Cost per Intelligence Index Task

Artificial Analysis Intelligence Index · Weighted average cost (USD) per Artificial Analysis Intelligence Index task



Consumer GPUs vs Datacenter GPUs

CONSUMER GPUs

RTX 4090 (24 GB)

RTX 5090 (32 GB)

VRAM

RTX 4090 (24 GB)

24 GB
VRAM

DATACENTER GPUs

H100 (80GB)

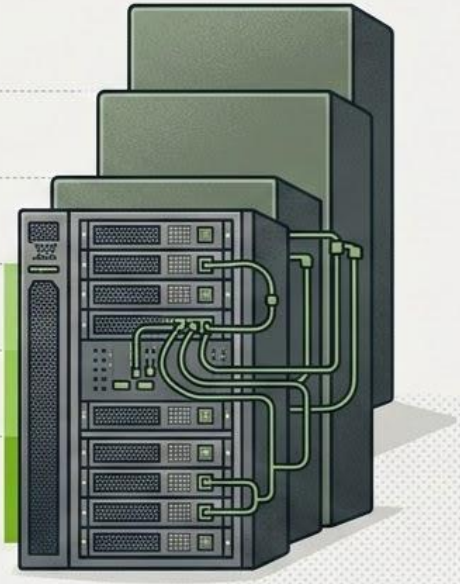
B300 (288GB)

B200 (192 GB)

H200 (141GB)

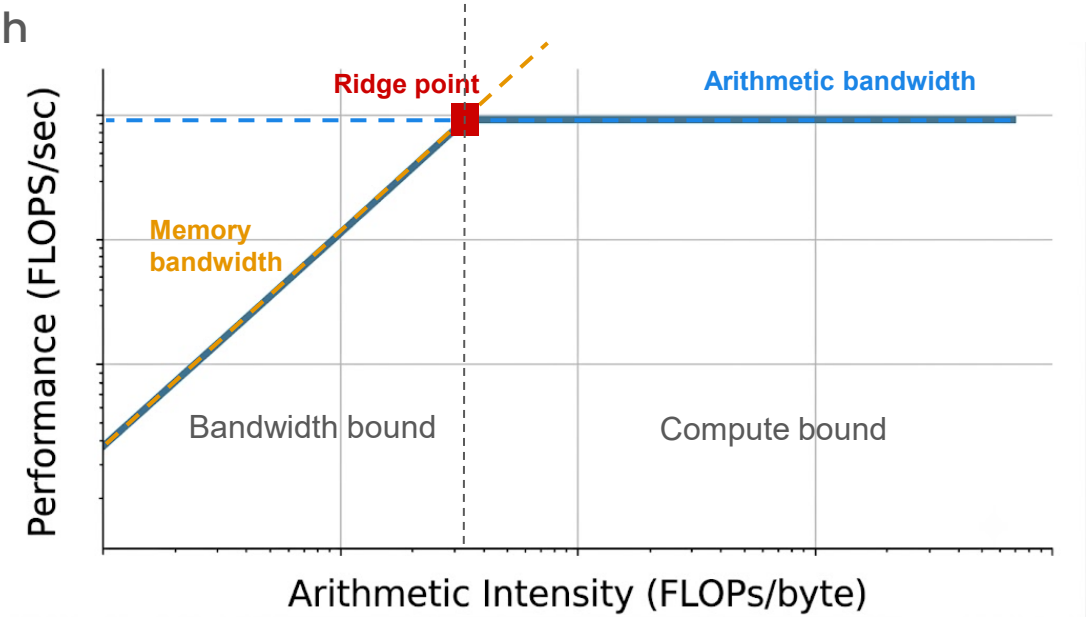
GB300 (288GB)

H100 (80GB)
VRAM



Hardware constraints in vLLM serving

- Compute power
- Memory bandwidth
- Memory space



What is consuming VRAM

- Model weights

  $$Memory_{weight} = N_{parameters} * Bytes_{weight}$$

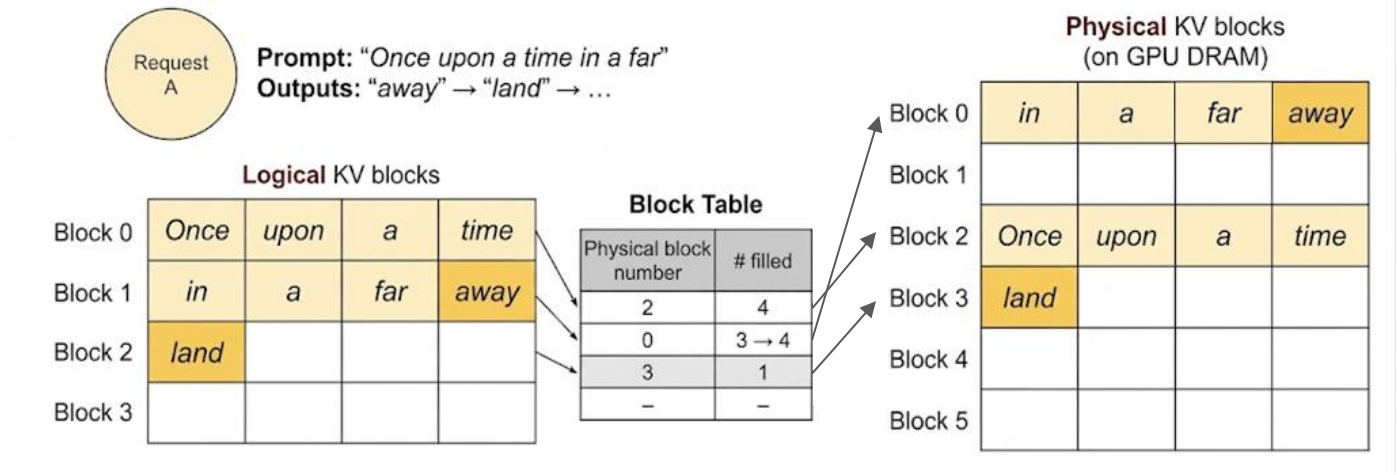
  - e.g. Llama-3.1-70B with BF16 weight: 141GB
- KV cache

  $$Bytes_{token} = 2 * L * H_{kv} * d_{head} * Bytes_{kv}$$

  - e.g. Llama-3.1-70B with BF16 KV cache: 2.56GB with 8k sequence
  - SSM or hybrid models
- Static memory
  - CUDA graph
  - Pytorch memory pools
  - Kernel workspace buffers

KV Cache

- Skip costly KV recalculation
- Paged attention divides sequences into blocks
- LRU eviction policy



How to make the most of your GPUs

- **Quantization**
  - Model weights
  - Activations
  - KV cache
- **Improve cache hits**
  - Attend to prefix caches
- **Workload optimization**
  - Balance compute and memory bandwidth

Model weights/activations quantization

$$Memory_{weight} = N_{parameters} * Bytes_{weight}$$

- BF16 -> FP8: saves half the space

- **vLLM supports quantized model weights**
  - GGUF, GPTQ, AWQ, llm-compressor, bitsandbytes, etc
- **Online quantization at model loading**
  - --quantization [fp8|...]
- **Faster hardware tensor/matrix arithmetics for quantized workloads**

Radeon™ RX 9070 XT

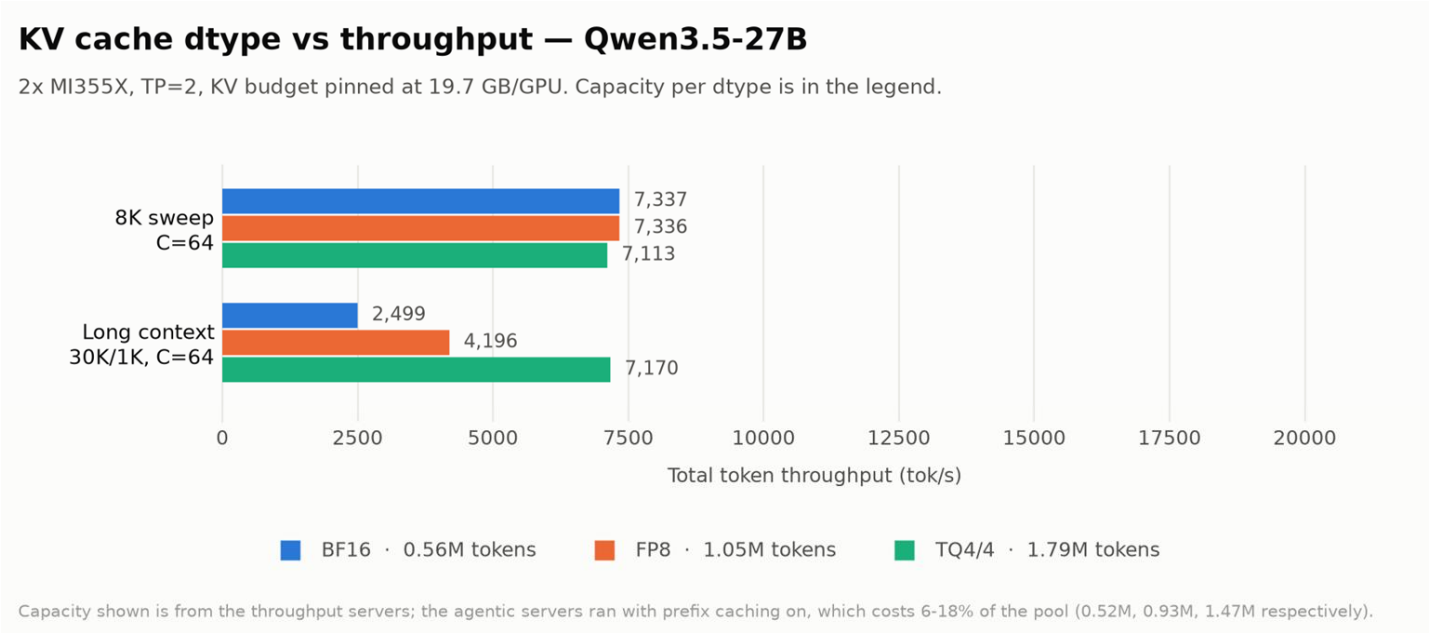
Peak Half Precision (FP16 Vector) Performance	48.7 TFLOPs
Peak Half Precision (FP16 Matrix) Performance	195 TFLOPs
Peak Half Precision (FP16 Matrix) Performance with Structured Sparsity	389 TFLOPs
Peak 8-bit Precision (FP8 Matrix) Performance (E5M2, E4M3)	389 TFLOPs
Peak 8-bit Precision (FP8 Matrix) Performance with Structured Sparsity (E5M2, E4M3)	779 TFLOPs
Peak 8-bit Precision (INT8 Matrix) Performance	389 TOPs
Peak 8-bit Precision (INT8 Matrix) Performance with Structured Sparsity	779 TOPs

KV cache quantization

$$Bytes_{token} = 2 * L * H_{kv} * d_{head} * Bytes_{kv}$$

- Store more tokens in memory
  - --kv-cache-dtype fp8
  - --calculate-kv-scales # Calibration
  - Also supports calibration with external datasets
- Less memory traffic for bandwidth bound workloads

KV cache quantization



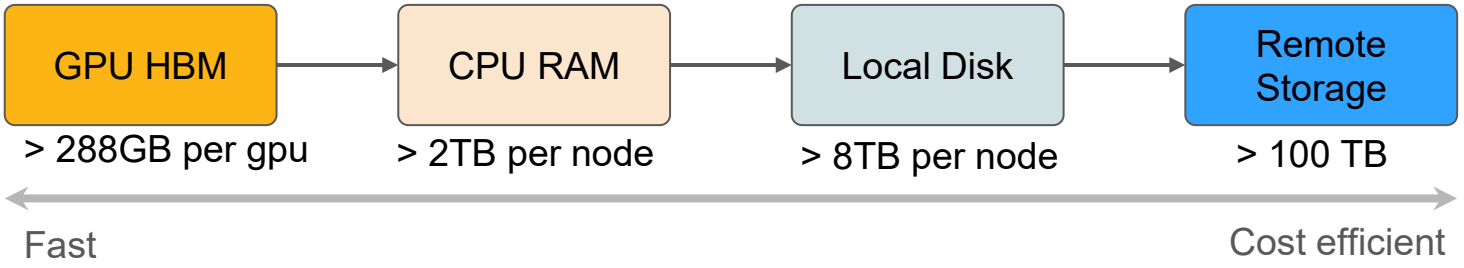
Automatic prefix caching

- Re-use KV cache across requests
  - --enable-prefix-caching

Request A	*You are a helpful assistant gives detailed and polite answers to the user's questions. User: Hello!*
Request B	*You are a helpful assistant gives detailed and polite answers to the user's questions. User: How's the weather?*

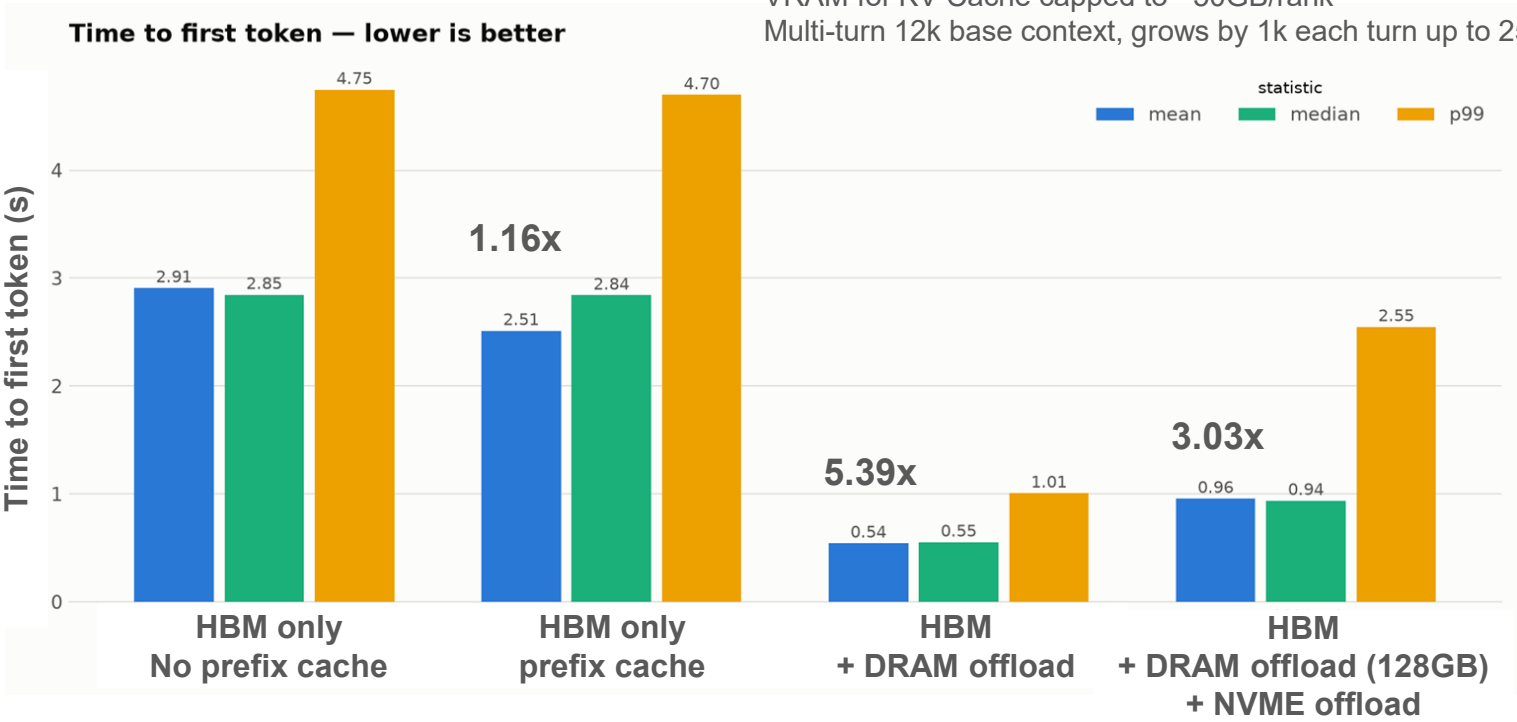
KV cache offloading

- In multi-turn agentic workloads, context can get very long and re-computing kv cache can be expensive
- In combination with **LMCache**, vLLM's KV cache can be offloaded to CPU DRAM, NVME, or remote storage

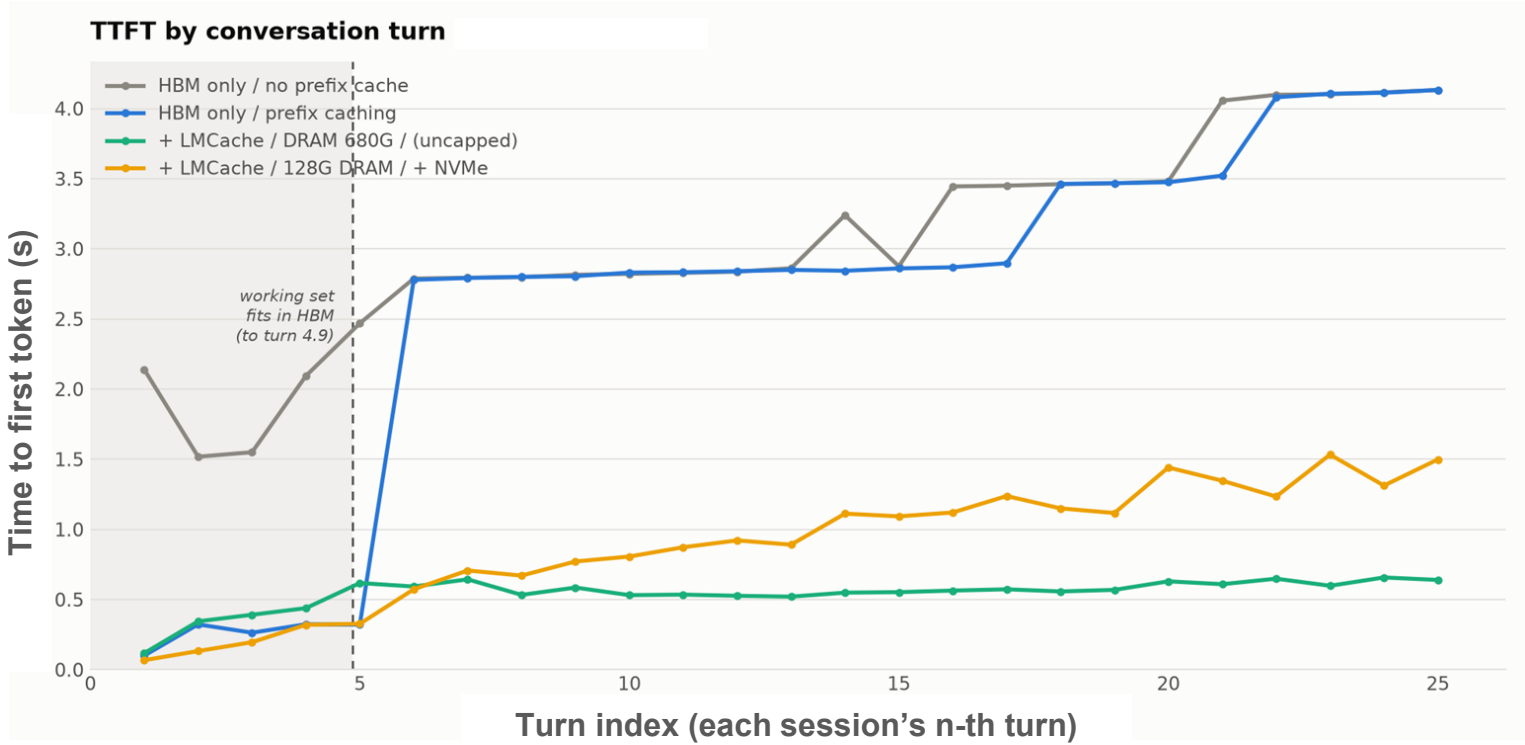


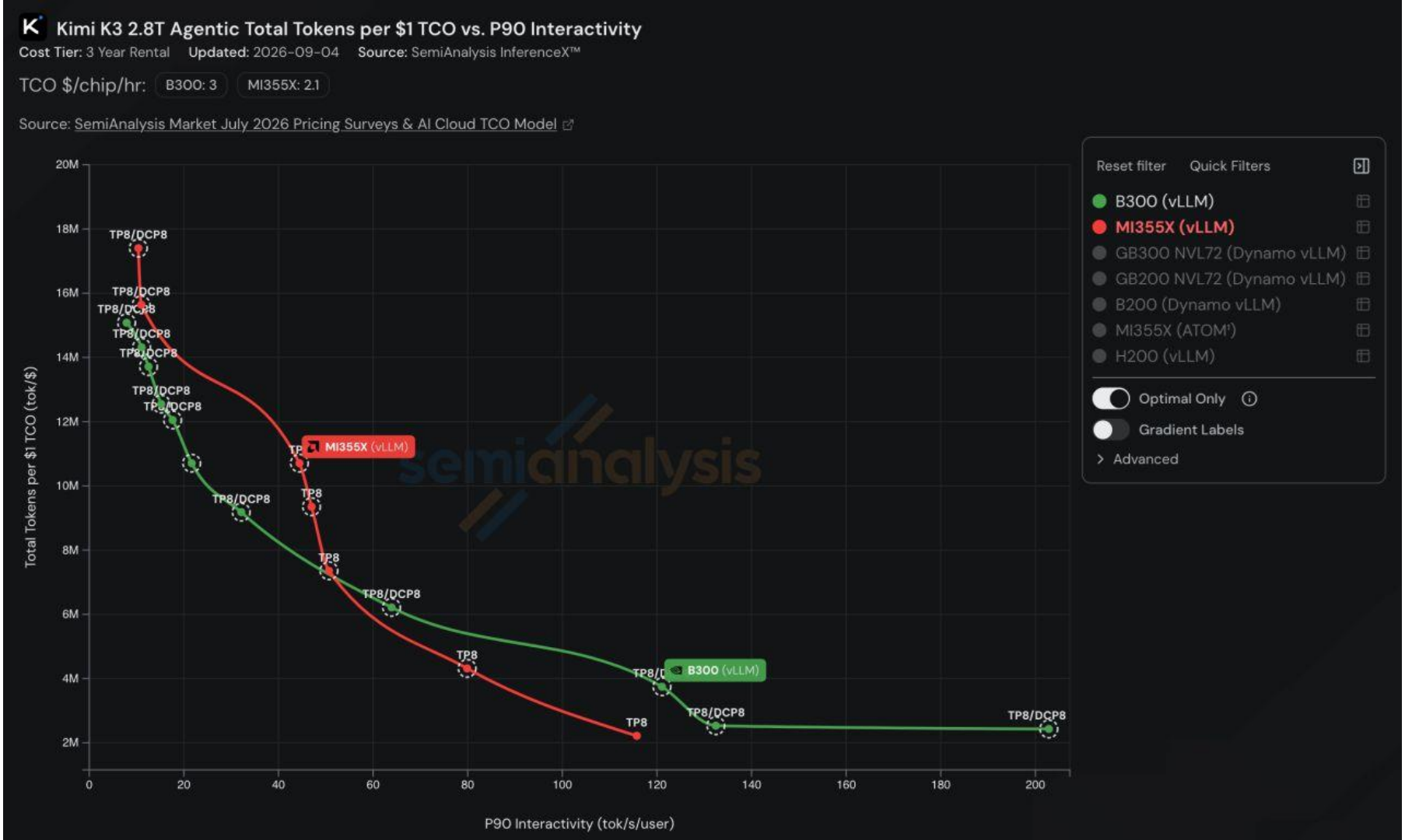
KV cache offloading

MiniMax-M3 TP4 on MI355X
VRAM for KV Cache capped to ~50GB/rank
Multi-turn 12k base context, grows by 1k each turn up to 25 turns, C=16



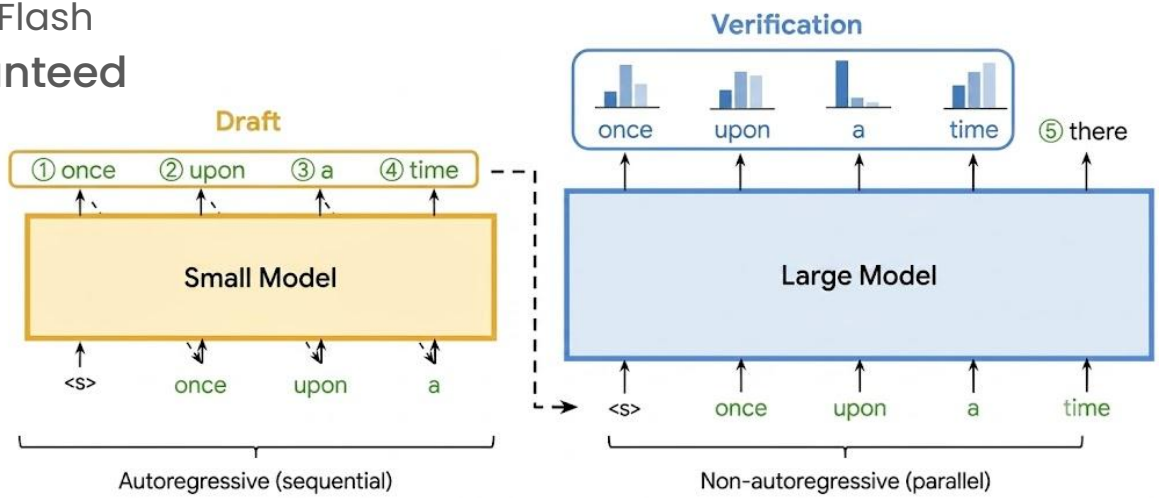
KV cache offloading




Kimi K3 2.8T Agentic Total Tokens per $1 TCO vs. P90 Interactivity
Cost Tier: 3 Year Rental Updated: 2026-09-04 Source: SemiAnalysis InferenceX™
TCO $/chip/hr: B300: 3 MI355X: 2.1
Source: SemiAnalysis Market July 2026 Pricing Surveys & AI Cloud TCO Model
Total Tokens per $1 TCO (tok/$)
P90 Interactivity (tok/s/user)
20M
18M
16M
14M
12M
10M
8M
6M
4M
2M
0
20
40
60
80
100
120
140
160
180
200
TP8/DCP8
TP8
MI355X (vLLM)
B300 (vLLM)
Reset filter
Quick Filters
B300 (vLLM)
MI355X (vLLM)
GB300 NVL72 (Dynamo vLLM)
GB200 NVL72 (Dynamo vLLM)
B200 (Dynamo vLLM)
MI355X (ATOM¹)
H200 (vLLM)
Optimal Only
Gradient Labels
Advanced
semianalysis

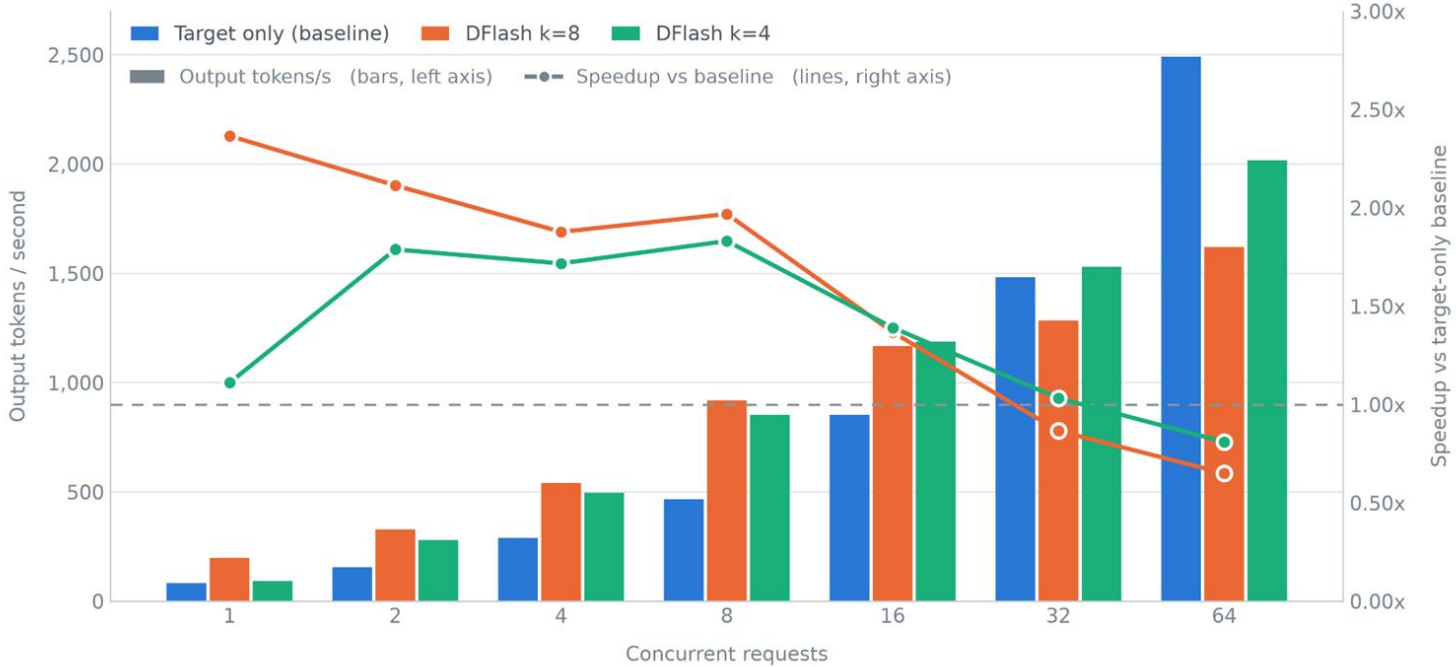
Speculative Decoding

- Reduce ITL in low-medium concurrency workloads
  - Pushing workload from memory bound regime more towards compute
- Separate draft model or weights
  - MTP, EAGLE, DFlash
- Accuracy guaranteed



Speculative Decoding

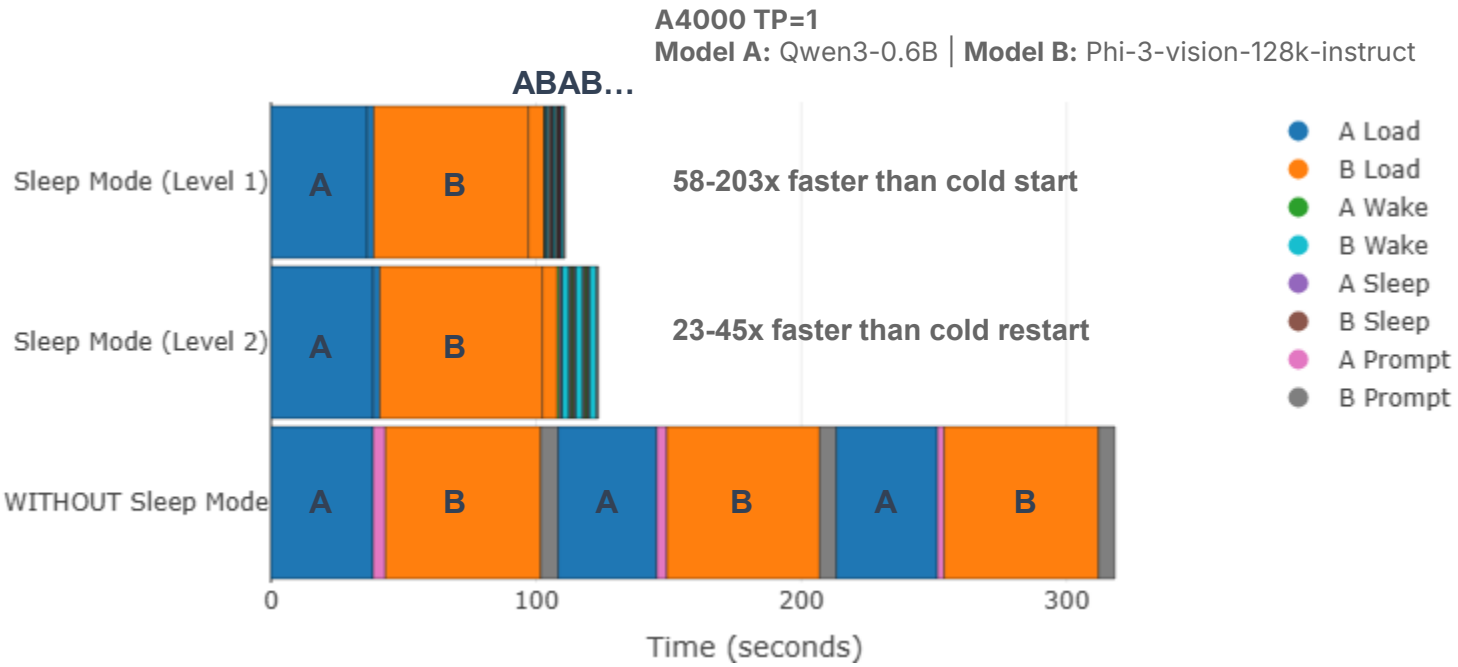
Qwen3.5-27B TP=2 on MI355
with z-lab/Qwen3.5-27B-DFlash



Sleep mode

- Reducing cold-start time for multi-model switching on the same hardware
- Level 1: Weight Offloading - Offloads weights to CPU RAM, discards KV cache
  - Memory allocator, process state, CUDA graphs, and compiled kernels retained
  - Provides the fastest wake times
  - Best for frequent model switching when CPU memory is sufficient
- Level 2: Memory Optimization - Discards both weights and KV cache from CPU RAM
  - Best for limited CPU RAM that can't hold all served models
  - Avoids reinitialization overhead, still much faster than a cold restart

Sleep mode


A4000 TP=1
Model A: Qwen3-0.6B | Model B: Phi-3-vision-128k-instruct
ABAB...
Sleep Mode (Level 1)
Sleep Mode (Level 2)
WITHOUT Sleep Mode
A
B
58-203x faster than cold start
23-45x faster than cold restart
A Load
B Load
A Wake
B Wake
A Sleep
B Sleep
A Prompt
B Prompt
0
100
200
300
Time (seconds)

They are not free gains

- **Quantization - potential accuracy drop**
  - Be sure to verify with evaluation tools (lm-eval, lmms-eval, etc)
- **Memory hardware**
  - Additional memory spaces and devices
- **Draft models and MTP layers**
  - Require additional VRAM
  - Pushing batching and speculative decoding together can over-saturate compute

What to keep in mind in setting up vLLM

Know your workload

Before allocating system resources, analyze the demands of your specific deployment environment.

- **Concurrency Requirements**

  Identify how many concurrent users or requests the system must serve.

- **Task Characteristics**

  Determine the types of tasks (e.g., short chat inputs, massive document generation, structured extraction).

Choose strategies based on workloads

Tailor your model configurations and software optimizations to match your analyzed query patterns.

- **Optimization Techniques**

  Select parameters such as chunked prefill, tensor parallelism, or speculative decoding based on traffic density.

- **Resource Allocation**

  Balance compute budget against memory limits to maximize throughput without exceeding hardware bounds.

Get Involved with vLLM

Building the fastest and easiest-to-use open-source LLM inference & serving engine together!

https://vllm.ai/
Explore vLLM and join Weekly SIG meetings

https://github.com/vllm-project/vllm
Star, file issues, send PRs

https://slack.vllm.ai
12.5K+ members and SIG channels

https://www.linkedin.com/company/vllm-project
Follow and connect to get latest news

https://twitter.com/vllm_project
Follow and connect to get latest news

https://roadmap.vllm.ai
Follow the Q2 board

If you are serving LLMs in 2026,
you're either using  or explaining why you're not.

Thank you
–embedded