AgentX - InferenceXv3: Does CUDA Moat Hold up in Agentic Inferencing?

AgentX - InferenceXv3: Does CUDA Moat Hold up in Agentic Inferencing? $3 Million USD dataset open sourced, 1 Mil+ Context Length, Multiturn, Sub Agents 95%+ KVCache HitRate, GB300 NVL72, MI355, B200

AUG 24 https://newsletter.semianalysis.com/p/agentx-inferencexv3-does-cuda-moat

View in browser

AgentX - InferenceXv3: Does CUDA Moat Hold up in Agentic Inferencing?

$3 Million USD dataset open sourced, 1 Mil+ Context Length, Multiturn, Sub Agents 95%+ KVCache HitRate, GB300 NVL72, MI355, B200

QQ:1413221435

CAM QUILICI, BRYAN SHAN, ALEC IBARRA, DANIEL NISHBALL, ZANE FONG, KIMBO CHEN, AND DYLAN PATEL

AUG 23 • PAID

READ IN APP

Since the Claude Code inflection point in November 2025, long-context, multi-turn agentic workloads have grown rapidly. They now dominate traffic for production inferencing. In April 2026, OpenAI's Enterprise agentic spending overtook ChatGPT spending.



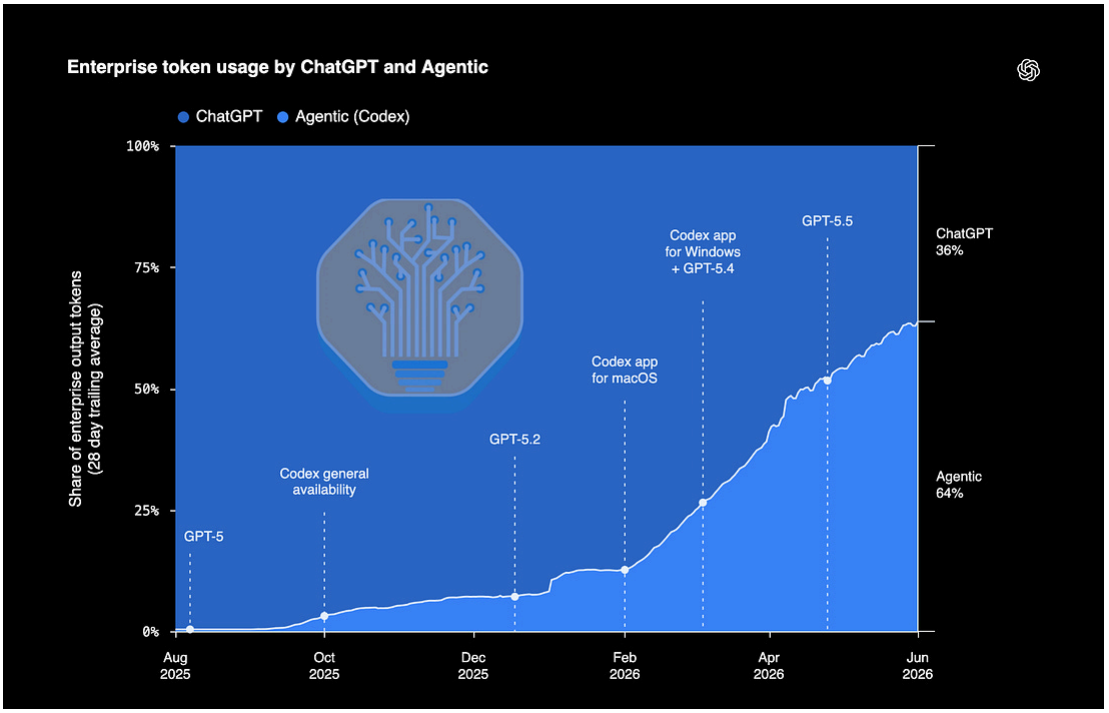
Claude Code is the Inflection Point

DOUG O'LAUGHLIN, JEREMIE ELIAHOU ONTIVEROS, AND 3 OTHERS · FEB 5

Read full story

Agentic workflows have decisively taken the baton. **Today, we announce AgentX 1.0 - the world's first fully open source, multi-turn agentic coding inference benchmark at 1 million context, released under Apache 2.0. Our full dashboard is available here.**

QQ:1413221435



Source: SemiAnalysis

QQ:1413221435

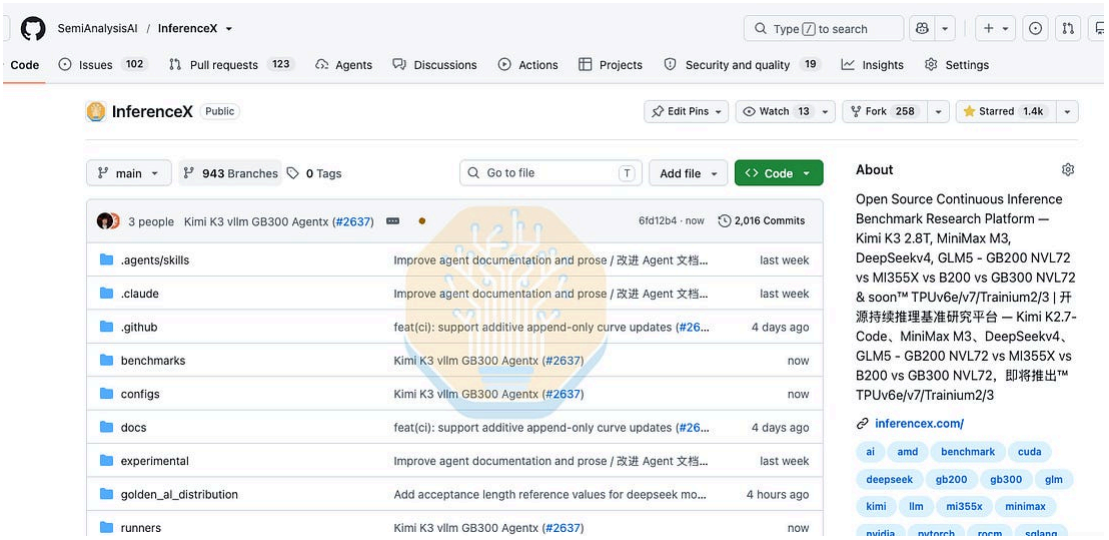
In the past most measured performance based on fixed sequence length prefill and decode workloads, but this is an inaccurate way to measure workloads. Reality is multi-turn, long context, high prefill reuse, with sub agent bursts, KVCache offload, and numerous tool calls. As such we aimed to build the correct way for the industry to measure AI hardware and software performance.

We have spent more than $3M building this dataset. Today, we open source everything. InferenceXv3 implements AgentX, a new realistic scenario in addition to the existing “fixed sequence length” scenarios (8k1k, 1k1k, 1k8k). It improves the benchmark scenarios by using agentic coding traffic instead of the previous single-turn traffic of 8k input and 1k output tokens.

The full matrix runs on ~2MW of continuously operated compute across over 1000 chips spanning a wide range of SKUs, featuring the MI355X, GB300 NVL72, GB200 NVL72, B300, B200, MI325, MI300X, H200, and RTX Pro Servers. Rubin arrives later this month, and TPUs and Mi455X UALoE72 arrive later this year. Please drop a star if you found our free open source work valuable.

It is great to see amazing performance from both NVIDIA and AMD on agentic workloads. NVIDIA does very good on a lot of frontier models while AMD also does well on some frontier models for specific comparsions.

QQ:1413221435



Source: [SemiAnalysis GitHub]

The most valuable thing AgentX produced in its first months was not the initial results. It was the massive industry impact the benchmark is already having. **Over 70+ upstream PRs** for optimizing real world production agentic workloads across vLLM, SGLang, TensorRT-LLM, ATOM, AITER, Dynamo, LMCache, and Mooncake, uses AgentX as the north star benchmark proxy. Most of these optimization improvements are transferable to production traffic. We deep dive into each of these optimizations later in the article.

QQ:1413221435
QQ:1413221435

50 merged PRs mention “agentx”, 1 May – 13 Aug 2026

dynamo 25, TensorRT-LLM 9, sglang 8, vllm 5, ATOM 3 — no mentions in aiter, LMCache

merged	*repo*	*PR*	*title*
7 May	vllm	#40549	[ROCm] Enable SimpleCPUOffloadConnector on ROCm
4 Jun	vllm	#43447	[Prefix Caching] DeepSeekv4 - Support selective prefix-cache retention for sliding-window KV ca…
9 Jun	dynamo	#10433	perf(frontend): optimize chat SSE streaming hot path
10 Jun	dynamo	#10521	perf(kv-router): bucket prune expiries
	dynamo	#10540	perf(kv-router): reduce lookup hot-path overhead
12 Jun	dynamo	#10645	perf(llm): incrementally track worker overload state
	dynamo	#10676	perf(llm): avoid repeated suffix cleanup on removes
18 Jun	vllm	#45444	[Mooncake] Skip KV lookup for non-reachable SWA blocks
24 Jun	vllm	#46188	[Mooncake] Optimize lookup pool key string construction
	vllm	#46412	[Mooncake] Only check and store new KV cache range
26 Jun	dynamo	#10437	perf(runtime): add request-plane msgpack payload codec
30 Jun	dynamo	#11095	perf(mocker): reduce KV block bookkeeping overhead
6 Jul	dynamo	#11104	perf: transcode Python request plane directly
8 Jul	sglang	#30255	Fix DSV4 prefill large Triton recompilation idle across context lengths
9 Jul	sglang	#28534	[AMD] Enable JIT staged HiCache write-back and fix CPU-index crash
10 Jul	dynamo	#11508	perf(kv-router): use arena block ownership counts
	dynamo	#11539	perf(runtime): avoid copying MessagePack event payloads
12 Jul	dynamo	#11569	perf(llm): reduce ITL metrics overhead
13 Jul	dynamo	#11574	perf(runtime): avoid copying received ZMQ frames
14 Jul	TensorRT-LLM	#16279	[None][fix] Seq-slot pool overlap headroom with consistent slot-indexed buffer sizing
15 Jul	dynamo	#11644	perf(kv-router): compress block tracker prompt paths
16 Jul	ATOM	#1586	Fix: support agentic dataset benchmark under PD disaggregation mode
	sglang	#31294	Skip no-op EAGLE sampling renormalization
	sglang	#31373	[Docs] Align B200 DeepSeek-V4-Pro balanced recipe with MegaMoE
17 Jul	dynamo	#11820	perf(logging): avoid dynamic filter span locks
19 Jul	dynamo	#11878	fix(runtime): resolve response stream bind hosts safely
22 Jul	ATOM	#1640	[Feature] DeepSeek-V4 paged-SWA sparse checkpoint prefix-cache retention
24 Jul	TensorRT-LLM	#16734	[None][perf] Avoid implicit device-scalar syncs in DeepSeek-V4 ctx sparse metadata
26 Jul	dynamo	#12158	feat(router): add active-request decode cost [DYN-3849]
27 Jul	dynamo	#12161	perf(tokens): remove redundant inner DashMap sharding
28 Jul	dynamo	#12240	perf(mocker): avoid cloning offline worker groups
	dynamo	#12242	perf(mocker): drive only ready offline worker groups
	dynamo	#12244	perf(mocker): use intrusive inactive block LRU
29 Jul	dynamo	#12248	perf(mocker): flatten native G1 sequences
	dynamo	#12386	perf(mocker): batch attention-DP completions
	sglang	#29735	[GDN] Support FlashInfer GDN prefill with extra-buffer radix cache
30 Jul	dynamo	#12329	perf(mocker): move request KV ownership into backend leases
	dynamo	#12341	perf(mocker): optimize offline replay hot paths
31 Jul	TensorRT-LLM	#17105	[None][fix] Skip corrupt Bfloat16 MxE2m1MxE4m3 splitK2 MOE kernels
5 Aug	TensorRT-LLM	#17285	[None][fix] Honor MSA block-index strides for q_len <= 32
	ATOM	#1740	[feat][ci] support swebench_lite precision validation
6 Aug	sglang	#30545	[Disagg][StagingBuffer][2/2] Support radix cache
7 Aug	TensorRT-LLM	#17316	[None][perf] Add CuteDSL to MiniMax-M3 autotune (for MXFP8 linear GEMM+quant)
	TensorRT-LLM	#17428	[None][fix] Backport nonblocking CTX transfer polling to M3
10 Aug	TensorRT-LLM	#17473	[None][perf] Capture MiniMax-M3 context graph producers
11 Aug	sglang	#33662	[DSV4] Avoid host syncs in EAGLE prefill
13 Aug	TensorRT-LLM	#17518	[None][perf] Optimize head-mismatched MiniMax-M3 KV transfer with corrected cache mapping and N…
	TensorRT-LLM	#17520	[None][test] Add MiniMax-M3 disaggregated accuracy coverage
	dynamo	#12999	perf(frontend): flush detokenization metrics once
	sglang	#30762	fix(hicache/umbp): support DeepSeek-V4 hybrid HostPoolGroup (multi-po…

QQ:1413221435

Source: SemiAnalysis

Open source is a core principle for InferenceX and thus, we open more of the stack than most people who use that word. That includes an open frontend, a public database served through an easily consumable REST API **that multiple tier 1 AI lab’s capacity planning teams already consume**, public GitHub Actions CI provenance, logs, and accuracy validation on every single point. Crucially, our benchmark configs mainly track recipes.vllm.ai and SGLang cookbook on upstream images such that we are measuring the performance actual customers are experiencing instead of measuring benchmax’ed images.

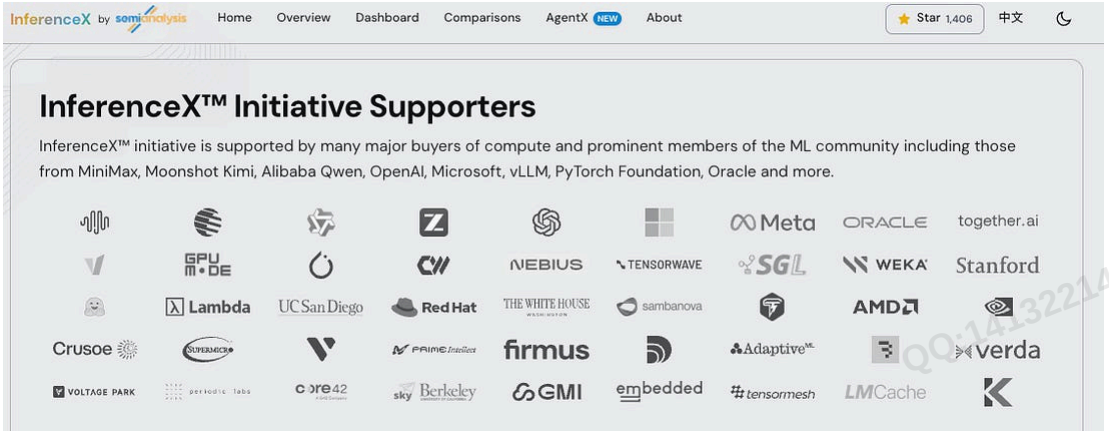
In three to four weeks, we will release an AgentX update article. It will cover further optimizations to agentic workloads, plus updated performance results from AMD and Nvidia. It is important to understand that the profile of agentic workloads is updating fast. InferenceX will continue to move swiftly to benchmark the relevant workloads.

QQ:1413221435

InferenceX is 100% committed to being open-source - this would not be possible without the contributions and support from our OSS partners. We would like to thank the following people that have made massive contributions to the AgentX 1.0 release:

- **Inferact/vLLM**: Roger Wang, Yifan Qiao, Simon Mo, Jeff Ma, and many others
- **RedHat/llm-d**: Michael Goin, Robert Shaw, Tyler Michael Smith
- **RadixArk/SGLang**: Baizhou Zhang, Yuwei An, Mingyi Lu, and many others
- **LMCache/TensorMesh**: Samuel Shen
- **Weka**: Callan Fox, ValB
- **MoonCake Maintainers:** Teng Ma, Xu Wenjie, Ke Yang
- **AMD**: Thomas Wang, HaiShaw, Andy Luo, Seungrok Jung, Chun Fang, Parth Panchal, Bill He, Theresa Shan, Hongxia, Fangzhou, Gilbert Lei, Yanfei Wang, Duyi Wang, Peng Sun, Lingpeng Jin, Simon Danielsson, Xiaohu Guo, Haichen Zhang, Chang Liu, Doug Lehr, Poovaiah Palangappa, and many others in the AMD Shanghai Development Centre
- **Nvidia**: Xin Li, Anthony Casagrande, Kedar Potdar, Ankur Singh, Ishani Dhanani, Nick Comly, Nvidia Shanghai TensorRT-LLM team, and many others
- **Anthropic staff, for promptly fixing multiple bugs that made implementing AgentX possible**
- **GitHub:** Austen Stone for helping with reliability of GitHub Actions that AgentX uses
- And many others

In addition, we are thankful to all who support our open source InferenceX initiative, including Meta, Microsoft, Oracle, OpenAI, MiniMax, Moonshot Kimi, Alibaba Qwen, and Zhipu GLM.



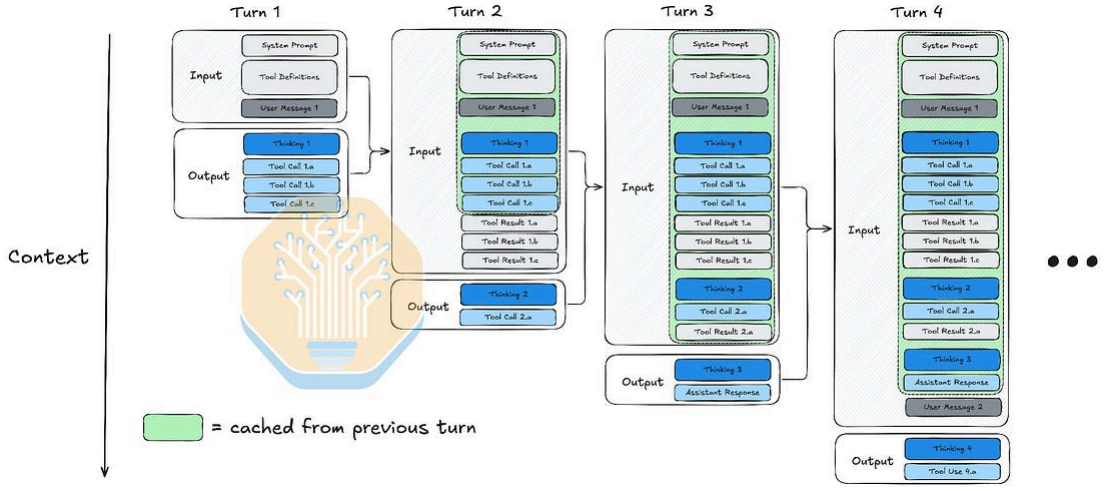
Source: InferenceX

QQ:1413221435

A Brief Overview of Agentic Workloads

At a high level, an agentic workload is characterized by four elements:

1. **Multi-turn:** a session includes many user / assistant interactions (tens or hundreds) compared to a handful in a chatbot scenario. Multi-turn, long context, high prefill reuse, with sub agent bursts and numerous tool calls.
2. **Long context:** system prompts, tool definitions, and the large number of turns make context accumulate quickly.
3. **High prefix reuse:** since the conversation progresses linearly, where output from turn n-1 is concatenated to turn n (typically), most context can be served from KV cache rather than recomputed (this depends on amount of storage available to store KV tensors). As n grows, the ratio of cached input relative to uncached typically tends towards 1.
4. **Sub-agent bursts:** a session launches multiple short-lived sub-agents with fresh context, which create bursty KVCache patterns.



Source: DeepSeek, SemiAnalysis

Considering the characteristics above, benchmarking these workloads is fundamentally different from the existing fixed sequence length benchmarks. Namely, agentic inference is inherently a *systems problem*. Because of extremely high prefix reuse, KV tensors must be efficiently transferred across nodes/ranks (NIXL, MORI-IO, Mooncake). Additionally, different conversations should be routed to different nodes/ranks depending on where the appropriate prefix resides in order to maximize cache hit rate (LLM-d, Dynamo, vLLM/SGLang router). Long context conversations stress the HBM capacity for KV cache and necessitate offloading KV tensors to different tiers of memory (DRAM, SSD), a process that needs to be carried

QQ:1413221435

out efficiently (Mooncake Store, LMCache, vLLM Simple Offloading, SGLang HiCache).

This is in contrast to fixed sequence length, single turn workloads where prefix reuse is not relevant and inference performance is largely reflective of baseline chip/kernel performance. This is not to say that the plethora of fixed sequence length data on InferenceX is not important. In fact, stripping away the complexities of agentic serving shows clearly how low-level inference performance optimizations are progressing. It also provides an important baseline for AgentX results.

In an attempt to make the AgentX workloads as realistic as possible, we collected an initial corpus of 393 internal SemiAnalysis anonymous Claude Code traces to replay. To anonymize the content while keeping the original prefix reuse pattern, we use a method similar to the Qwen-Bailian dataset, one of the earliest corpora of production traces. We then use AIPerf to reconstruct the traces according to the original schedule of requests at varying levels of concurrent clients. We worked with Anthropic to ship two Claude Code features to make the AgentX dataset possible. We thank the Anthropic staff for their help.

QQ:1413221435

1. https://github.com/anthropics/claude-code/issues/49207
2. https://github.com/anthropics/claude-code/issues/66761

This brief introduction to agentic workloads should give the reader enough context to understand the results in the next section. In a later section, we will provide a deeper technical dive into the methodology, replay harness, and dataset.

Agentic Coding Inference Performance

When looking at inference coding performance, OpenAI, Anthropic, xAI, and other frontier labs focus on three things. They look at performance per dollar versus interactivity (TPOT), TTFT (time to first token), and overall end-to-end task completion. Performance per megawatt is also important, considering that terrestrial datacenter power is a critical constraint (money is a social construct and it appears the labs have an unlimited supply, but power is physically hard to come by in this day and age). Our datacenter model has estimates of quarter by quarter build up of power demand and supply.

QQ:1413221435

In this section, we highlight some of the overall agentic performance themes across frontier models. We strongly encourage the reader to use this as a guide to investigate

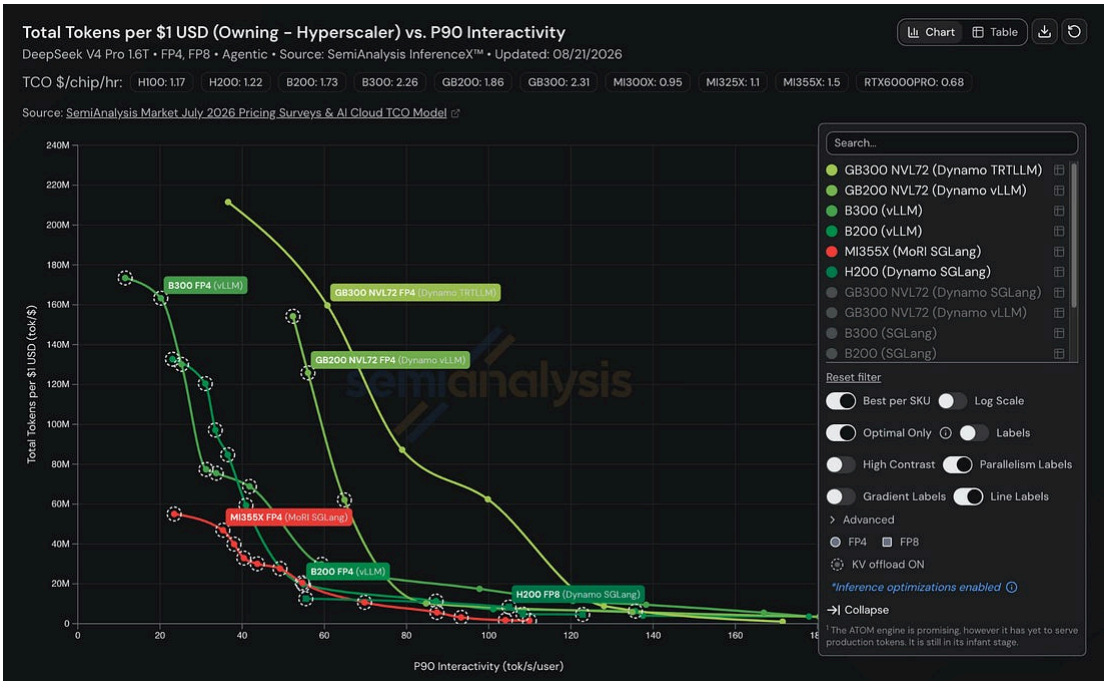
the results for themselves. **All of the data is open source and the community has the opportunity to draw their own conclusions on the current state of real world inference performance.**

DeepSeek V4 Pro 0813

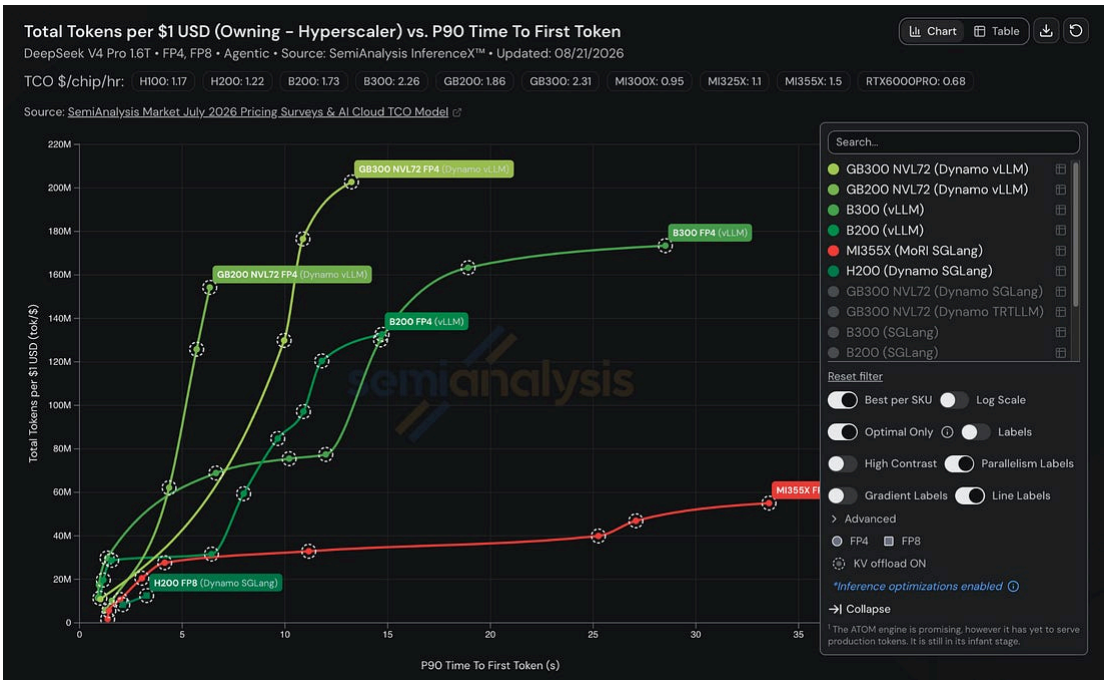
DeepSeek V4 Pro 0813 is an ultra popular frontier open weight model from China. It has ~1.6 Trillion parameters with 49 Billion active parameters.

As of August 21, the following graph shows the best performance per SKU for all submissions, normalized by total cost of ownership (TCO).

The ISL/OSL distribution of *all* requests among all DeepSeek v4 runs was as follows: ISL p50=88k, p90=272k, p95=404k, p99=675k and OSL p50=413, p90=2.2k, p95=3.7k, p99=8.6k.



QQ:1413221435
QQ:1413221435

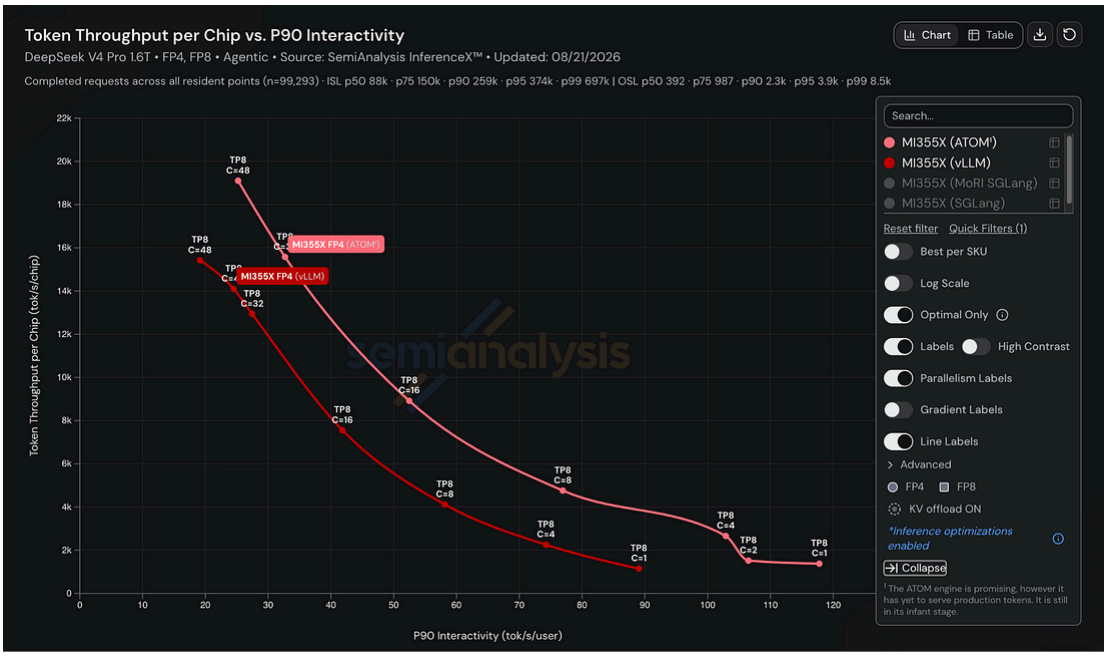


Source: [InferenceX]

In general, **it is important to consider both tokens per second per user** (TPS - also known as interactivity) and TTFT, **since these often come at the expense of one another**. For instance, in the graph above, some SKUs achieve very high throughput at decent interactivity, however TTFT is severely degraded. What is an “acceptable” p90 TTFT varies heavily depending on the application. For most production systems serving agentic workloads, you can expect p90 TTFT to be anywhere from 200-5,000ms. Anything over 5-10s is pushing the boundary of what can be considered “online inference.” There are still practical applications for the ultra-high throughput sector of the curve, where latency does not matter and peak system utilization is desirable (batch processing, *very* long running agents, etc).

In terms of single node performance, MI355X open-source performance (vLLM) trails behind vendor specific ATOM (AMDs equivalent of TensorRT LLM). We think it’s great that AMD is pushing the frontier quickly with ATOM, however we encourage them to make a higher priority of upstreaming these improvements into vLLM.

QQ:1413221435
QQ:1413221435



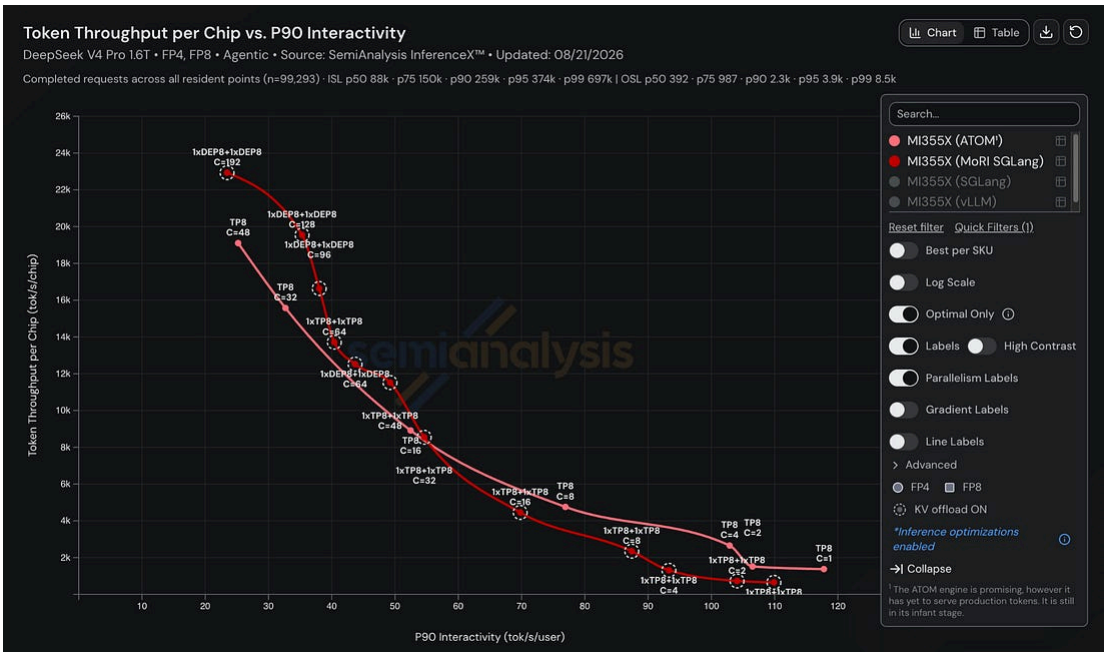
Source: InferenceX

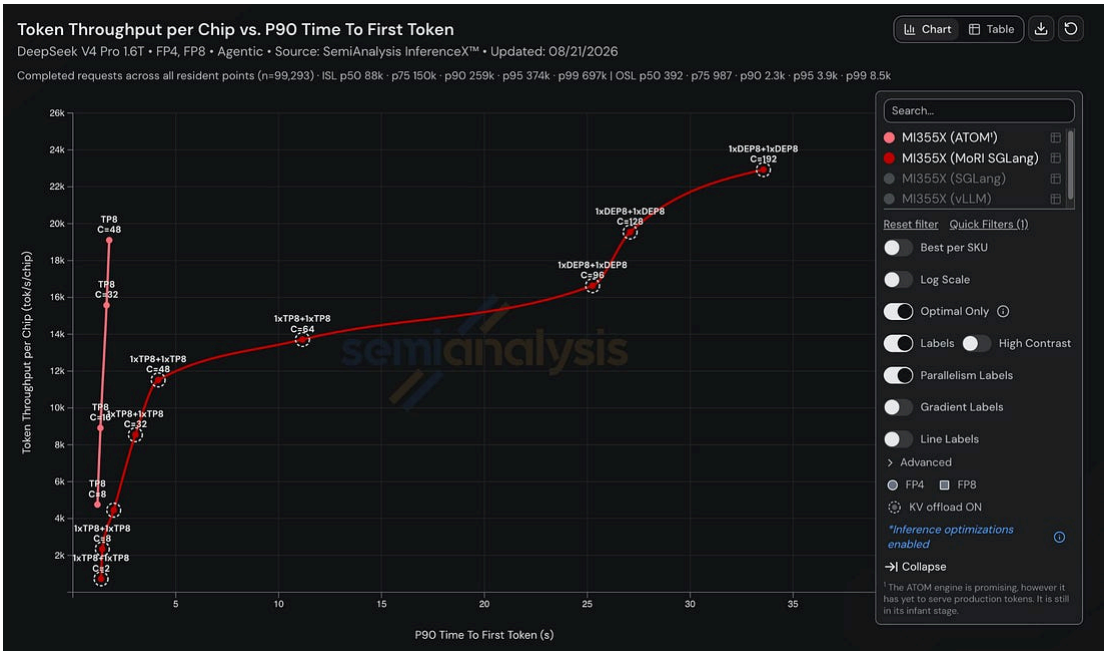
QQ:1413221435

AMD’s distributed inference (DI) team has made great progress on 8k1k scenarios over the past six months. The team still has some way to go before DI is a viable solution for realistic workloads. In terms of throughput per GPU vs. interactivity, we observe the 1xDEP8+1xDEP8 disagg config is only able to realize slight performance gains in the high throughput scenarios, while actually performing worse in low latency scenarios.

To make matters worse, any increase in throughput at the middle-to-high interactivity configs is overshadowed by the significant spike in p90 TTFT. One of the reasons for this is the use of SGLang’s --enable-prefill-delayer argument above concurrency 64, which postpones prefill admission so DP ranks can form fuller batches (for up to 30 forward passes). Additionally, these points also increase chunked prefill size from 8,192 to 65,536.

QQ:1413221435

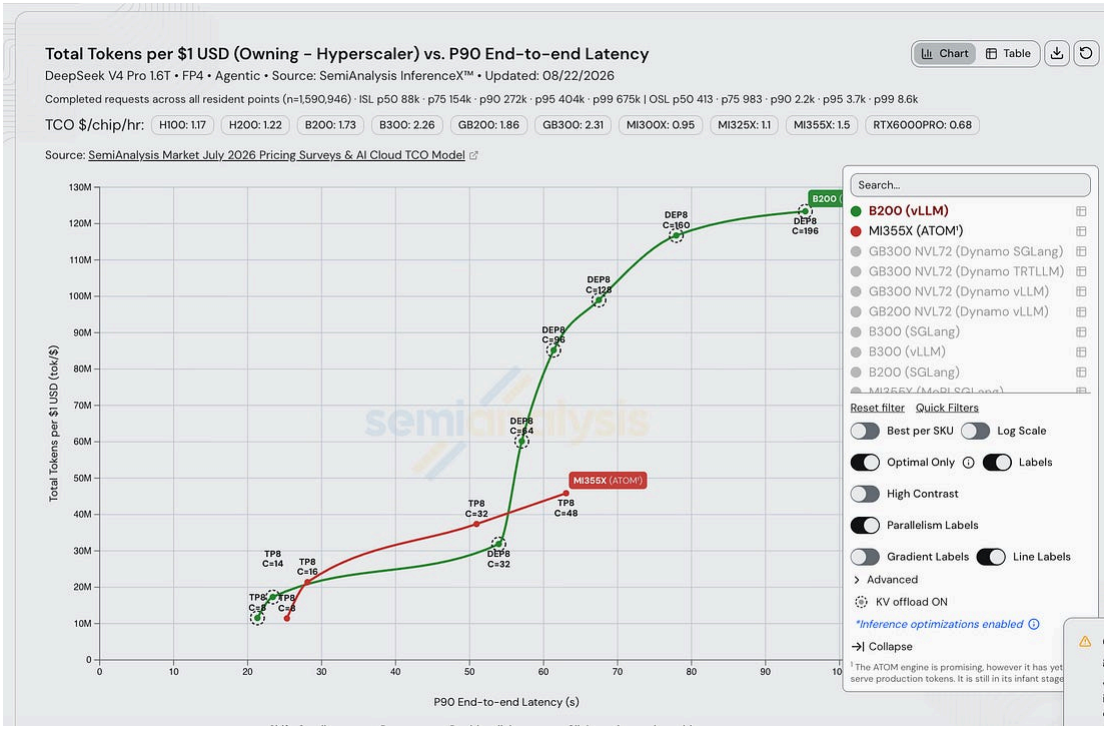




Source: [InferenceX]

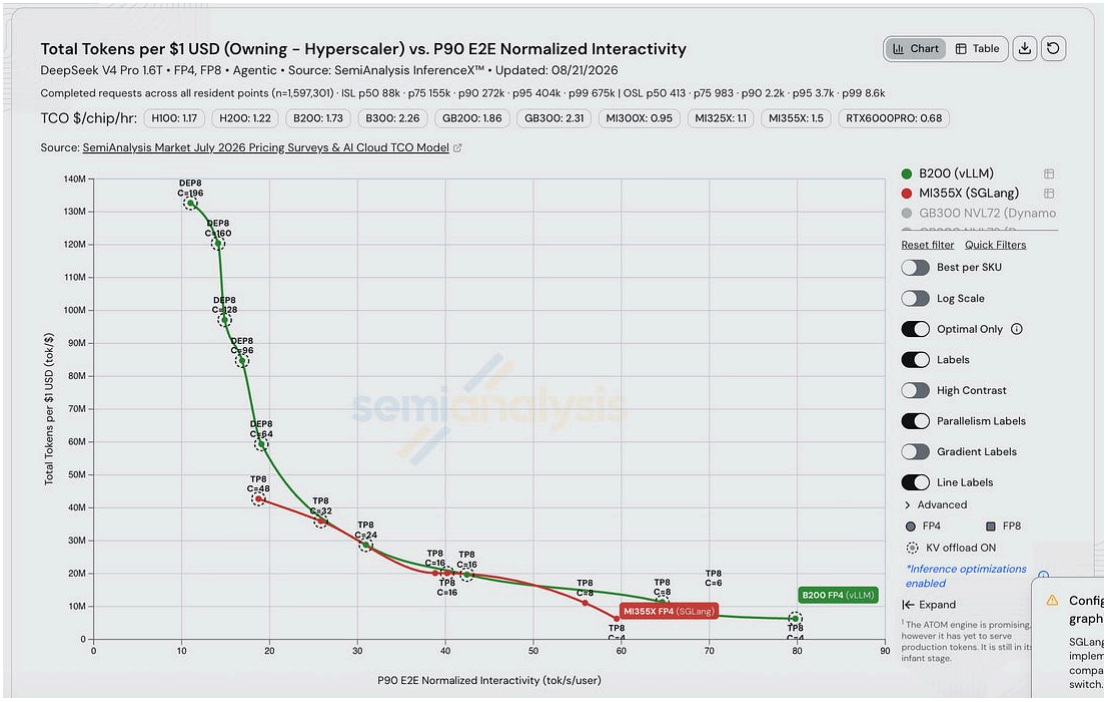
On e2e latency, ATOM MI355X beats B200 vLLM (it does not beat B300 or B200 SGLang though). The issue with this is that most AI labs in China or the west do not want to use ATOM in production besides 1 small advertising business unit at Alibaba Corp due to tons of missing features. The main Qwen LLM org at baba does not use ATOM in production.

QQ:1413221435
QQ:1413221435



Source: InferenceX

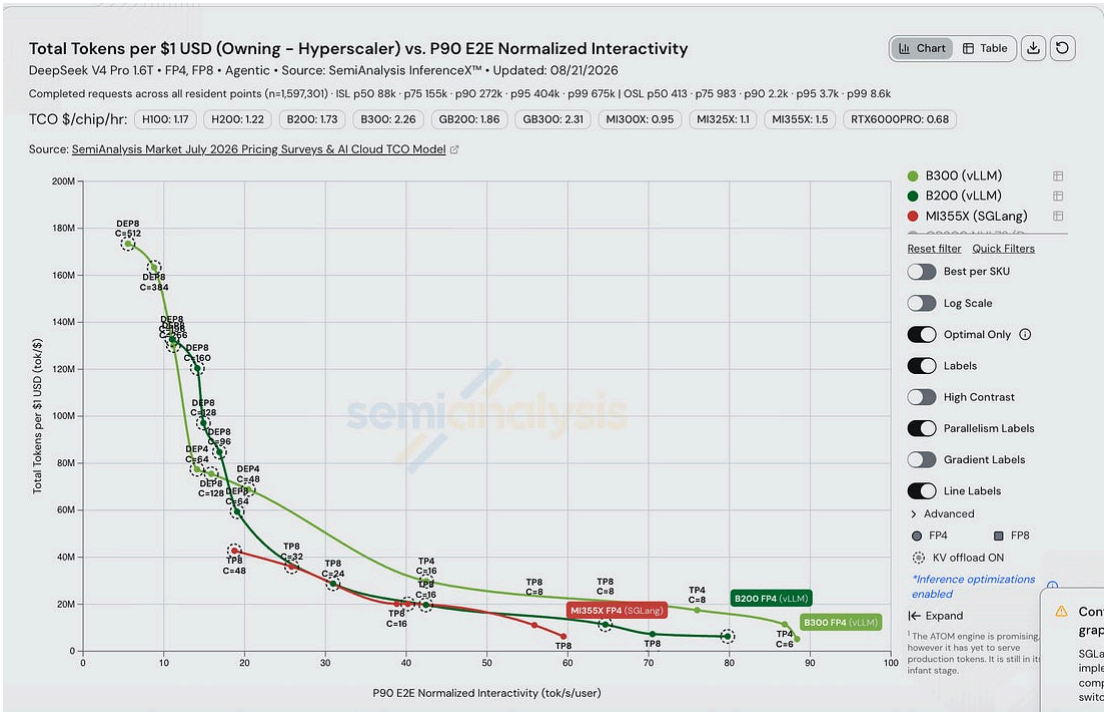
Before August 21, 2026, AMD's MI355X strong SGLang development team was matching B200 vLLM on performance per dollar on end to end (e2e) performance.



Source: InferenceX

However, B300 vLLM and B200 SGLang still beat AMD's MI355X.

QQ:1413221435

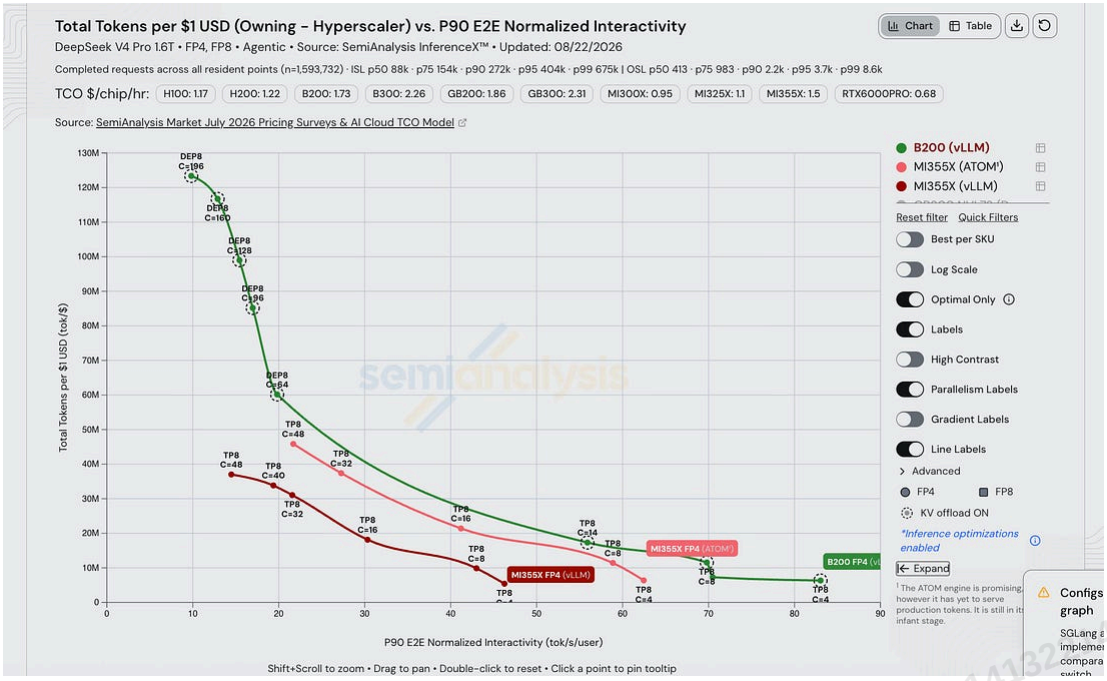


Source: InferenceX

QQ:1413221435

After August 21, 2026, due to optimizations in vLLM from Inferact and Nvidia, the performance per dollar of Nvidia's B200 has surpassed that of the MI355X. This is a close race and we are excited to see the performance optimizations over the next couple weeks. We will be publishing an AgentX update article very soon.

AMD has listed their DeepSeekv4 vLLM optimization and includes lots of exciting things that they can do to improve their performance.

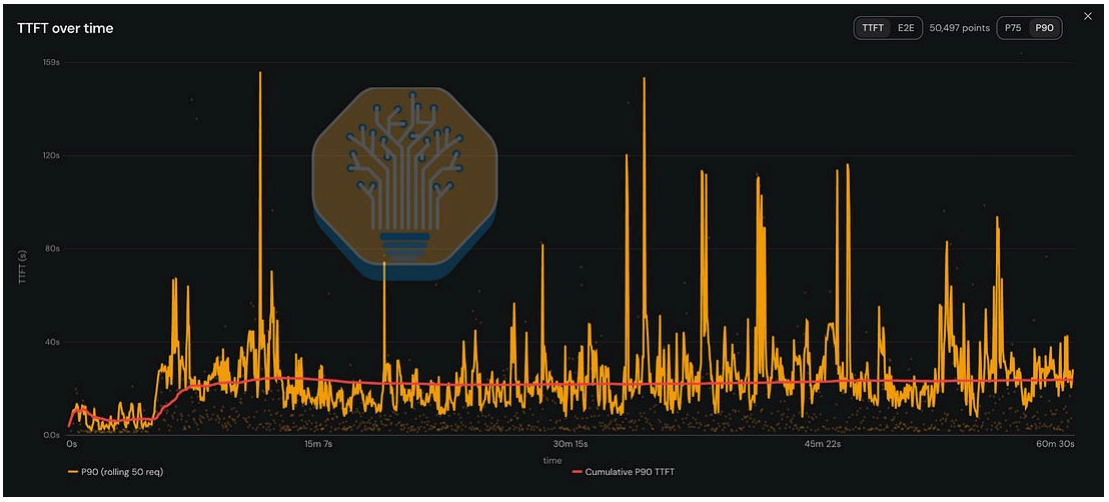


Source: SemiAnalysis

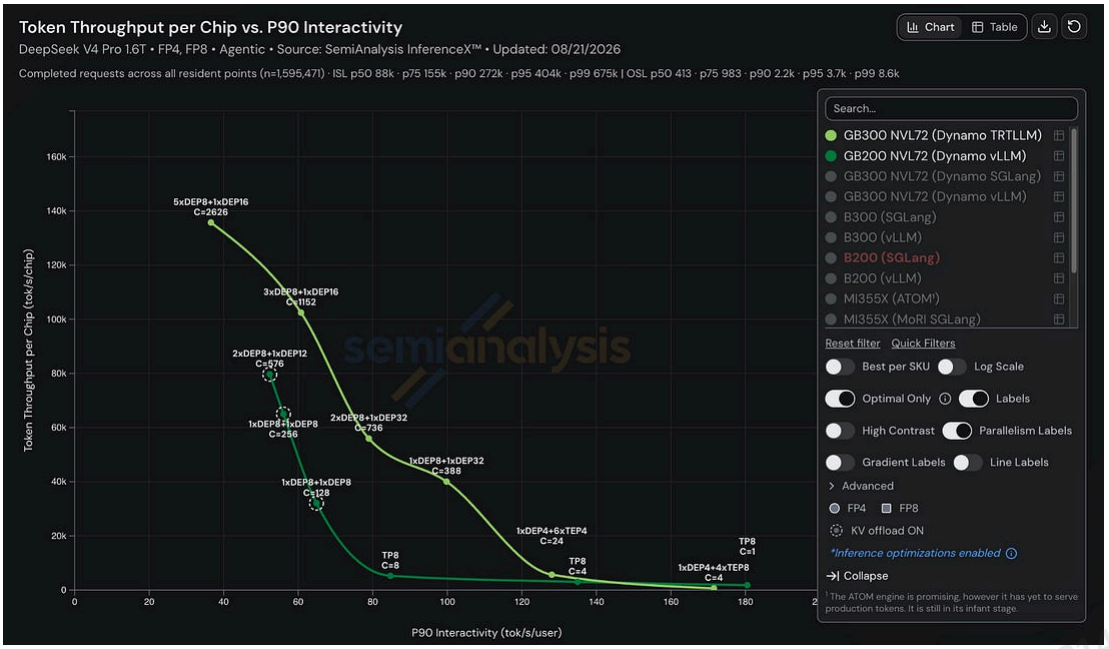
QQ:1413221435

Now turning to Nvidia. Their most competitive solutions are GB300 Dynamo TRTLLM and GB200 Dynamo vLLM. Both configs rely on PD disagg to achieve high throughput at reasonable interactivity. Additionally, GB300 configs employ wide-EP (DEP32) decode instances in order to achieve higher throughput at the middle of the frontier.

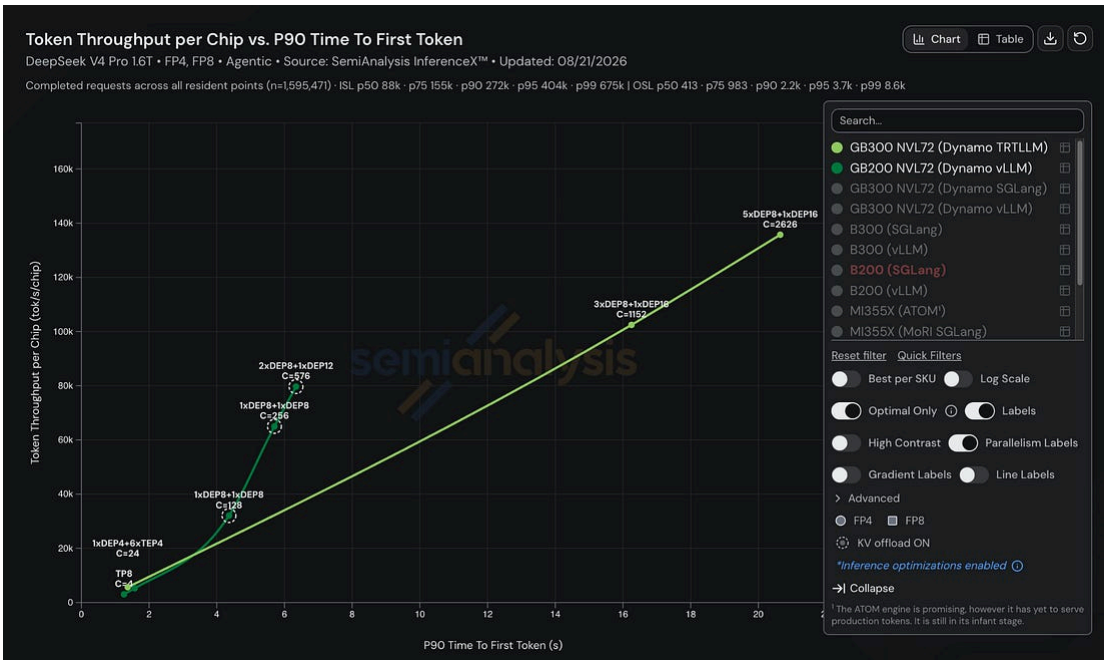
Note that the 2xDEP8+1xDEP12 GB200 point is significantly closer to the 3xDEP8+1xDEP16 GB300 point in terms of TPS compared to TTFT. Again, TTFT is, in general, more sensitive to the “spikiness” of the workload. Since the GB300 point achieves much higher overall concurrency, it incurs more subagent traffic and hence more cold prefills. We can see this in the TTFT chart for the point:



Source: InferenceX

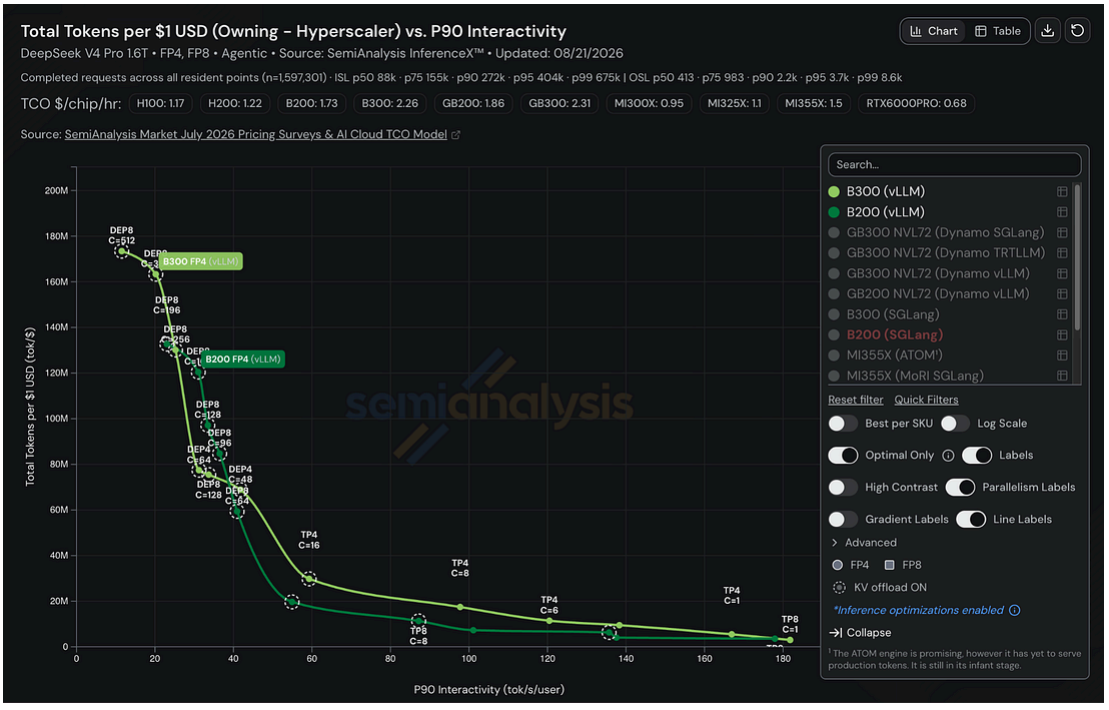


QQ:1413221435



Source: InferenceX

When normalized by TCO, B300 vLLM versus B200 vLLM aggregated performance is quite similar. The main difference being that B300 can “squeeze” out extra throughput, given its 50% increase in HBM capacity over B200.



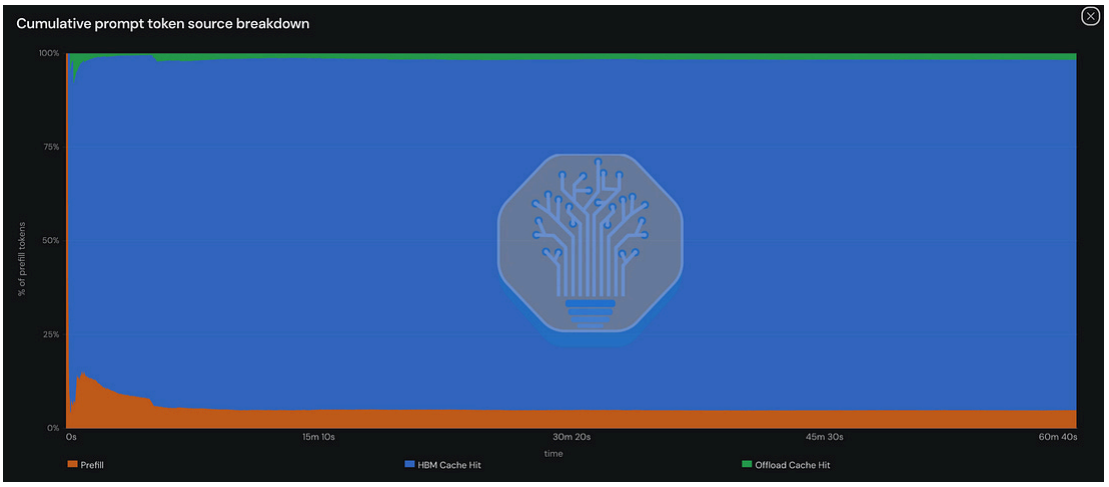
Source: InferenceX

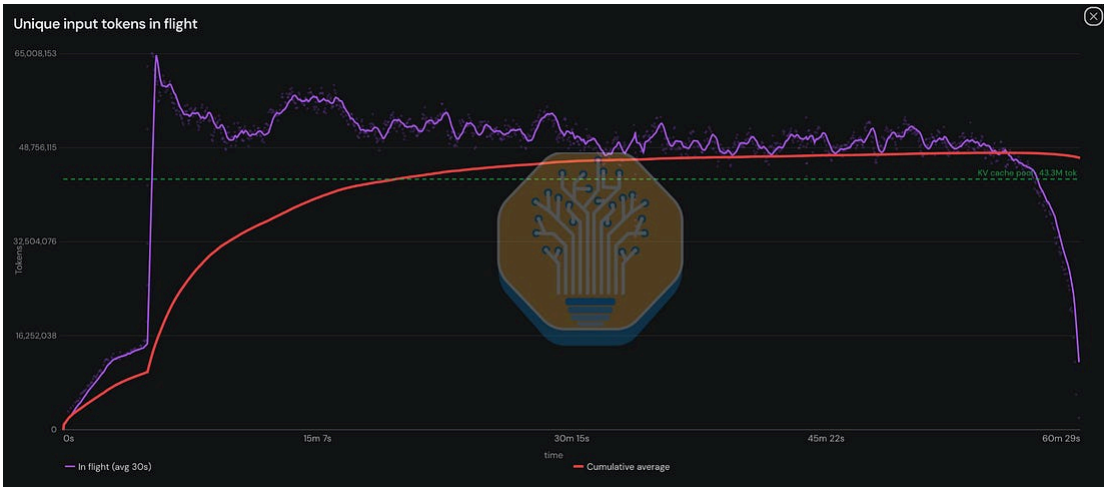
We can further visualize this difference using our server metric visualizations, which are new to AgentX.

Under the load of 384 concurrent agentic traces, B300 vLLM DEP8 w/ 3TB DRAM via vLLM simple offloading achieved a 91% HBM cache hit rate with an additional

QQ:1413221435

1.36% DRAM cache hit rate. This is because the **HBM KV cache working set size** is approximately 43M tokens with this configuration, and the load barely exceeds this number of tokens in flight at any given time.



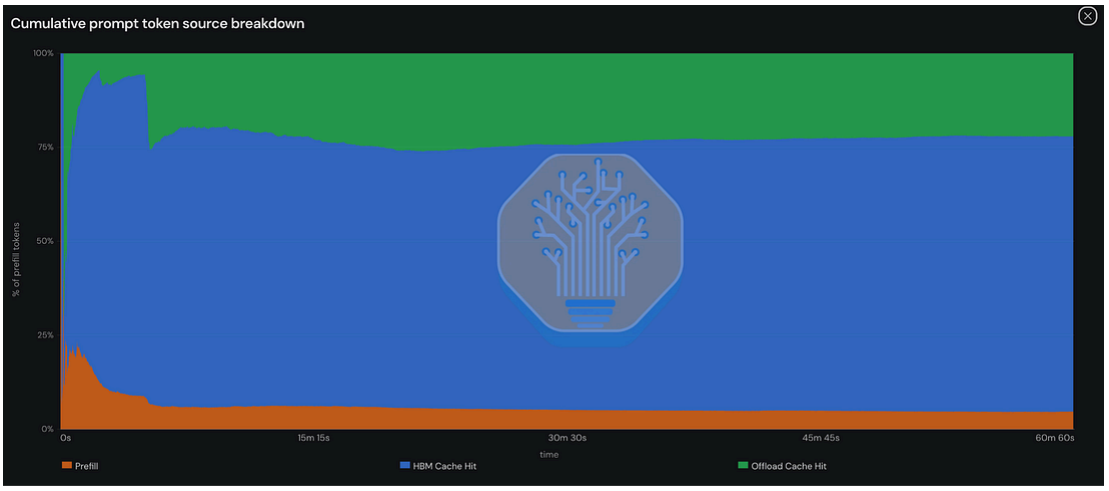


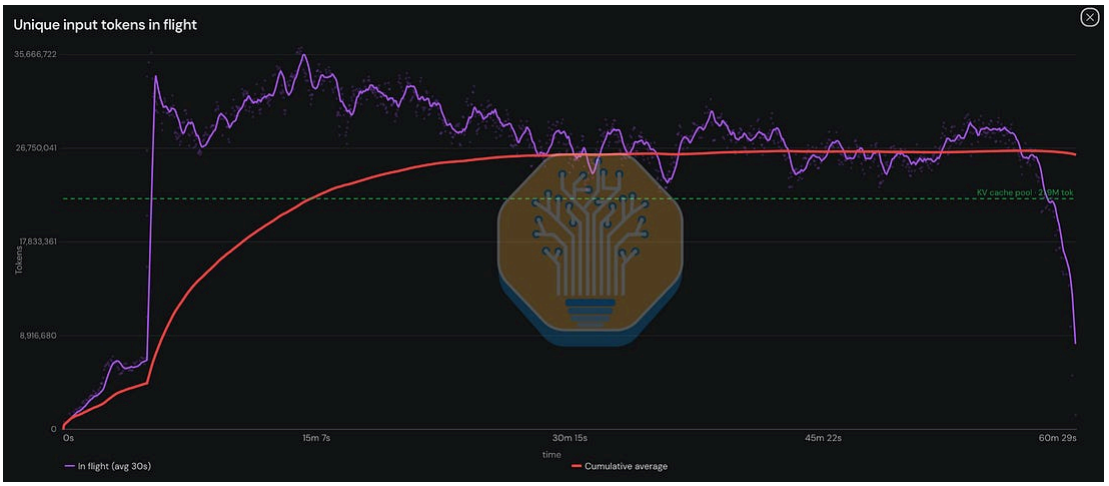
Source: [InferenceX]

With B200 concurrency 196 (all other parameters stay the same), we see only 73% HBM cache hit rate and rely more heavily on DRAM with an offload cache hit rate of nearly 20%. We observe that the HBM KV cache working set size is 22M tokens, roughly half that of B300.

DRAM KV offloading is typically implemented as a write-through cache, meaning every prefix written to the HBM cache is also written to the DRAM cache. Therefore, it is most effective when the amount of DRAM available for offloading is significantly bigger (a multiple of 1.5-3) than HBM KV cache capacity.

QQ:1413221435
QQ:1413221435





Source: [InferenceX]

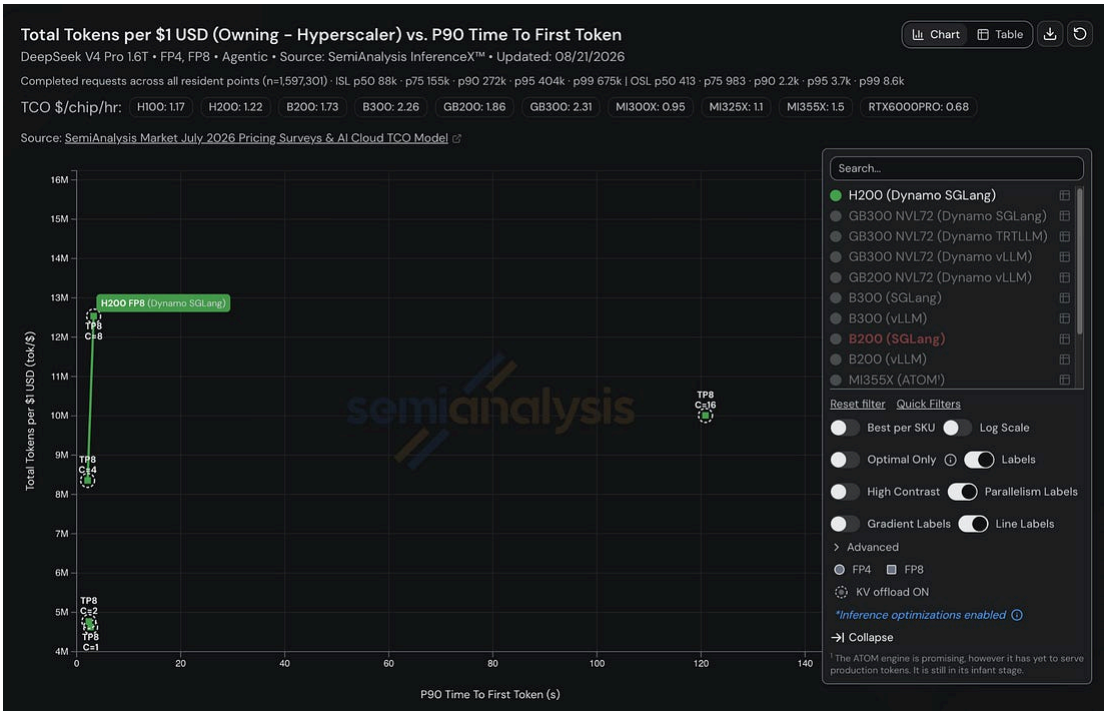
H200 SGLang FP8 is able to serve DeepSeek v4 at low concurrency, and is even competitive with B200/MI355X SGLang from a perf/$ standpoint. However, it cannot compete with the newer SKUs in high throughput scenarios due to lack of HBM.

QQ:1413221435
QQ:1413221435



Source: InferenceX

Furthermore, the reliance on DRAM KV offloading at higher concurrencies leads to unreasonable latency as the number of users scales.

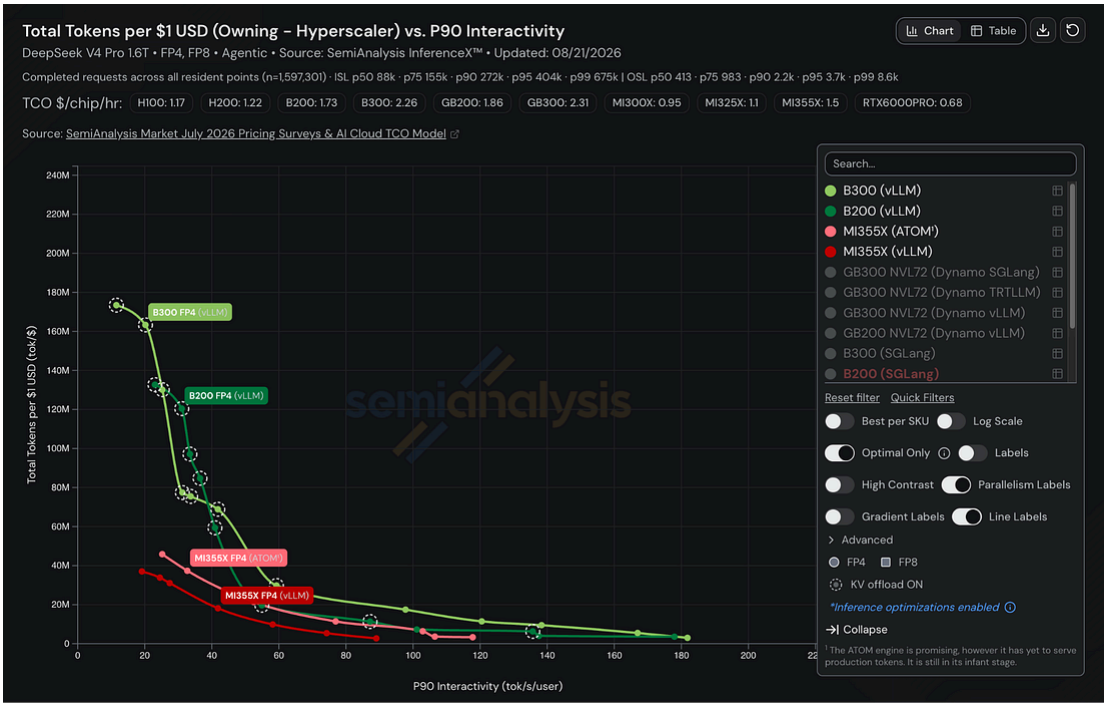


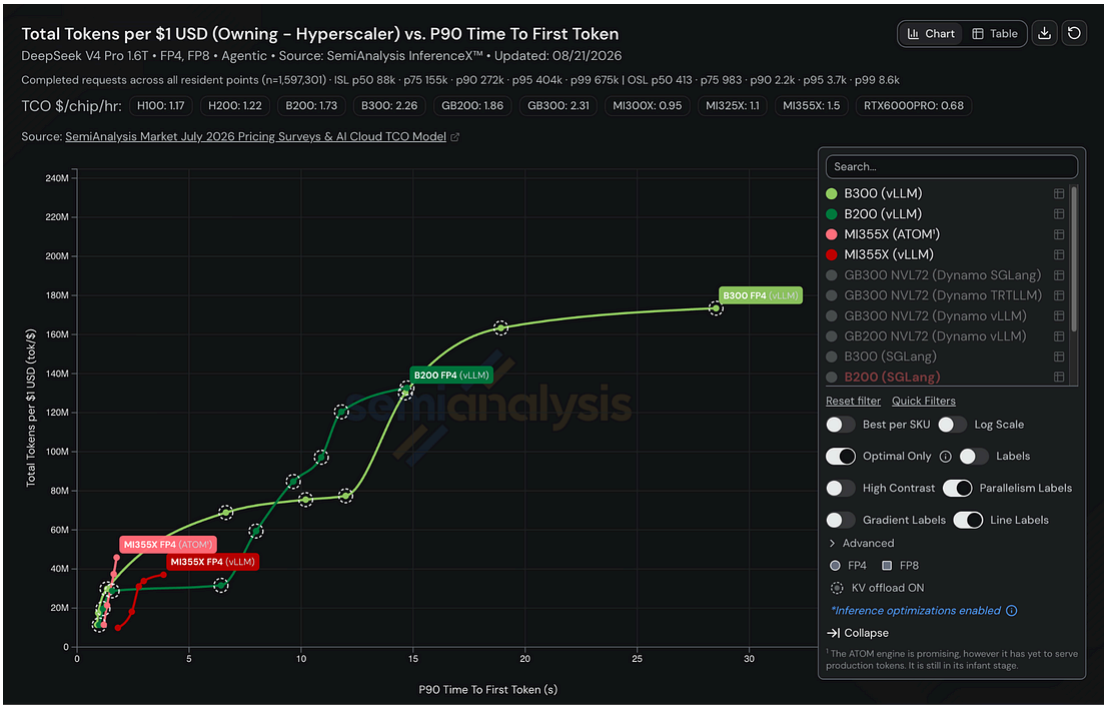
Source: InferenceX

Overall, MI355X performs decently well compared to its main competitors B200 and B300. Performance is most comparable at the lower throughput / lower latency parts of the curve, where only tensor parallelism and more rudimentary kernels are deployed. AMD needs to work on optimizing DEP kernels on MI355X to be more

QQ:1413221435

competitive in the high throughput scenarios, especially given the 1.5x HBM over B200.





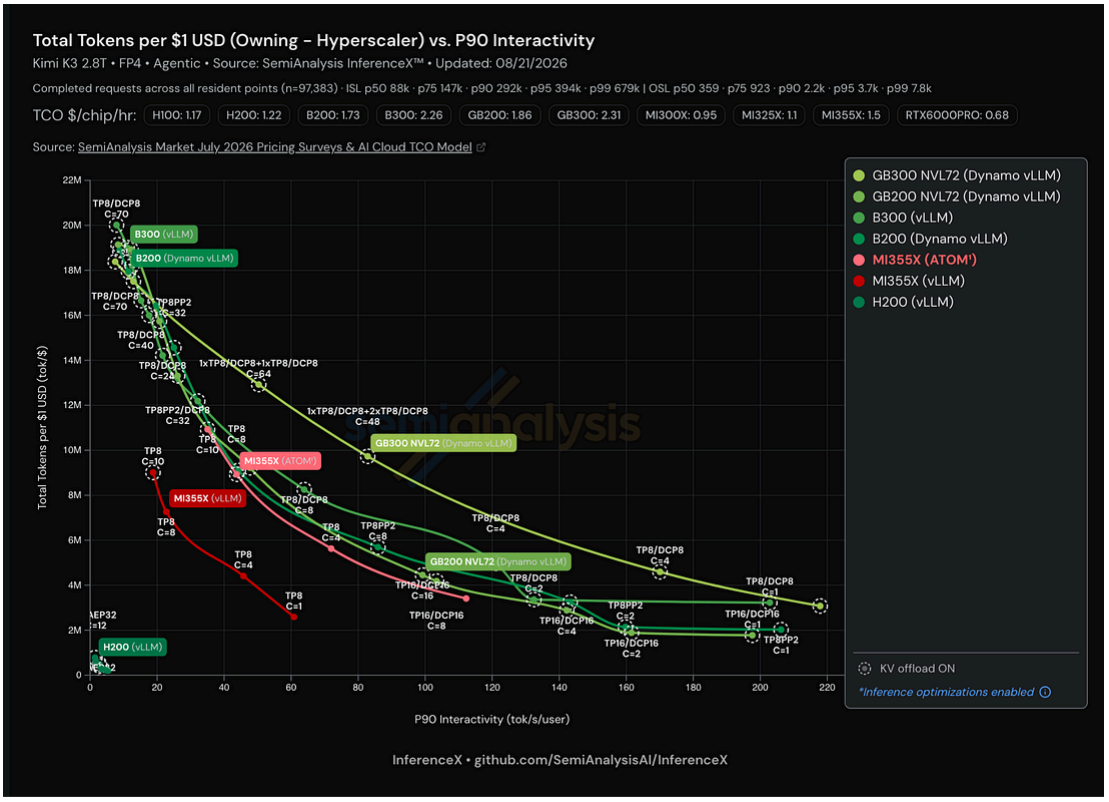
Source: InferenceX

Kimi K3 2.8 Trillion Parameters

Kimi K3 is another frontier open weight model from China that has 2.8 Trillion total parameters. This is in the same range in terms of number of parameters vs Claude's Mythos/Fable5 model architecture. We use this as an open weights proxy model architecture. The Kimi K3 model is so big that it does not even fit on a single B200 server and requires using wide EP/wide TP or pipeline parallelism in order to fit all of

the weights. On vLLM, speculation decoding/DSpark did not compose at all with pipeline parallelism until very recently, so B200 performance on Kimi K3 was horrible and was getting mogged by MI355X since B200 was unable to use speculative decoding with pipeline parallelism.

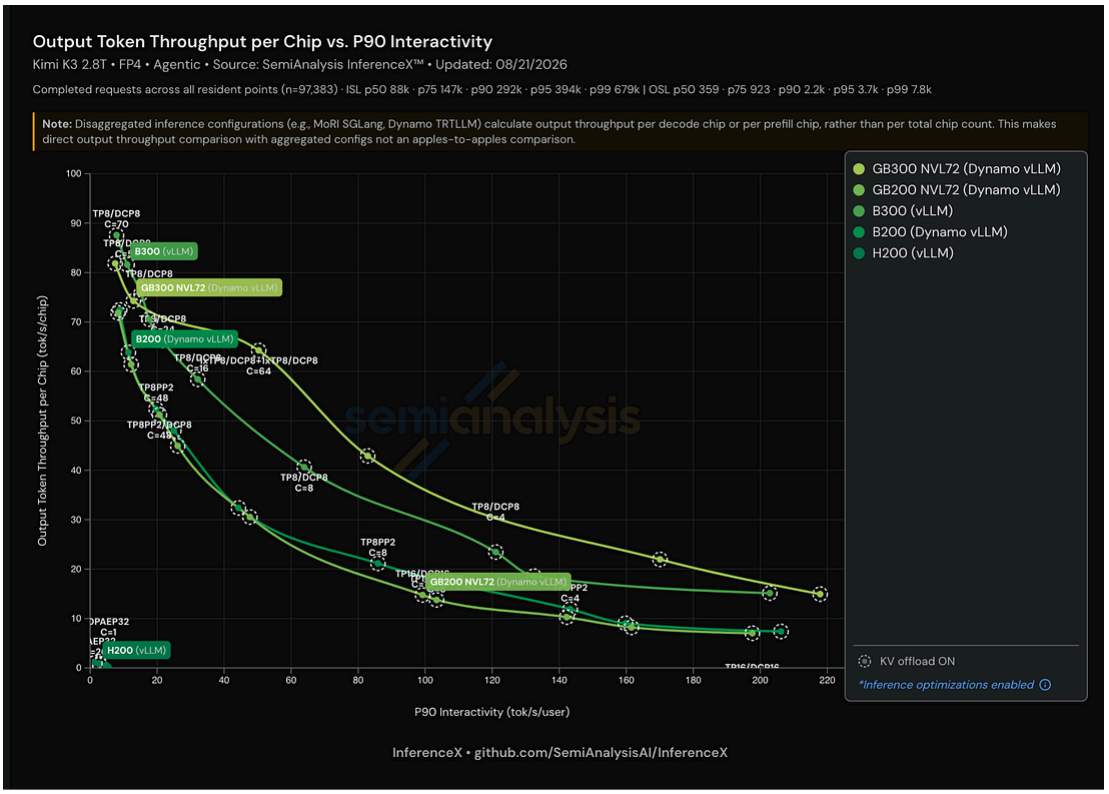
MI355X vLLM worked out of the box on day 0 for short context single turn workloads, but for long context multi turn workloads, MI355X AITER and Triton kernels suffered a massive panic attack on the first week and upstream vLLM was completely unusable for MI355X on realistic workloads.



Source: SemiAnalysis InferenceX

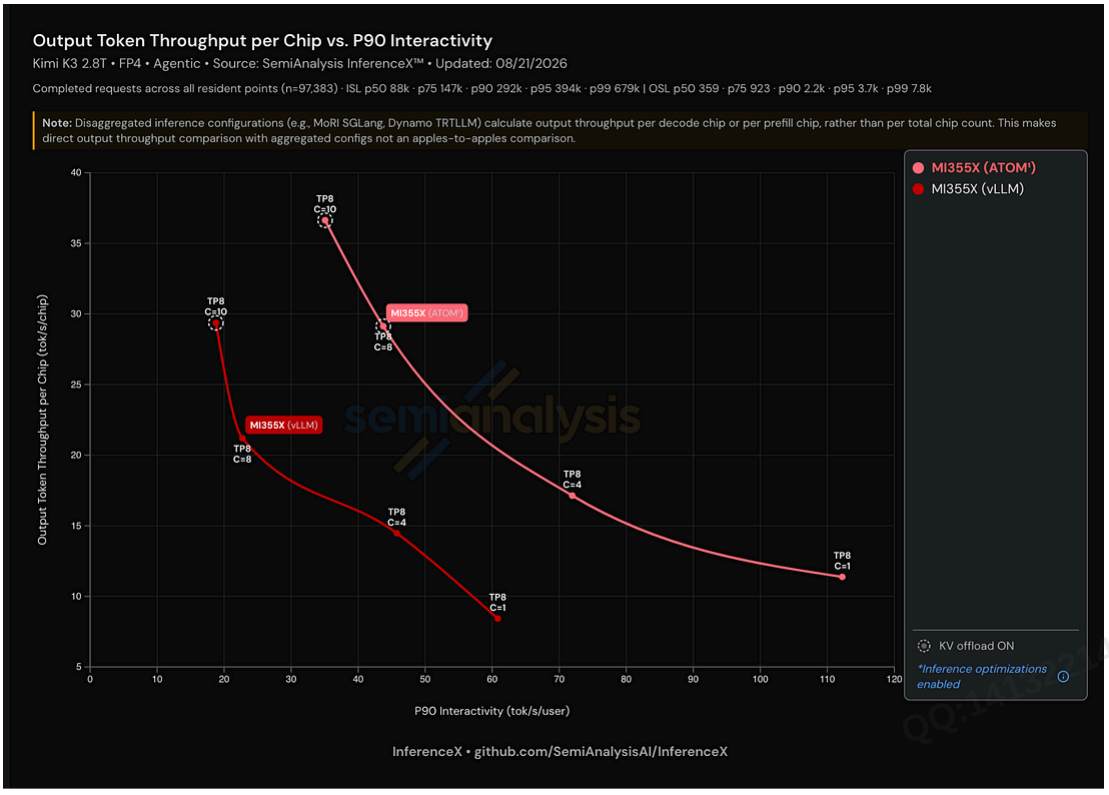
Hopper struggles to serve the AgentX workload for Kimi K3 since Kimi is a massive model and because vLLM maintainers/NVIDIA have not been focusing on optimizing Hopper for Kimi K3. Hopper (SM90) requires custom tuned kernels for K3 along with TP32/EP32 tuned shapes for serving at high interactivity.

QQ:1413221435
QQ:1413221435

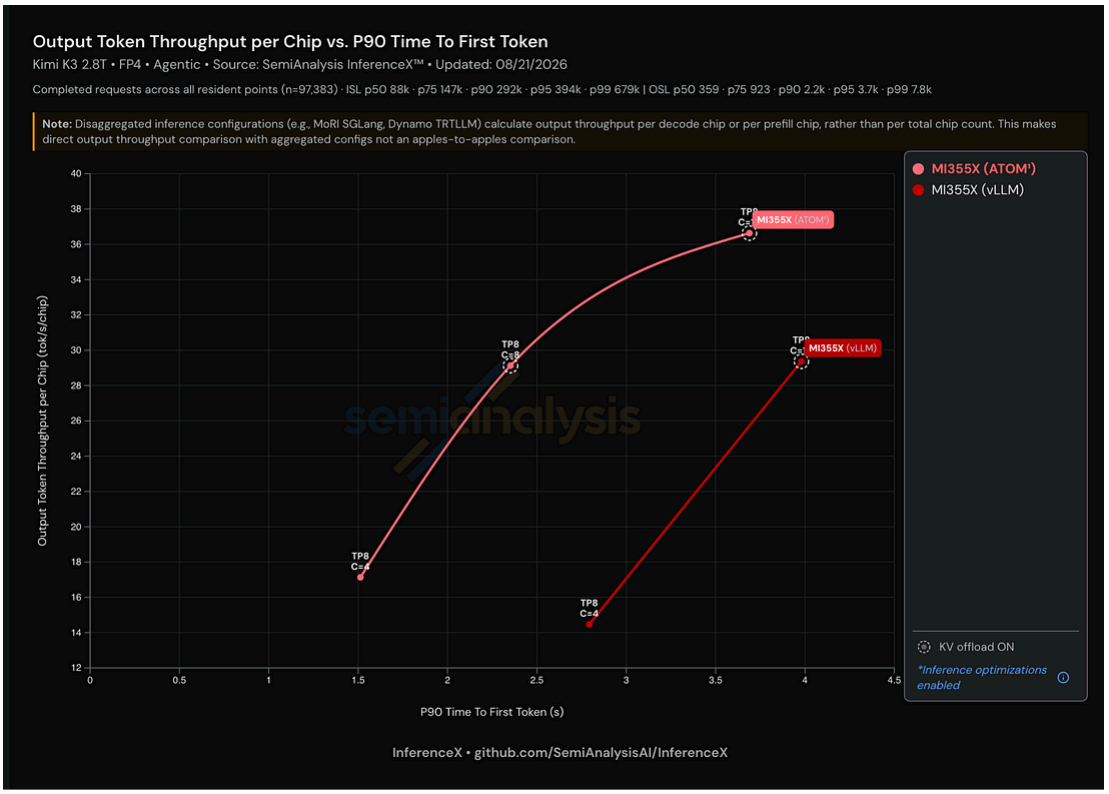


Source: SemiAnalysis InferenceX

We think it is great that AMD is quickly pushing K3 performance forward with ATOM. However, we encourage AMD to further prioritize upstreaming these improvements into vLLM. ATOM is currently AMD's best-performing engine, but vLLM remains the more relevant comparison for customers using an upstream open-source serving stack.

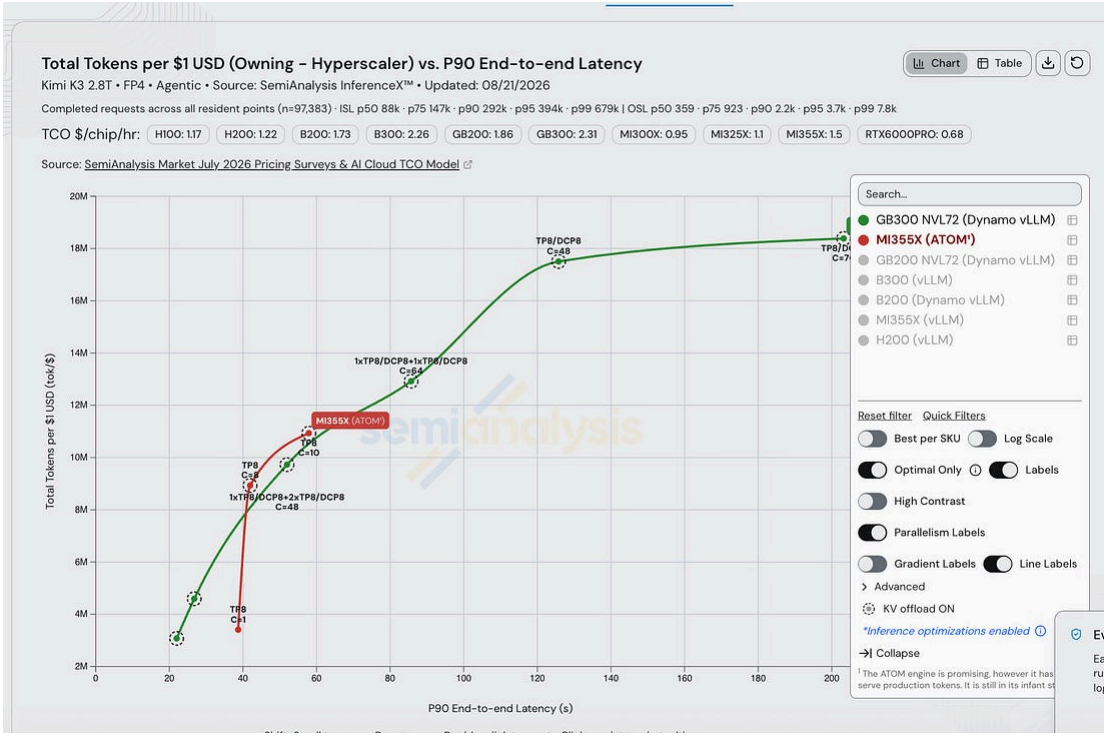


QQ:1413221435



Source: SemiAnalysis InferenceX

On part of the curve between 40 to 60 second e2e latency, MI355X ATOM beats even GB300 NVL72 vLLM on performance per dollar.

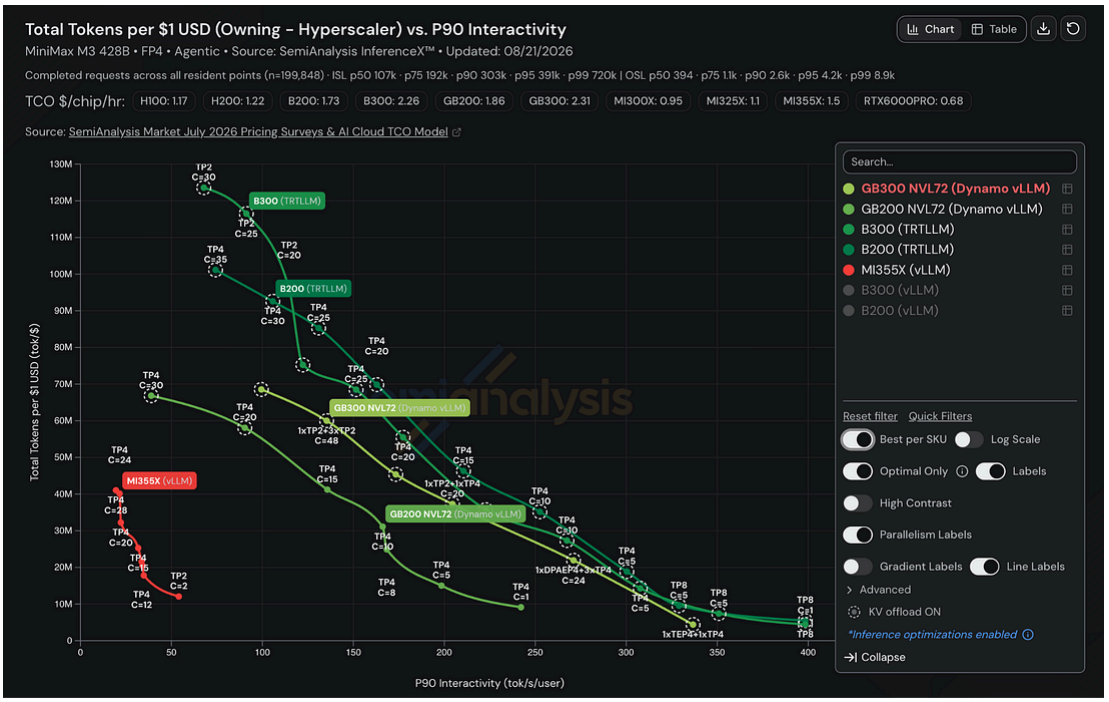


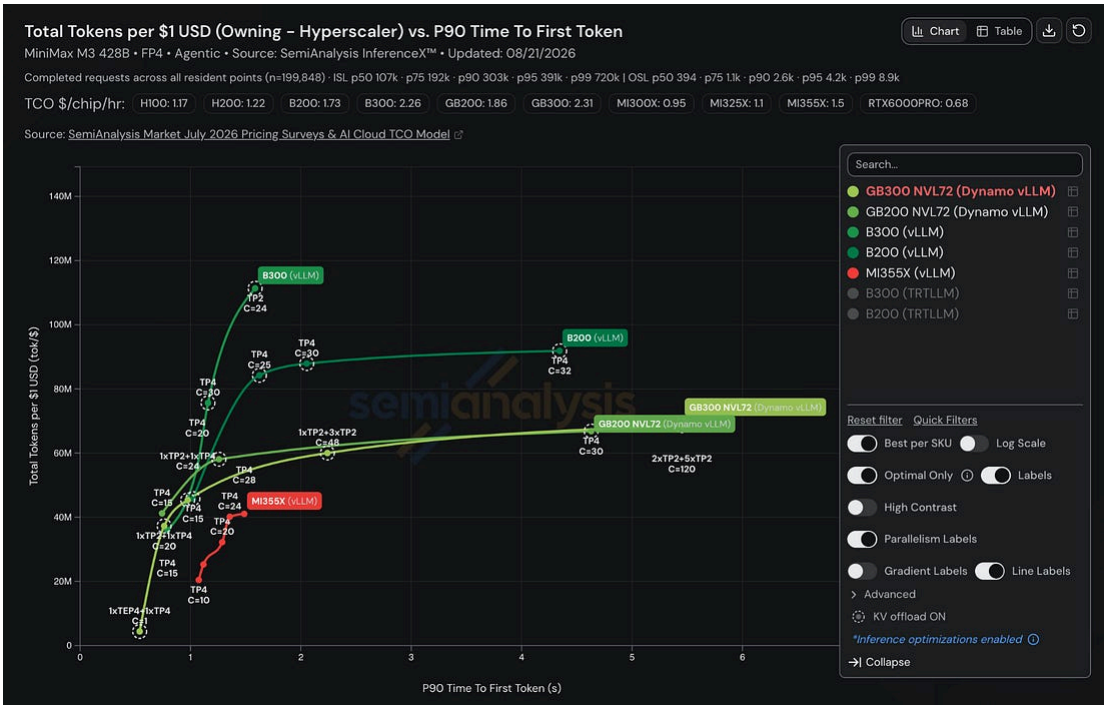
Source: SemiAnalysis InferenceX

MiniMax M3

QQ:1413221435
QQ:1413221435

Nvidia absolutely destroys all competitors on MiniMax M3 432B. AMD software performance is horrible on MiniMax especially at high context length due to AMD engineering leadership incentivizing tuning only for short context single turn workloads and ignoring long context multi turn workloads.





Source: [SemiAnalysis InferenceX]

B300 TRT-LLM TP2 owns the M3 crown. There is a lack of DP-attention points as it is non-optimal on M3 since KV cache locality becomes a routing constraint. This is further explained later on. For GB200 at concurrency 40, TP4/EP4/DPA gets 0.60x the throughput of plain TP4 at a >3x p90 TTFT. At concurrency 32 it hits 28.8% of

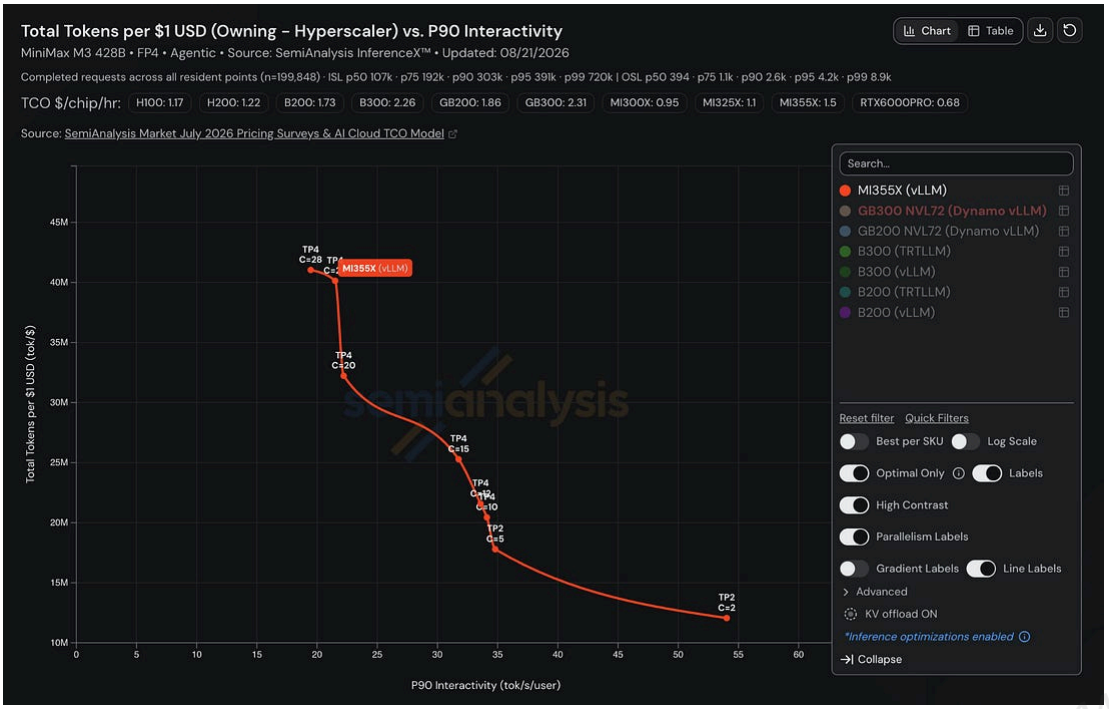
QQ:1413221435

cache vs 96.0% theoretical. Each DP rank owns a private quarter of the pool; a 300k-token session re-landing on the wrong rank recomputes everything. No decode config with EP appears on the M3 frontier, likely as the concurrency is not high enough to balance the loads on all experts.

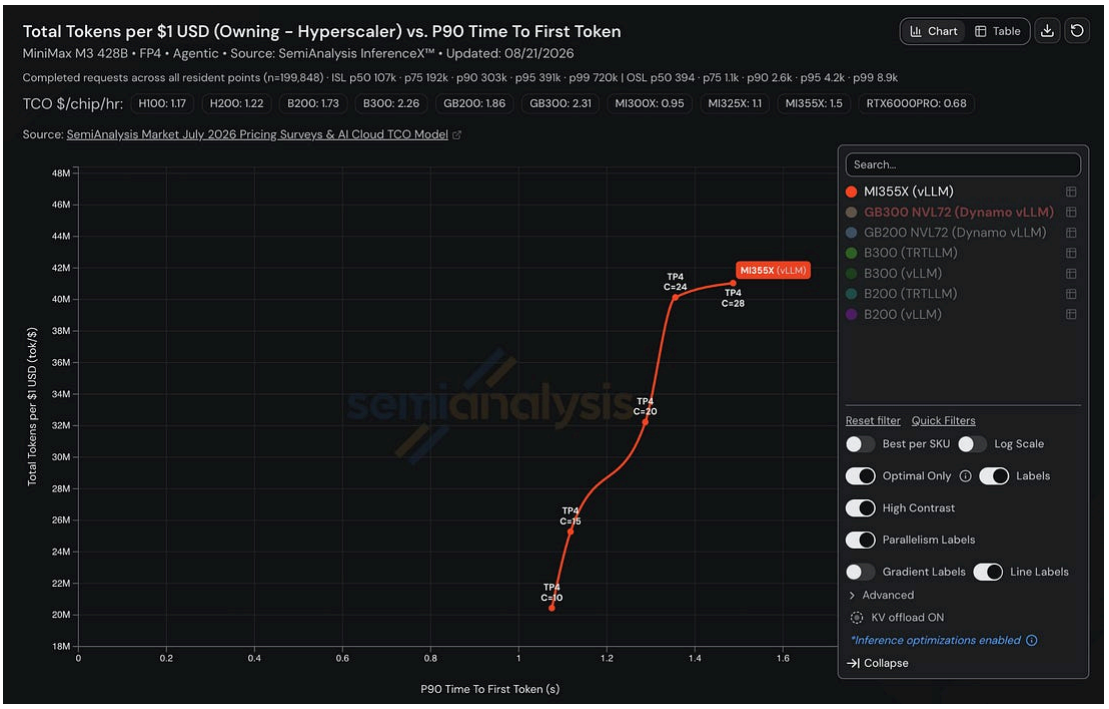
B200/B300 also completely beat their rack-scale counterparts for MiniMax M3 on TCO-normalized throughput. On AgentX, the rack-scale advantage isn't as pronounced as the Dynamo router can become the bottleneck because its work scales with the number and length of live prefixes. Optimizations on this and the several fixes which moved throughput by double-digit percentages are discussed later on in the article. Also, there are no well tuned kernels for wideEP, wide DCP, nor wide TP. And since GB200/300 have higher TCO so without wide ep/wide DCP, it shows up as worse perf per TCO.

With that being said, we also expect further optimizations from Nvidia on their rack scale solutions for this SKU. We will make sure to highlight these in our follow up article.

No submission currently runs context parallelism, despite P90 ISL of 317k. With 4 KV heads, DCP caps at 2 even at TP8, and the MSA indexer needs its own context-parallel handling (a vLLM PR is opened), see the Context Parallelism section for more discussion on this topic.



QQ:1413221435
QQ:1413221435

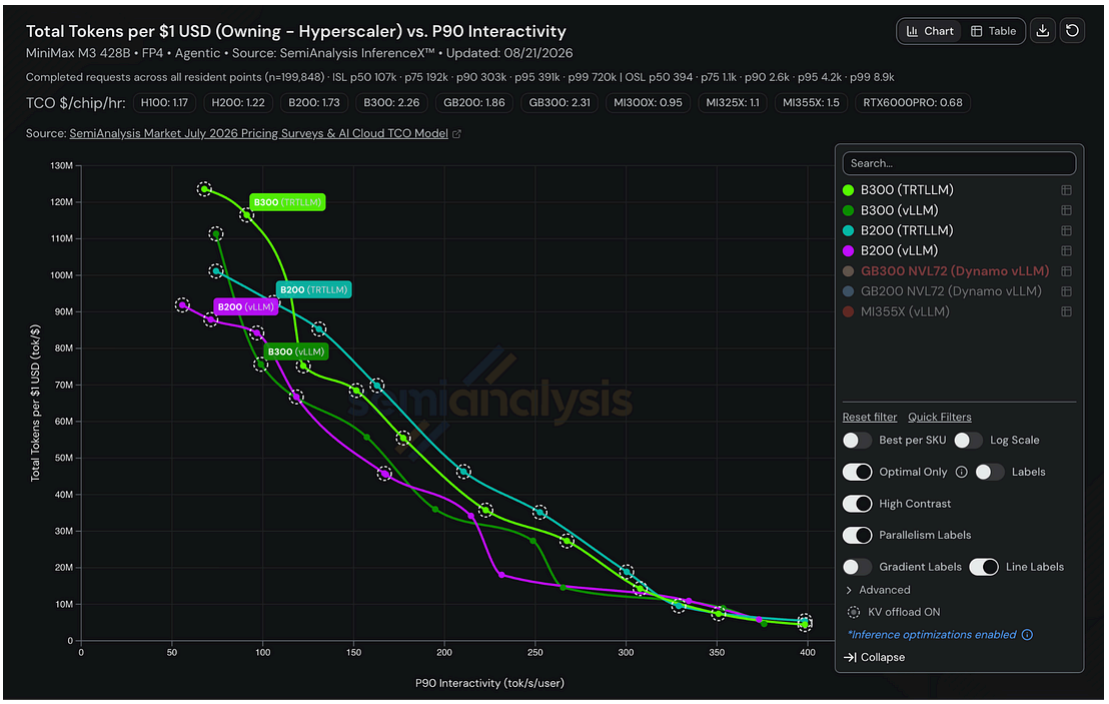


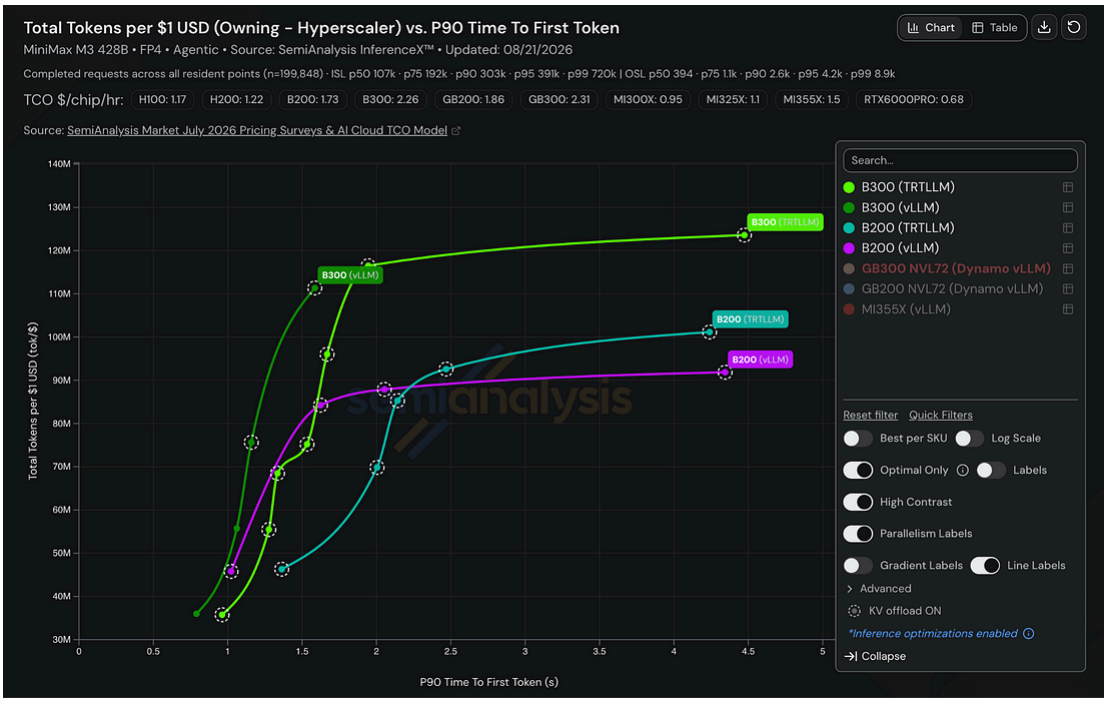
Source: SemiAnalysis InferenceX

All of Nvidia's Pareto optimal points include KV offload above concurrency 20, but for AMD none of the Pareto optimal points use KV offload to DRAM. AMD also uses KV offload less than Nvidia on the other models. The reason for this is that GPU-to-CPU transfers for CPU KVCache offloading are highly inefficient on AMD vLLM. The hipMemcpyBatchAsync API was missing until ROCm 7.14. Without hipMemcpyBatchAsync, vLLM's native Simple CPUOffloading requires doing serialized Memcpy from CPU to GPU instead of batching them into larger message sizes.

It is also worth mentioning that vLLM performance is very comparable to TRT-LLM in terms of throughput versus p90 interactivity. Additionally, vLLM performs better in terms of throughput versus p90 TTFT.

QQ:1413221435
QQ:1413221435





Source: SemiAnalysis InferenceX

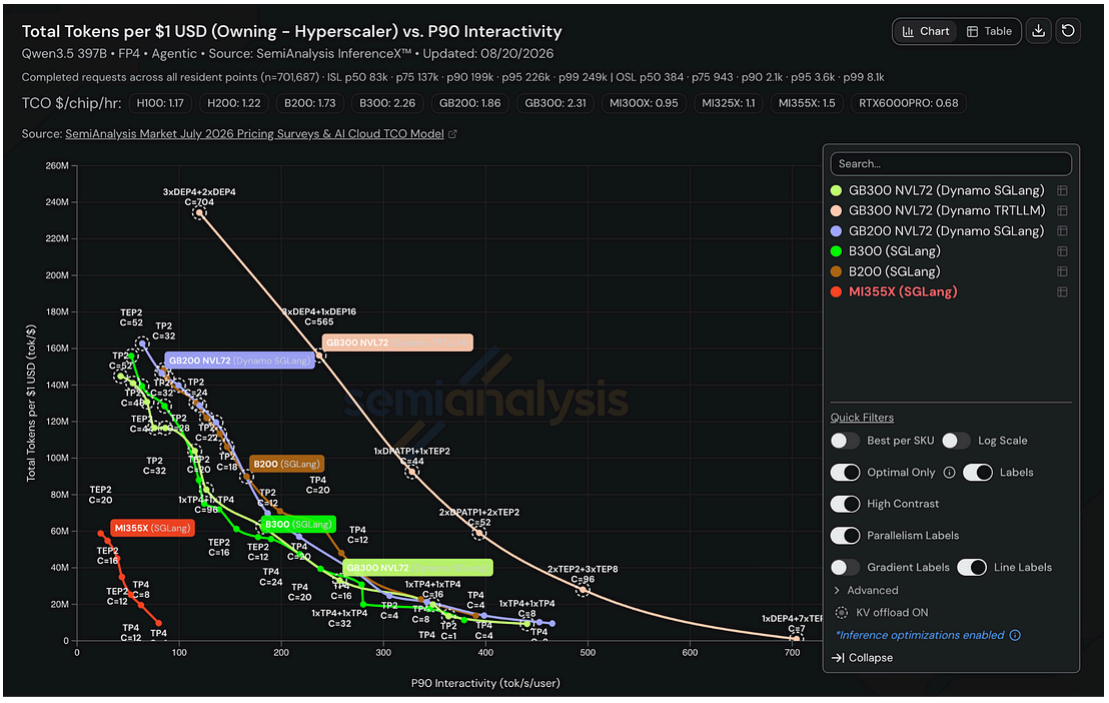
Qwen3.5 397B

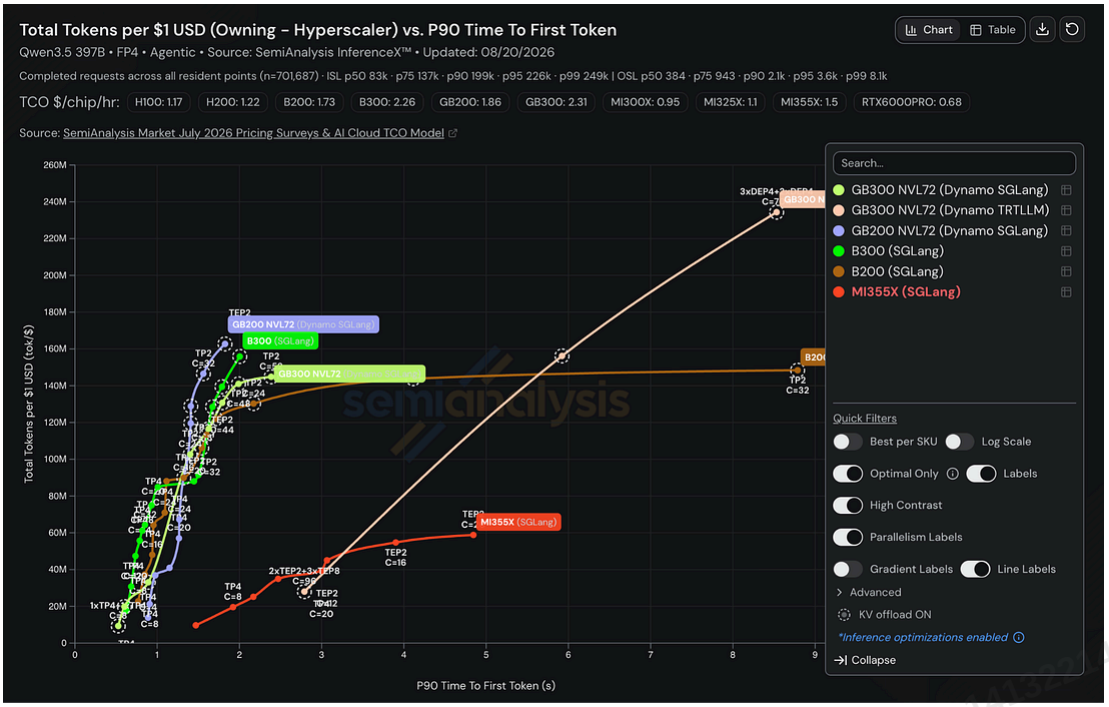
Qwen3.5 397B uses GatedDeltaNet instead of vanilla attention for every couple of layers. GatedDeltaNet was invented at MIT/Nvidia Research and has a theoretically constant state storage requirement instead of vanilla attention's linear storage requirements. This means that it has lower storage requirements compared to an equivalent dense attention model. Unlike end to end model training research like the Nemotron disaster, Nvidia Research is great at fundamental research like GDN and LatentMoE which is used on frontier models.

QQ:1413221435

Note that this model's native max context length is 262k tokens, so we use the truncated dataset. This simulates a workload on a smaller model where the max context length would be frequently reached with many compactions, how users would actually use this model.

Qwen3.5 397B is a strong hold for NVIDIA on SGLang versus SGLang, with over 20x better performance at 90 tok/s/user. There is currently zero competition from AMD for Qwen3.5 SGLang.

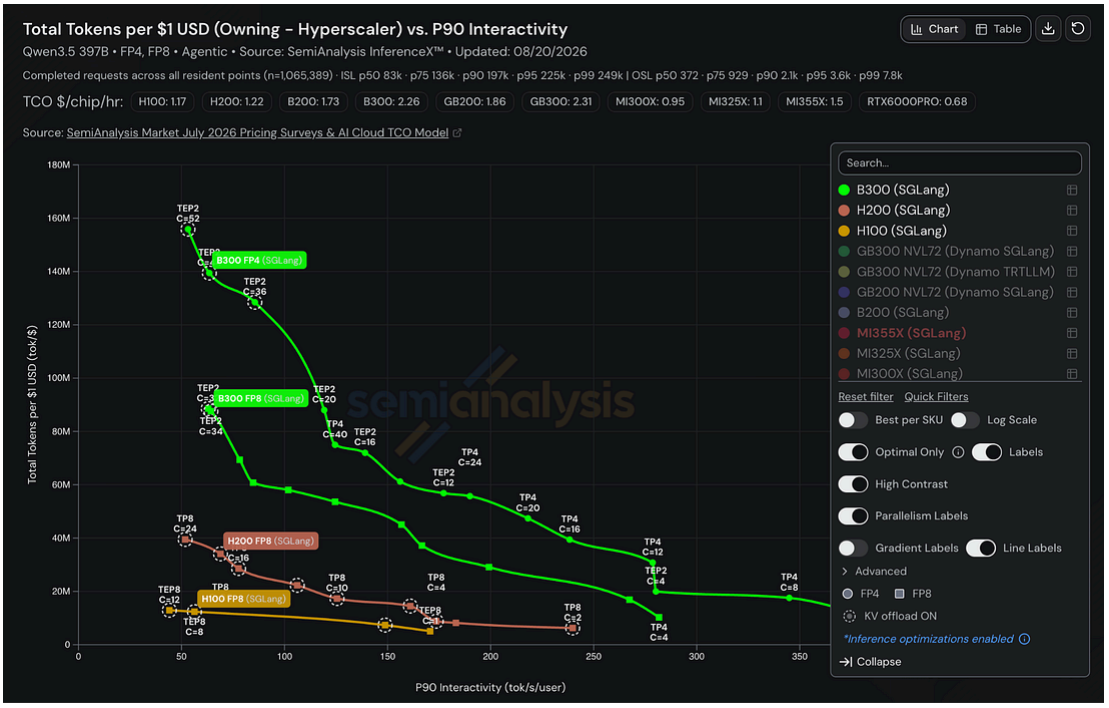




Source: SemiAnalysis InferenceX

Again, we observe Nvidia over optimizing for interactivity at the cost of TTFT, especially in the case of TRT-LLM. In the graph above, all of the Nvidia SGLang submissions have much lower p90 TTFT when compared to TRT-LLM.

Compared to H100, on Qwen3.5, B300 FP4 has 12x better performance per dollar.



Source: SemiAnalysis InferenceX

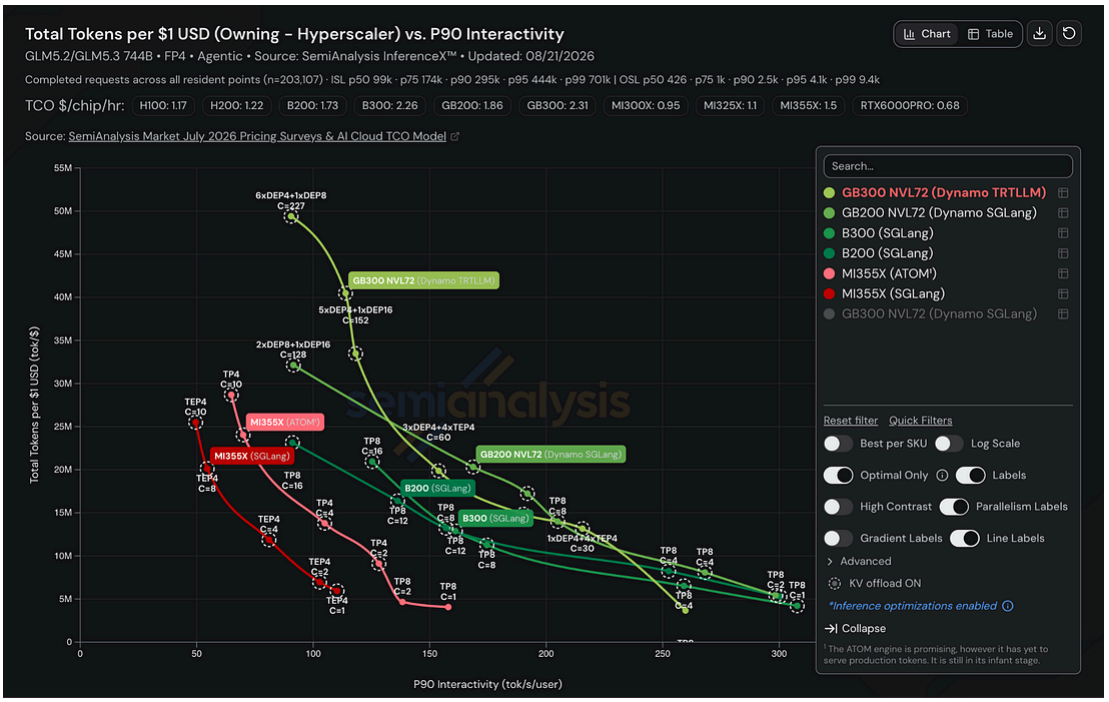
GLM 5.3

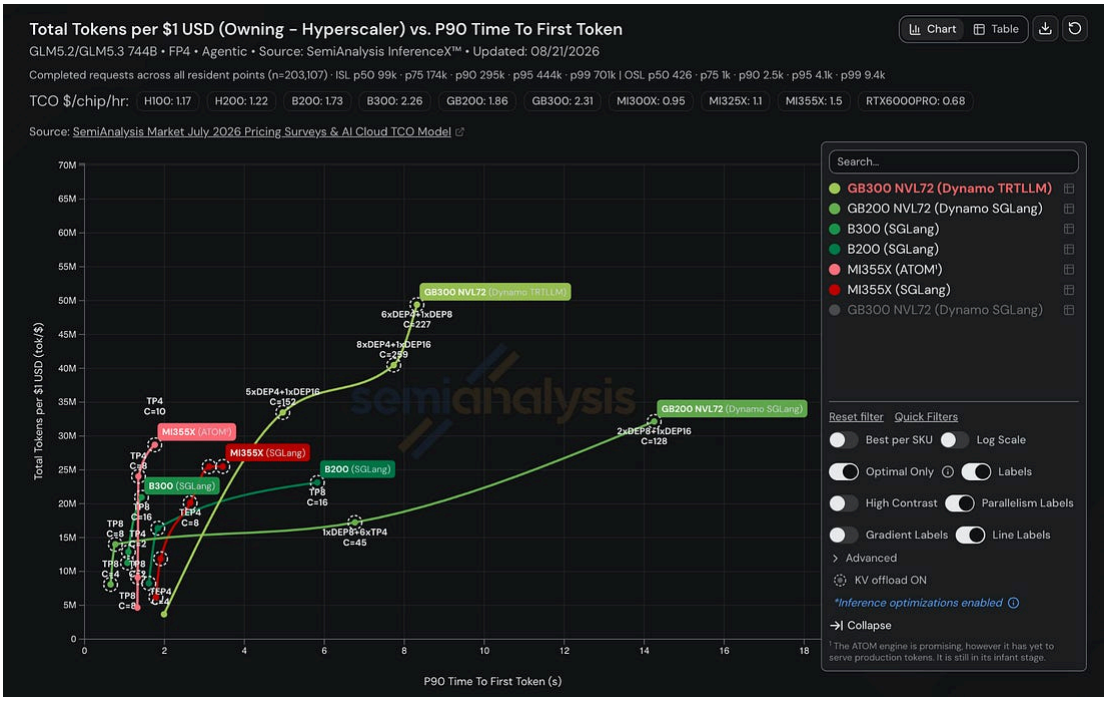
GLM 5.3 builds on top of GLM5.2 744B with additional post training. This is a frontier level model.

In terms of OSS SGLang performance, this is another model where Nvidia again beats AMD on realistic agentic inference performance. At 150 tok/s/user p90 interactivity, Nvidia has up to 5x better cost efficiency, With the current state of AMD software, at 150 tok/s/user, Nvidia's performance advantage is so great that even if the competitor chip hardware was sold for free (but with providers still of course paying for datacenter hosting and power and other operating costs), cost per token would still be cheaper when using Nvidia.

We look forward to AMD's performance optimizations in the upcoming AgentX update article in a couple of weeks, which will also include some other very exciting results.

QQ:1413221435
QQ:1413221435

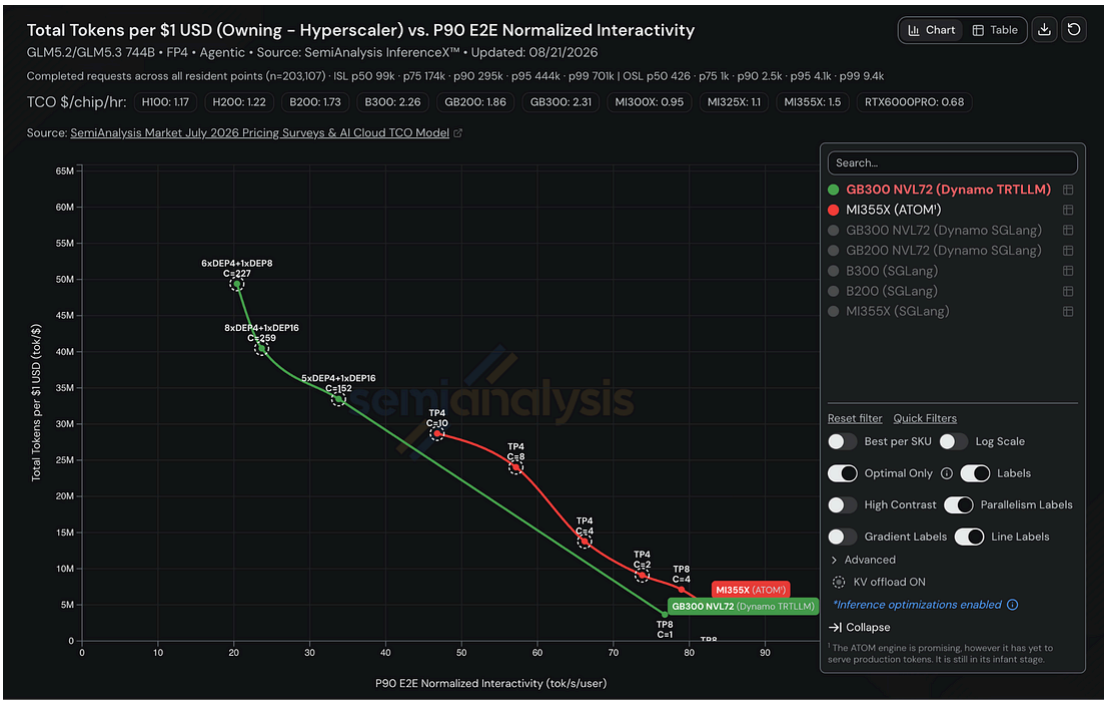




Source: [SemiAnalysis InferenceX]

When looking at ATOM, AMD has better performance per dollar than GB300 NVL72 SGLang and even TRTLLM for some parts of the range of p90 E2E Normalized Interactivity. Great work to the AMD team on these results. Again, we look forward to AMD porting over these optimizations to SGLang. We also are looking forward to NVIDIA quickly optimizing GB300 NVL72 in the coming weeks.

QQ:1413221435
QQ:1413221435

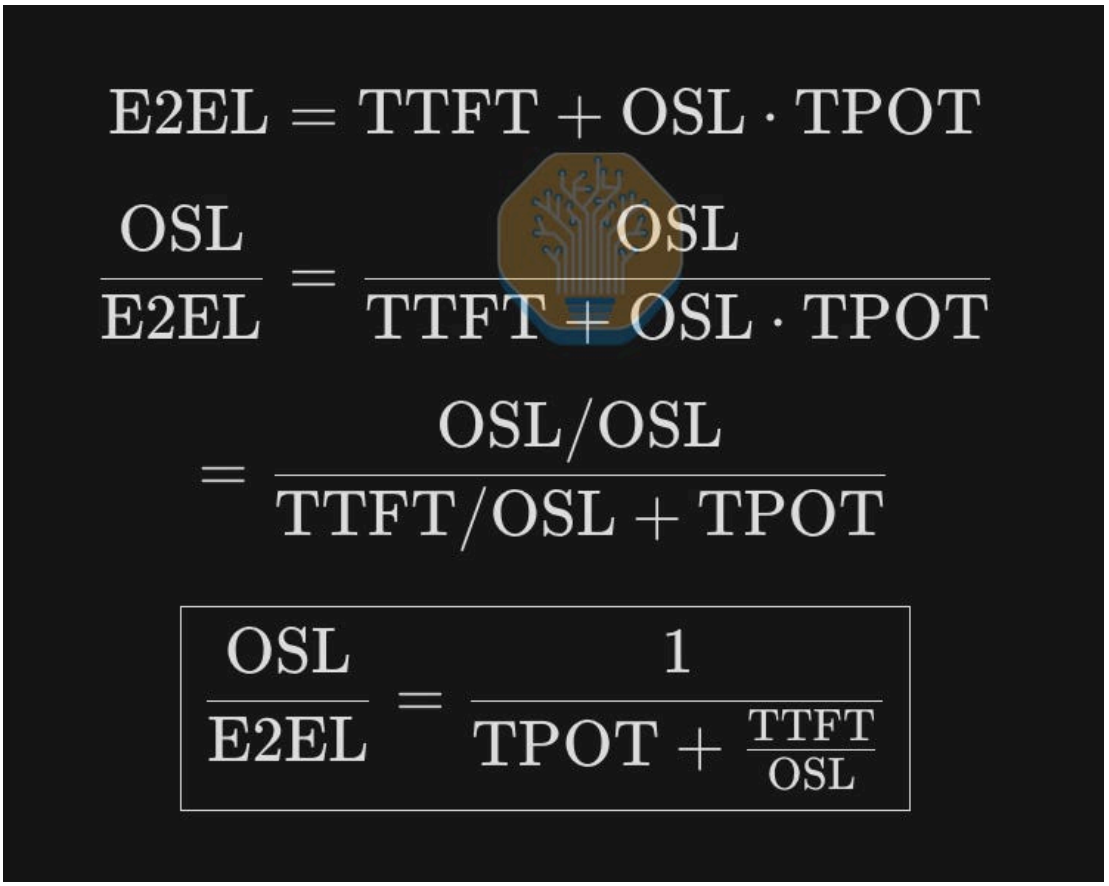


Source: SemiAnalysis InferenceX

We take a moment to introduce an experimental metric which we call **E2E Normalized Interactivity**. At a high level, this metric is supposed to evaluate how fast a user experiences responsiveness when considering both TTFT as well as TPS. It is defined by OSL/E2EL. Substituting the fact that E2EL equals TTFT plus OSL times TPOT (in reality only OSL - 1 tokens are decoded), we get the following equation.

This is effectively interactivity (the 1/TPOT portion) plus an additional penalty proportional to TTFT.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis

QQ:1413221435

Please note that this metric is experimental and is not perfect. For instance, it heavily penalizes high TTFT and doesn't capture all the nuances of certain optimizations such as PD disaggregation. All submissions for AgentX v1.0 optimize for both regular interactivity and TTFT separately. We will continue working on new north star metrics that reflect all nuances of modern agentic inference.

AgentX Industry Impact - Optimizations for Agentic Workloads

The most impactful result from AgentX in its first months was not producing open source datasets, instead, it has been the **industry impact of 50+ upstream PRs** created by AgentX partners to optimize real world agentic workloads using AgentX as the north star. AgentX's real agentic traffic benchmarks not only the raw prefill and decode kernels, but also tests the entire end to end token generation process from KV cache lifecycle, hybrid-attention cache correctness, CPU KV offload, transfer progress, routing affinity, to incremental tokenization, request serialization, and scheduler bookkeeping. All of these steps matter for every production agentic deployment.

QQ:1413221435

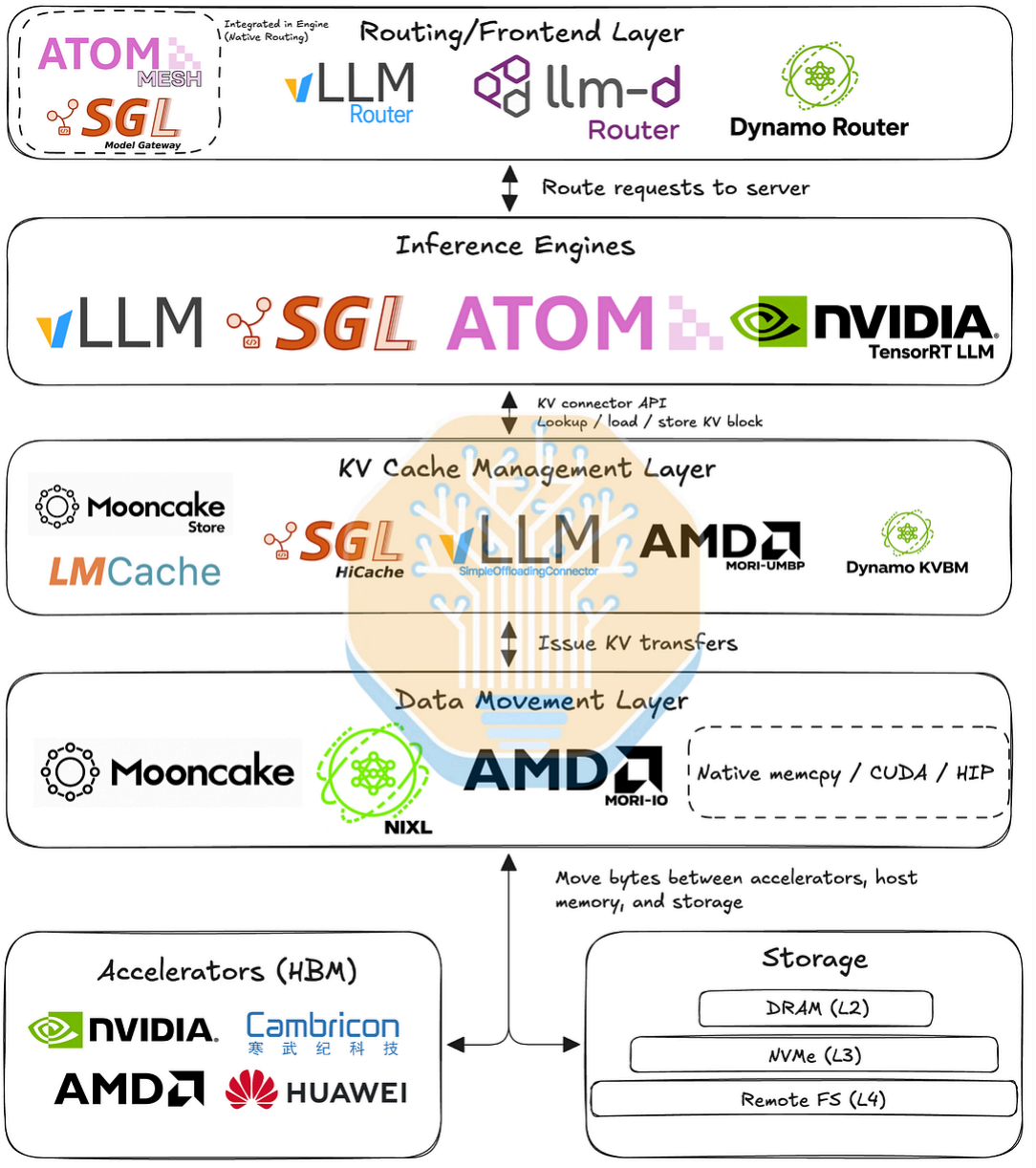
This is just a continuation of our ongoing mission to help the ecosystem accelerate improvement and deliver light speed improvement in software. A great example is SemiAnalysis's multi-year collaboration with AMD's software development team in which we have been providing continual feedback and input to help to modernize their software development principles. This has not only led to many changes that have accelerated AMD's progress, but also has been instrumental towards getting AMD open source closer to first class on agentic workloads.

A Brief Introduction to the Distributed Inference Ecosystem

As mentioned, agentic inference is inherently a system-wide problem as opposed to just a chip/kernel level problem. Additionally, when there are large *distributed systems* handling hundreds of thousands of agentic requests, the scheduling of requests and management of KV cache becomes non-trivial and has legitimate performance implications. For example, sub-agents give bursty KVCache patterns where not properly optimizing, it will improperly evict the main agents cache.

The following diagram illustrates the stack at a high level. At the top, routers (sometimes referred to as "frontends") route requests to different workers. For instance, in the case a server is running data parallel attention, there are separate KV caches for each DP rank. In order not to thrash any one of the KV caches, requests are routed according to different policies such as consistent hash, where requests in the same session/subagent are routed by their unique ID.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis

For most routing policies, there is nothing significantly different about each router implementation. Some are separate components such as vLLM router and llm-d router while others are integrated in the engine such as SGLang model gateway and ATOM Mesh.

After a request is routed, it is handled by the scheduler of an inference engine such as vLLM, SGLang, etc. The engine is responsible for actually performing the inference and returning the result over an API. Additionally, each engine has an interface for connecting the engine's internal KV cache to external KV cache managers. This allows for a "pluggable" ecosystem where different KV cache managers can integrate with a variety of inference engines.

A simple deployment, used in the current AgentX results, runs Mooncake alongside vLLM on the same node. Each vLLM worker embeds a Mooncake Store client and

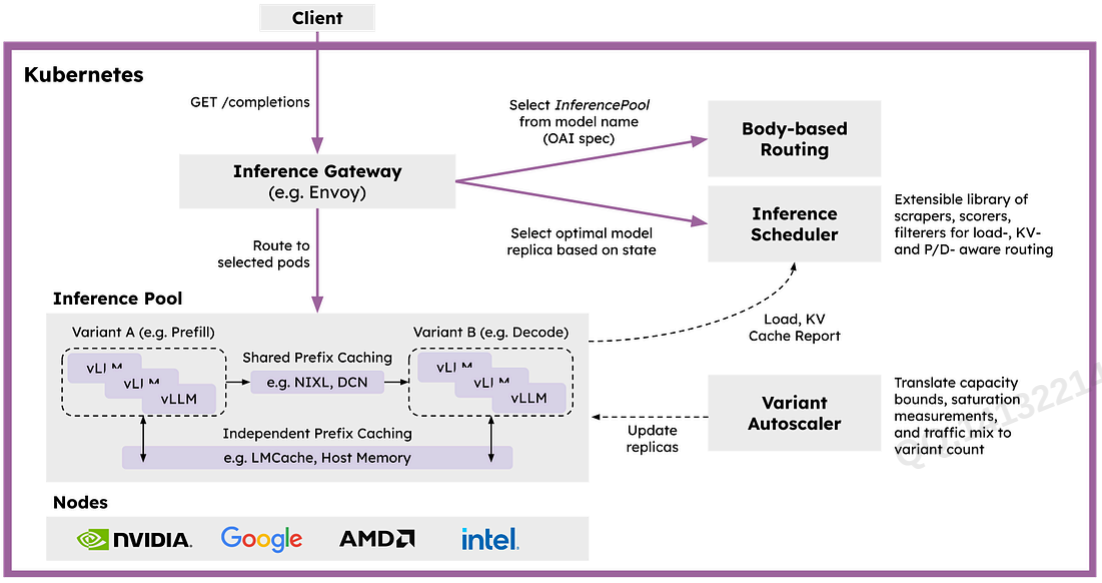
QQ:1413221435

contributes a portion of host DRAM to the external KV-cache pool. vLLM connects to this pool through the MooncakeStoreConnector interface which loads reusable KV blocks into GPU memory and saves newly computed blocks back to host memory. Mooncake Store manages the external cache, including placement and eviction, while Mooncake Transfer Engine performs the actual movement of data between GPU and CPU memory.

Different KV cache managers may use different transfer engines to physically move bytes between memory tiers or machines, such as between prefill and decode workers. Mooncake Store, for example, uses Mooncake Transfer Engine to move KV blocks between GPU memory, host DRAM, and remote nodes.

A deployment can use Mooncake Store to offload reusable KV blocks to host DRAM while simultaneously using NIXL to transfer request-specific KV directly from prefill GPUs to decode GPUs. Mooncake TE handles movement for the Mooncake Store path, while NIXL handles the separate prefill-decode path using UCX and GPUDirect RDMA where supported. Multiple KV-management and transfer paths can therefore coexist within the same inference engine.

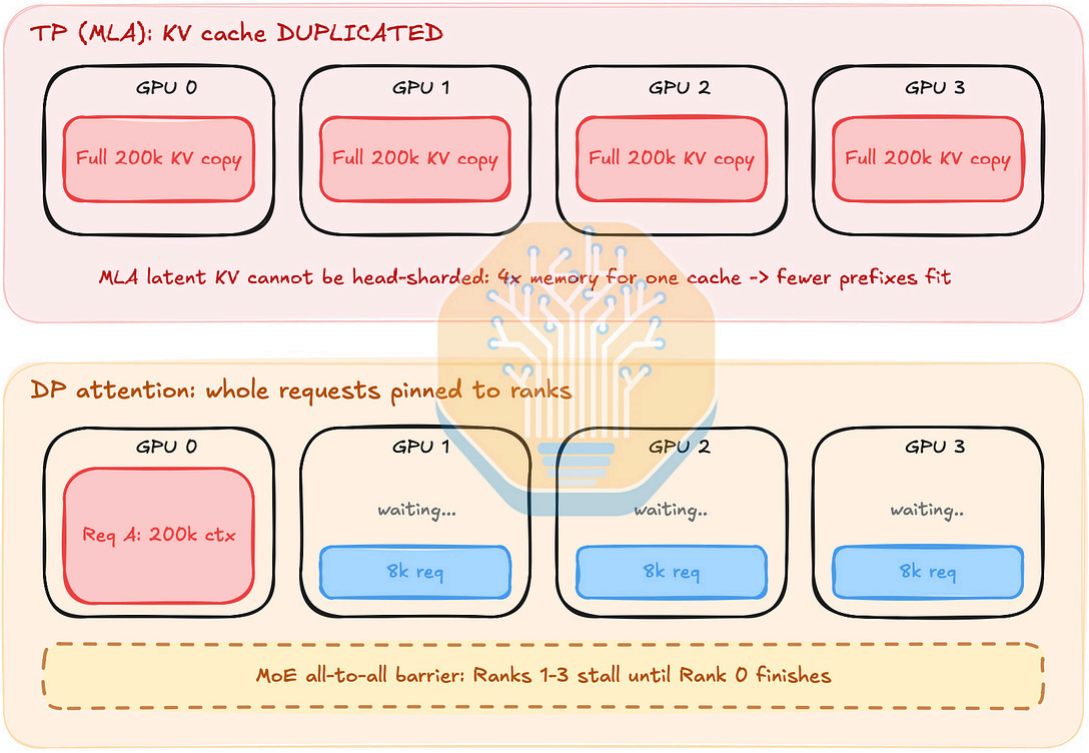
The ecosystem consists of many independent components, including inference engines, routers, KV-cache managers, data-transfer libraries, and cluster controllers. Platforms such as Nvidia Dynamo, llm-d, and AMD Infera “package” selected combinations of these components into complete software distributions. They publish compatible container images, connectors, deployment manifests, and orchestration logic that allow the components to be deployed and operated as one system. The resulting product is usually a collection of coordinated containers rather than a single monolithic service (for instance: Dynamo, llm-d, and Infera are typically deployed on k8s and co-ordinate large, distributed systems).



Source: RedHat / llm-d

Context Parallelism

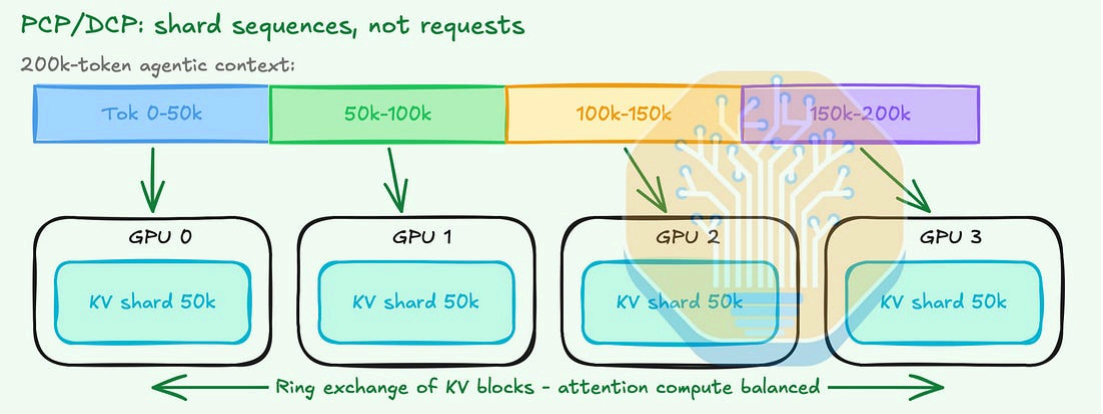
Long context benefits parallelism techniques that a fixed 8k prompt cannot strongly exercise, because at 8k there is little to divide and TTFT is already short. Moreover, parallelism strategies like TP and DP attention are not optimal at longer context lengths, TP can result in the full KV being replicated on each rank. Although KV is shared for DP attention, it can get hung up on longer contexts, as long context workloads also result in a higher variance of possible context lengths.

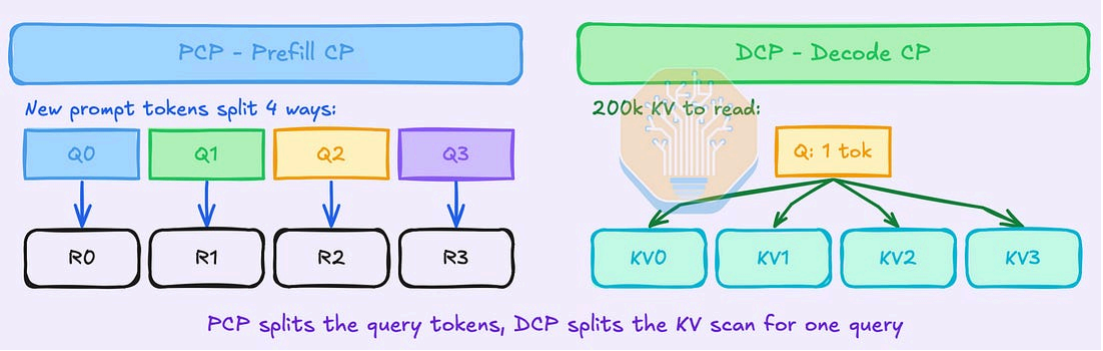


Source: SemiAnalysis

Context parallelism is a parallelism technique that splits query tokens across GPUs. It comes in two forms: PCP for prefill context parallelism and DCP for decode context parallelism. In PCP, each rank prefills its query chunk (KV ring-passed), since prefill tends to be compute-bound, this parallelizes FLOPs resulting in faster prefill with no giant-prompt prefill spike on one rank. For DCP, each rank scans its KV shard, and the partial attention is then merged flash-decode style. Since decode is memory-BW-bound, parallel KV reads can result in faster tok/s.

QQ:1413221435
QQ:1413221435





Source: SemiAnalysis

This parallelism technique was invented partially by Nvidia Research. Nvidia Research is great at fundamental research like this while for end to end training research, they are embarrassing America with their horrible Nemotron3 Ultra model which is currently getting massively beaten by even tiny Qwen3.8 27B model. DCP/PCP forms part of CUDA moat as the AMD implementation of DCP/PCP isn't optimized yet. In the vLLM support matrix, every single AMD backend is unsupported.

A few of the changes mentioned in the next few sections focus on DCP/PCP.

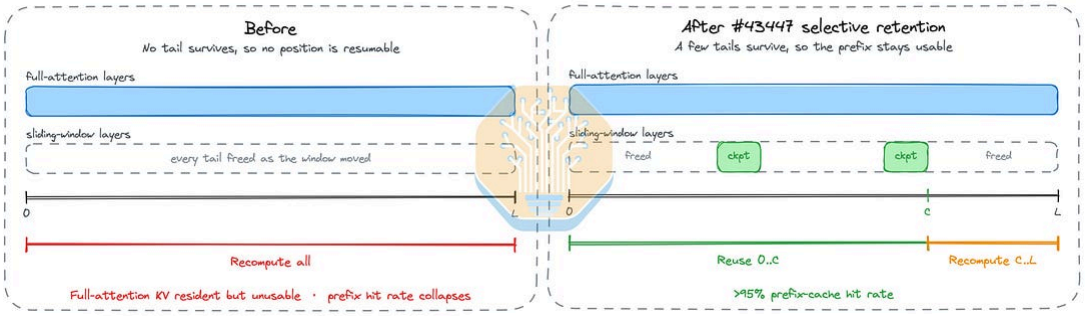
vLLM Agentic Optimizations

Working alongside vLLM maintainers from Inferact, Red Hat, NVIDIA, and AMD, we used AgentX's realistic replayer as a north star, with the resulting fixes landing upstream where most of the optimizations are highly transferable to production. A few examples follow:

vLLM improved hybrid-attention prefix caching so short-lived sliding-window allocations do not evict useful long-context checkpoints. Selective retention preserves sparse replay boundaries and reported a prefix-cache hit rate above 95% with fourteen concurrent requests and contexts up to one million tokens. The same reachability policy was applied to Mooncake, and unreachable sliding-window

QQ:1413221435

lookups were removed. Earlier follow-up work also stopped offloading sliding-window blocks that could never be reused and kept the speculative lookahead block in the retained prefix.



Source: SemiAnalysis and vLLM GitHub PR for Agentic Workloads

Agentic Workloads for high concurrent agents require offloading. Thanks to the aforementioned influence of AgentX, there are already workstreams on vLLM that aim to allow CPU KV offload for hybrid models rather than only for uniform full-attention models. This distinction matters because a uniform model has one KV layout per token, so a connector can describe what to save with a single block geometry. A hybrid model carries several cache groups at once, each with a different shape and a different lifetime, and a connector that assumes one uniform layout cannot express which group a given block belongs to. Offload was therefore unavailable for the models whose long sessions needed it most. The general SimpleCPU connector came first, was enabled on ROCm, and was then extended to DeepSeek-V4 hybrid attention, reporting 81.7% higher output throughput and 46.6% lower mean e2e latency against recomputing the prefix once it no longer fits in HBM.

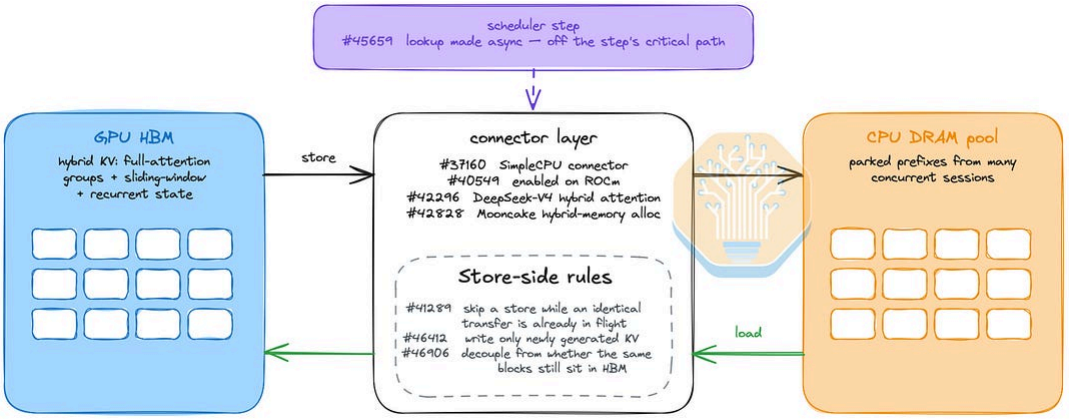
Mooncake has also gained equivalent hybrid-memory allocation support. The same layout problem resurfaces in disaggregated serving, where a pending change transfers Kimi-K3's conv+ssm recurrent state alongside the attention KV over MoRI-IO for 1P1D prefill/decode splits; without it the decode side starts from an uninitialized recurrent state. The recurrent-state slot rides the existing remote-block-ids channel, so the disaggregation router needs no model-specific changes, and the path was exercised end-to-end on MI355X, TP8 per leg over cross-node RDMA, with DSpark speculative decoding on both legs.

When profiling realistic workloads, vLLM maintainers noticed that during offload, the cost moved to the store path, which was writing too much and too often. Three new fixes have now addressed this issue:
A store is now skipped while an identical transfer is already in flight, so concurrent sessions sharing a prefix pay for it once rather than once each. A store covers only

QQ:1413221435
QQ:1413221435

newly generated KV ranges, so a session that extends its history writes the delta instead of rewriting the whole prefix on every turn. Finally, a store no longer depends on whether the same blocks still sit in HBM, so work already scheduled is not discarded when an eviction lands underneath it.

The load path was tuned separately, because lookups happen on every scheduling decision rather than only when data actually moves. Making lookups asynchronous in the scheduler path keeps the connector off the step's critical path, so a step no longer waits on CPU-side cache queries before it can admit work. Compact zero-copy lookup keys, parallel receive-side loading, and prebuilt Mooncake key strings then removed the CPU and transport overhead that remained.



Source: SemiAnalysis and vLLM GitHub PR for Agentic Workloads

Long-lived hybrid state also forced correctness and accounting fixes that fixed-shape requests rarely reach. vLLM emits cache events per hybrid cache group, strides distributed-context stores correctly, and computes lookup prefixes correctly under distributed context and prefill. A related context-parallel accounting change aligns cache ownership with sharded token ranges. Speculative state is now propagated across merged Mooncake groups and through the SimpleCPU coordinator, preventing the repeated-turn cache from silently losing EAGLE state.

Looking at the ROCm side of optimizing Agentic workloads in vLLM, the work continues below the cache layer, where the remaining cost is per-layer rather than per-request. Once the prefix survives and arrives on time, what is left is the decode step itself, and a decode step that runs thousands of times per session pays for every avoidable copy and every mismatched kernel.

Three open changes attack that layer:

- A Kimi-K3 change writes KDA decode results directly into the layer output buffer, removing one device copy per KDA layer; the saving is small in

QQ:1413221435
QQ:1413221435

isolation and repeated for every layer of every decoded token.

- A second change selects an AITER sparse-MLA decode kernel in place of the generic path, and reported 5.22% higher AgentX output throughput with substantially lower inter-token latency.
- The third is a useful illustration of how much the measurement shape matters. A companion change routes full-graph attention projections through tuned AITER GEMMs and reached a 2.3% gain on fixed sequences at low concurrency. This shows a kernel-level change can show a clean gain on uniform shapes and then be swamped, on an agentic trace, by the cache and scheduling variance that the trace introduces. A pending change turns that shape sensitivity into the dispatch criterion itself: it replaces the DeepSeek V4 C4A selector's ROCm top-k bottleneck with a hybrid AITER/native path on gfx950, routing short and medium contexts through AITER and long contexts through a graph-safe tuned native fallback. It reports end-to-end selector speedups of 1.21x to 1.76x, with decode-kernel geomeans of 1.2x to 2.9x across an 84-shape matrix.

QQ:1413221435

SGLang Agentic Optimizations

The AgentX team has been working closely alongside SGLang maintainers from RadixArk, Meta, Nvidia, and AMD to drive optimizations to Agentic workloads run using SGLang, resulting in massive improvements in production inference performance. Let's discuss these optimizations in more depth.

We will start by explaining how, from the allocator side, SGLang's sliding-window work addresses the same conflict vLLM's retention policy does. Window pages and prefix pages are drawn from one pool, and the window is the greedier consumer: it turns over constantly while the prefix sits still and so, under pressure, the transient allocation displaces the durable one.

Three design improvements attack this problem that from different angles. One proactively frees pages as they leave the window rather than waiting for eviction pressure to find them, so dead window state stops competing for pages it can no longer use. Another caps compute locks to a single window, bounding how much of the pool an in-flight request can hold pinned at once. A third removes stale full-KV entries that outlive their usefulness.

Proactive freeing has a fork-shaped blind spot, though: a request branching from a shared prefix can still hold reusable full-KV while the window state at the branch point has already been released, and the whole prefix gets recomputed for want of the

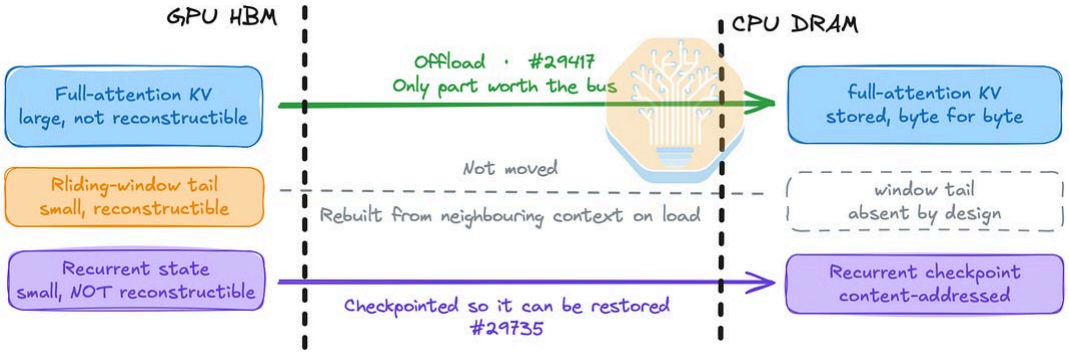
QQ:1413221435

cheap half. Open work preserves the SWA state at those branch points so forks inherit the window instead of rebuilding it.

Alongside those, the ROCm ring-cache fix is a correctness rather than a capacity change: a ring buffer reuses slots by construction, and reusing one whose old contents are still referenced yields wrong output rather than slow output. None of this is visible on a single 8k prompt, where the window never laps the prefix and the pool is never contended. On a multi-turn hybrid session, these changes are what decide whether the expensive full-attention history is still there on the next turn.

HiCache is SGLang's first class in-tree offloading mechanism. It faced the same hybrid problem vLLM's connectors did, and solved it with an asymmetry: offload the full-attention cache and reconstruct the short sliding-window tail on the way back. Only the expensive half is worth moving across the bus, and the cheap half can be rebuilt more quickly than it can be fetched. On AMD, staged write-back keeps that movement from blocking the engine while it happens. Recurrent state was the remaining gap, because it cannot be rebuilt from neighbouring tokens the way a window tail can; FlashInfer GDN checkpoints let it participate in prefix reuse at all, and raised throughput from 47,771 to 53,004 tok/s/GPU at a 92.4% cache-hit rate.

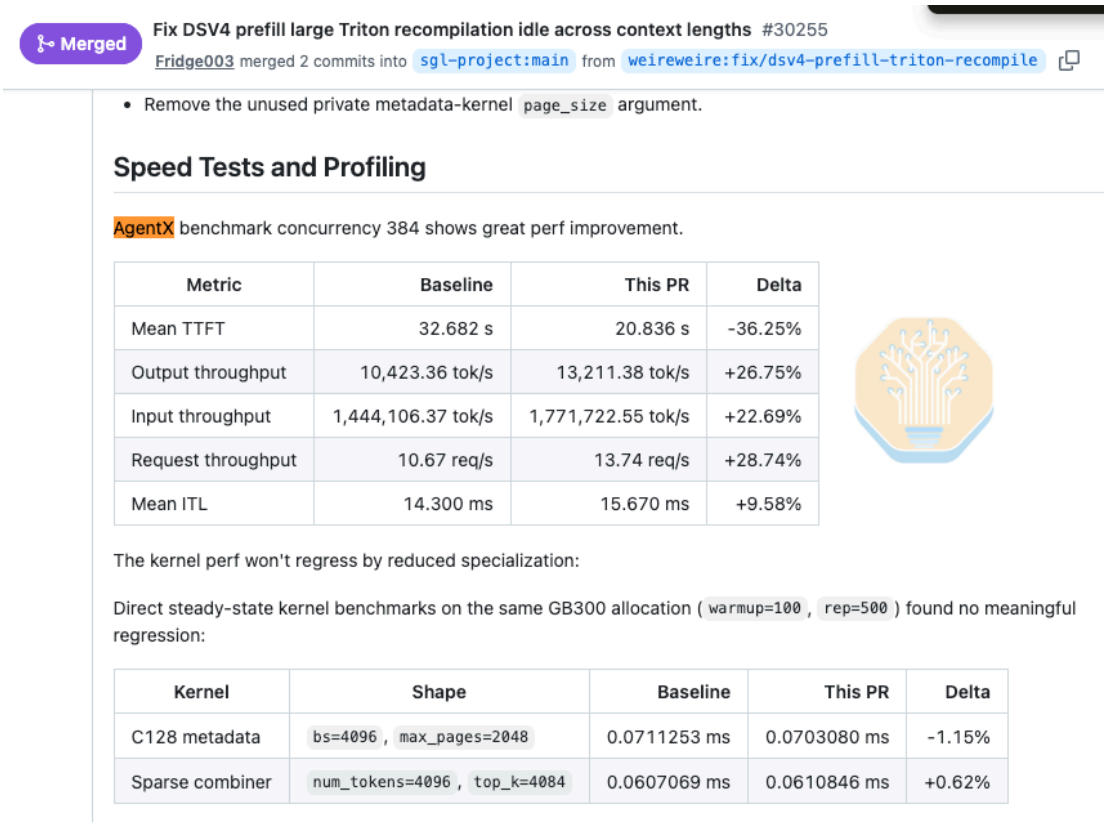
QQ:1413221435



Source: SemiAnalysis

Two further changes address how variable-length traffic affects kernel pipeline. AgentX realistic sessions tend to arrive at continuously varying context lengths, a similar pattern observed for production traffic. A naive runtime that specializes on length will compile a fresh kernel for nearly every request it sees. SGLang maintainers solved this by passing context length as a runtime scalar instead collapsing it into one compilation, improving AgentX concurrency 384 output throughput by 26.75% and mean TTFT by 36.25%, by removing compilation, not from computing anything faster.

QQ:1413221435



Merged **Fix DSV4 prefill large Triton recompilation idle across context lengths** #30255

Fridge003 merged 2 commits into `sgl-project:main` from `weireweire:fix/dsv4-prefill-triton-recompile`

- Remove the unused private metadata-kernel `page_size` argument.

Speed Tests and Profiling

AgentX benchmark concurrency 384 shows great perf improvement.

Metric	Baseline	This PR	Delta
Mean TTFT	32.682 s	20.836 s	-36.25%
Output throughput	10,423.36 tok/s	13,211.38 tok/s	+26.75%
Input throughput	1,444,106.37 tok/s	1,771,722.55 tok/s	+22.69%
Request throughput	10.67 req/s	13.74 req/s	+28.74%
Mean ITL	14.300 ms	15.670 ms	+9.58%

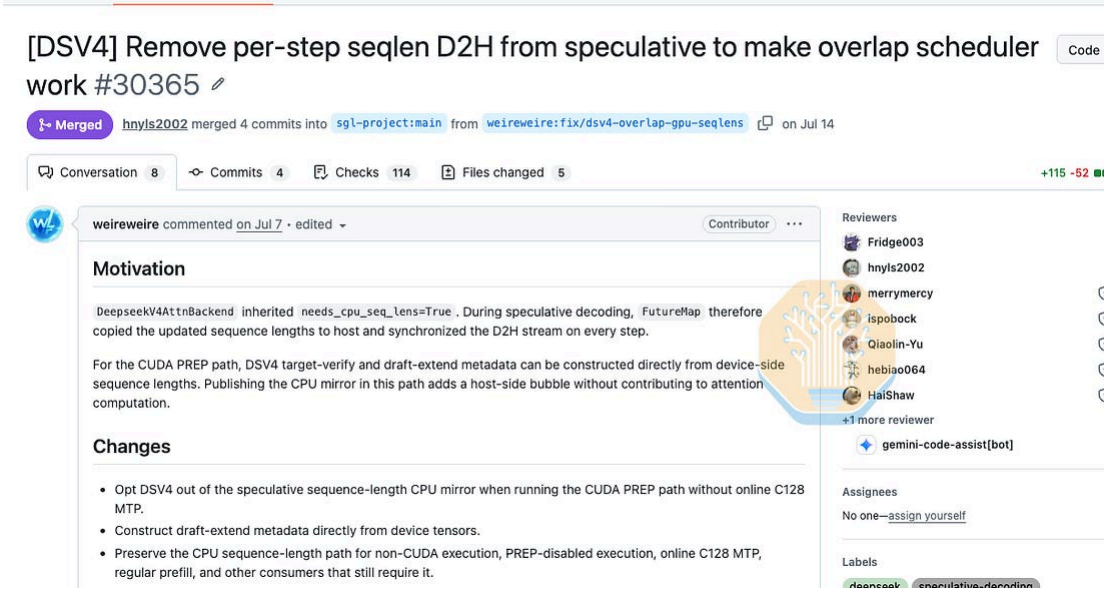
The kernel perf won't regress by reduced specialization:

Direct steady-state kernel benchmarks on the same GB300 allocation (`warmup=100` , `rep=500`) found no meaningful regression:

Kernel	Shape	Baseline	This PR	Delta
C128 metadata	`bs=4096` , `max_pages=2048`	0.0711253 ms	0.0703080 ms	-1.15%
Sparse combiner	`num_tokens=4096` , `top_k=4084`	0.0607069 ms	0.0610846 ms	+0.62%

Source: GitHub

In the same spirit, removing a per-step device-to-host sequence-length synchronization eliminates a decode bubble that exists only because the host wanted to know a length the device already had.



[DSV4] Remove per-step seqlen D2H from speculative to make overlap scheduler work #30365

Merged hnyls2002 merged 4 commits into `sgl-project:main` from `weireweire:fix/dsv4-overlap-gpu-seqlens` on Jul 14

Conversation 8 · Commits 4 · Checks 114 · Files changed 5 +115 -52

weireweire commented on Jul 7 · edited — Contributor

Motivation

`DeepseekV4AttnBackend` inherited `needs_cpu_seq_lens=True` . During speculative decoding, `FutureMap` therefore copied the updated sequence lengths to host and synchronized the D2H stream on every step.

For the CUDA PREP path, DSV4 target-verify and draft-extend metadata can be constructed directly from device-side sequence lengths. Publishing the CPU mirror in this path adds a host-side bubble without contributing to attention computation.

Changes

- Opt DSV4 out of the speculative sequence-length CPU mirror when running the CUDA PREP path without online C128 MTP.
- Construct draft-extend metadata directly from device tensors.
- Preserve the CPU sequence-length path for non-CUDA execution, PREP-disabled execution, online C128 MTP, regular prefill, and other consumers that still require it.

Reviewers: Fridge003, hnyls2002, merrymercy, ispobock, Qiaolin-Yu, hebiao064, HaiShaw, +1 more reviewer, gemini-code-assist[bot]

Assignees: No one—assign yourself

Labels: deepseek, speculative-decoding

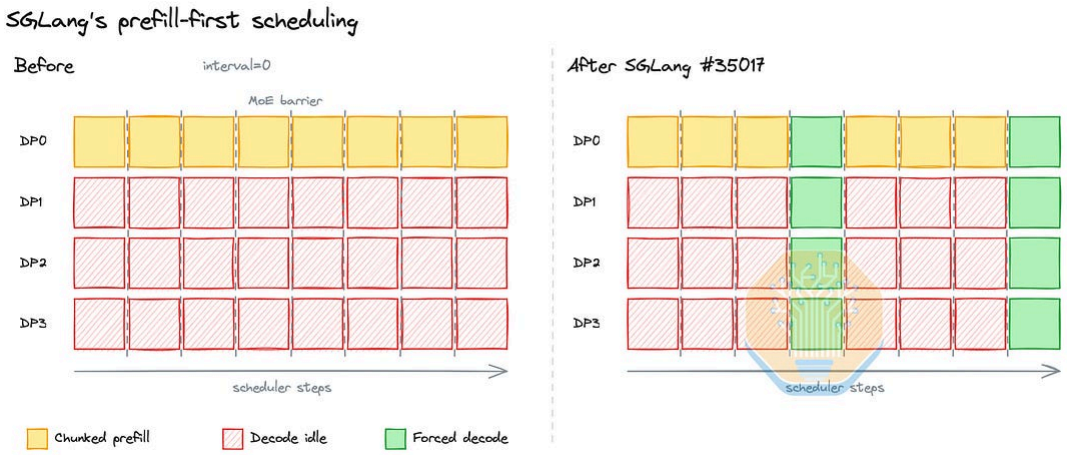
Source: GitHub

Variable length also bites inside the attention kernel itself. On GB300, a mixed-context decode batch pays a tail tax: a matched profile attributed 8.4 ms of a 9.8 ms decode-step delta to attention, with the longest requests dragging out the persistent

QQ:1413221435

kernel's shared wave. Open work splits TRTLLM MHA decode batches into KV-length-sorted groups so the short requests stop waiting on the longest ones.

Decode can starve one level up as well, at the scheduler rather than in a kernel. Under DP attention, every rank joins the same MoE collective while scheduling attention work locally, so a rank fed a stream of chunked-prefill continuations keeps winning the prefill-first decision while peer ranks' running batches sit waiting, observed on AgentX runs. A configurable decode interval after prefill forces decode rounds between prefills; on AgentX DSv4 Pro, output throughput rose 141% and p99 inter-token latency fell 97.3%, at the cost of median TTFT rising from 36.5 to 59 seconds, trading off first-token wait for stream smoothness.



Source: SemiAnalysis

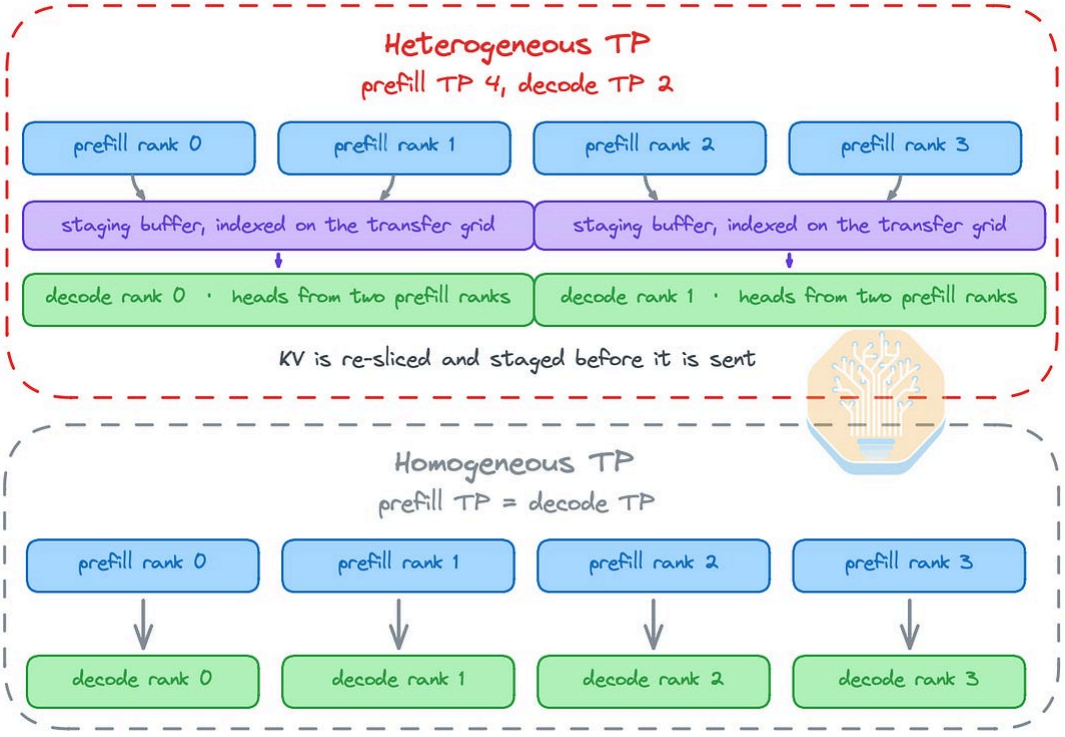
When a request carries no reusable history such as the start of a subagent, any worker will do just fine, and load balancing is the only question worth asking. When the request carries a MB of cached prefix, sending it to an idle worker that does not hold that prefix is the expensive choice, and the router needs to know where the state already lives. SGLang added DP cache affinity, so a session is sticky to the rank holding its cache. In this PR, DP-aware prefill and decode routing is both implemented so that both halves of a disaggregated deployment make that decision consistently, and cache balance as a routing signal so affinity does not degenerate into one hot worker. A router can only act on what it is told, so hybrid cache events also become radix-cache aware and sliding-window aware.

Speculative decoding receives special attention, because MTP adds a second, smaller piece of per-request state that has to survive everything the main cache survives. SGLang fixed draft-window transfer in disaggregated serving so that state crosses the prefill-to-decode boundary intact, added overlap scheduling for high-concurrency online decoding, removed a no-op EAGLE renormalization, and avoided host synchronizations during EAGLE prefill. The open resource-lease scheduling work

and data-parallel graph-metadata fix continue the same effort, which is to make overlap safe when requests can be retracted and resumed rather than simply run to completion.

When looking at Agentic workloads with Heterogeneous prefill and decode topologies, prefix-aware staging is needed, and this is where prefix caching and disaggregation interact badly. When the two sides are not sharded identically, KV cannot be copied across as one contiguous stream; it has to be split on a transfer grid and reassembled at the offsets the decode side expects. A prefix hit makes that harder, not easier, because the prefill worker now sends only the uncached remainder while the decode side still expects a complete, correctly positioned cache. Radix-cache support in the staging buffer splits cached sends on that grid and scatters them at the correct decode offsets.

QQ:1413221435

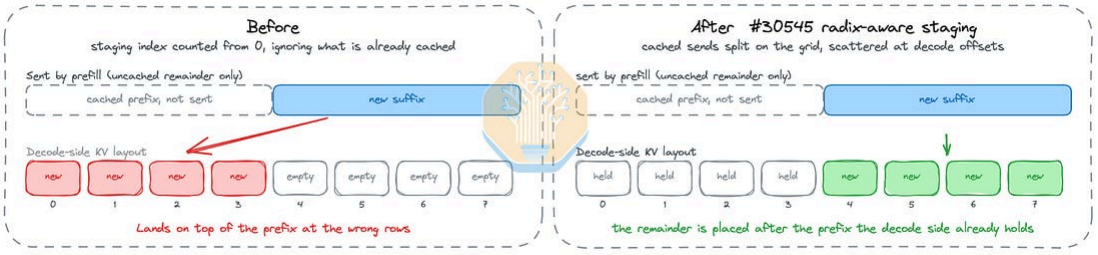


Source: SemiAnalysis

However, this results in correctness problems. A 127,500-token shared-prefix test went from 2 correct needles out of 128 to 128 out of 128, meaning the cache had been silently landing in the wrong places, which a throughput benchmark would have scored as a fast, confident, but wrong answer. The AgentX comparison additionally raised median per-user output throughput by 9.6% at nearly unchanged total throughput per GPU. The transfer itself also carried dead weight: decode-side PREBUILT batches never enter a model forward, yet every transferred prompt was still flattened and copied into a CUDA input tensor that the first decode step reconstructs from relay metadata anyway. Dropping that unused prompt transfer is

QQ:1413221435

the largest single win, driving +18.0% per-user output throughput and +12.7% decode throughput per GPU on AgentX GB300. A follow-up moved the prefill DP-rank bootstrap query off the decode scheduler's critical path, overlapping an HTTP round trip that had been paid synchronously at result consumption, driving a further +1.36% per-user output throughput on the same deployment. Open work continues along the same seam: multi-pool DeepSeek-V4 support in UMBP, unified-KV HiSparse state carried over MoRI, and preserving the prefill-owned token when decode terminates without visible content. The HiSparse work should be read as a capacity and correctness enabler for long contexts rather than as a throughput win at high concurrency, which it is not yet.



Source: SemiAnalysis

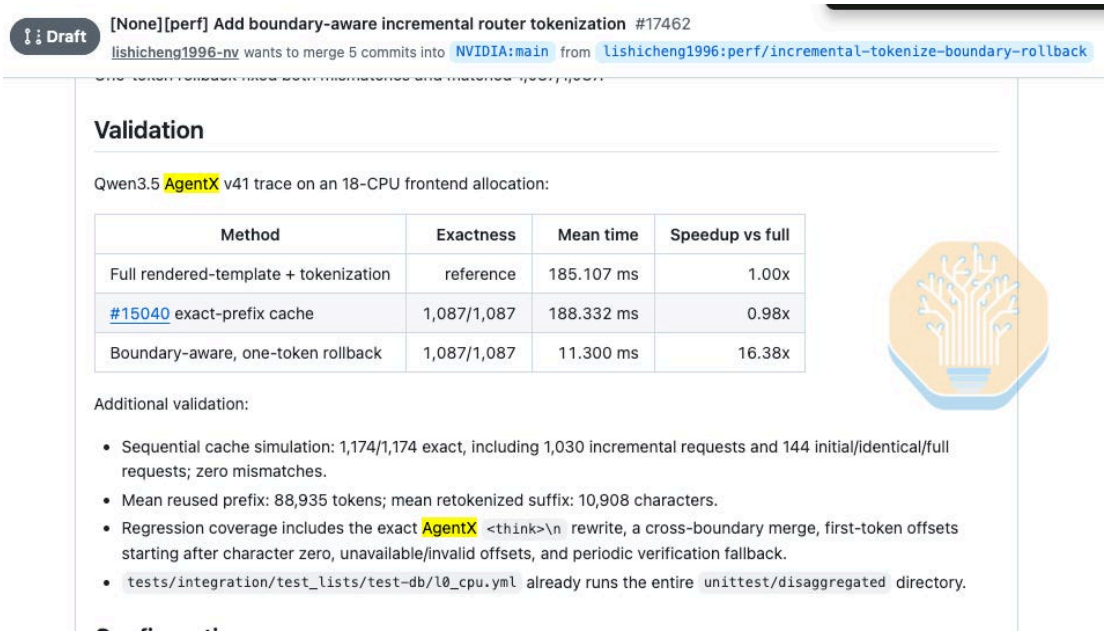
TensorRT-LLM Agentic Optimizations

Next, we shall move onto explaining a couple of the recent optimizations from TensorRT-LLM. First, we look at TRTLLM's distinct frontend optimization for repeated chat turns, a cost that only exists because the workload is multiturn. Every turn of a conversation re-sends the entire history plus a little more, and the naive implementation re-tokenizes all of it. Tokenization is cheap per kilobyte, but is ruinous when the same 100,000 tokens are tokenized again on every turn.

The obvious fix, to tokenize only the new suffix, is wrong in a way that is easy to miss, because byte-pair encoding is not position-independent. Tokens can merge across the join, so splitting the text at the boundary and concatenating the two token sequences can produce a different sequence than tokenizing the whole string, which quietly diverges from the sequence the prefix cache was built against.

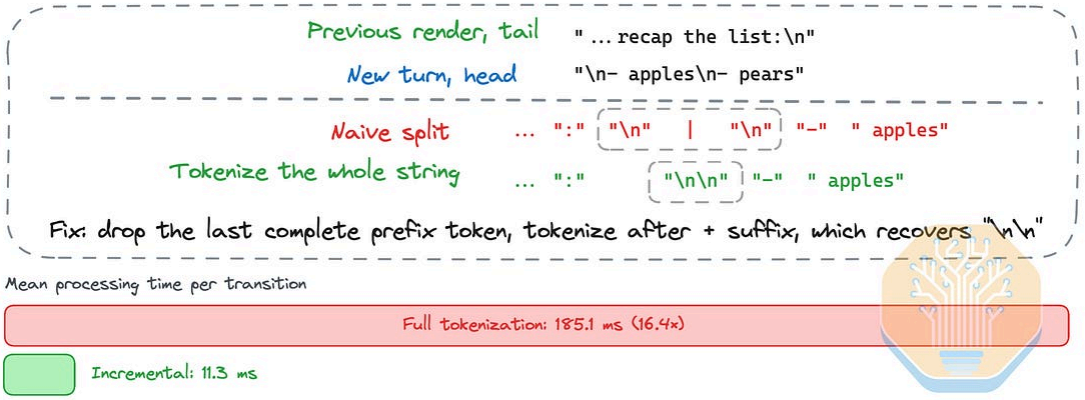
TRTLLM implements Boundary-aware incremental tokenization which handles this by finding the rendered-text common prefix, rolling back one complete token so any merge that spans the join is recomputed, and tokenizing only the changed suffix from there. On the Qwen3.5 AgentX trace, it matched full tokenization on all 1,087 transitions — the correctness claim, tested rather than assumed — and reduced mean processing time from 185.1 ms to 11.3 ms.

QQ:1413221435



Source: GitHub

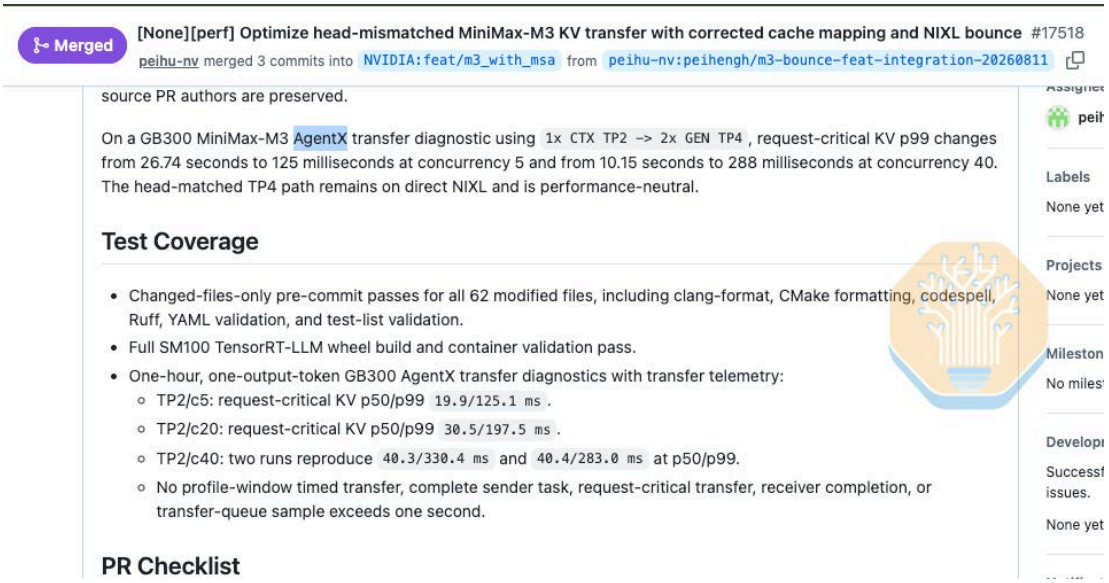
A fixed 8k1k request has no prior rendered turn to reuse, so none of this appears there. Relatedly, chat-template rendering was moved into the input-processing pool, so a long template no longer serializes the main request loop behind it.



Source: SemiAnalysis

The MiniMax-M3 work focuses on disaggregated KV movement, where the failure is one of granularity. When prefill and decode do not agree on head layout, the KV for one logical request stops being a few large contiguous regions and becomes thousands of small strided pieces, each of which turns into its own transfer descriptor. The bytes moved are unchanged; the per-descriptor overhead is what explodes, and it explodes in the worst way on exactly the long prompts that matter! Corrected multi-pool mapping and a chunked NIXL bounce path coalesce those pieces through a bounded reusable arena, trading an extra staging copy for orders of magnitude fewer descriptors. Its AgentX diagnostic reduced request-critical KV p99 from 26.74 seconds to 125 ms at concurrency five, and from 10.15 seconds to 288 ms at concurrency forty.

QQ:1413221435



Source: GitHub

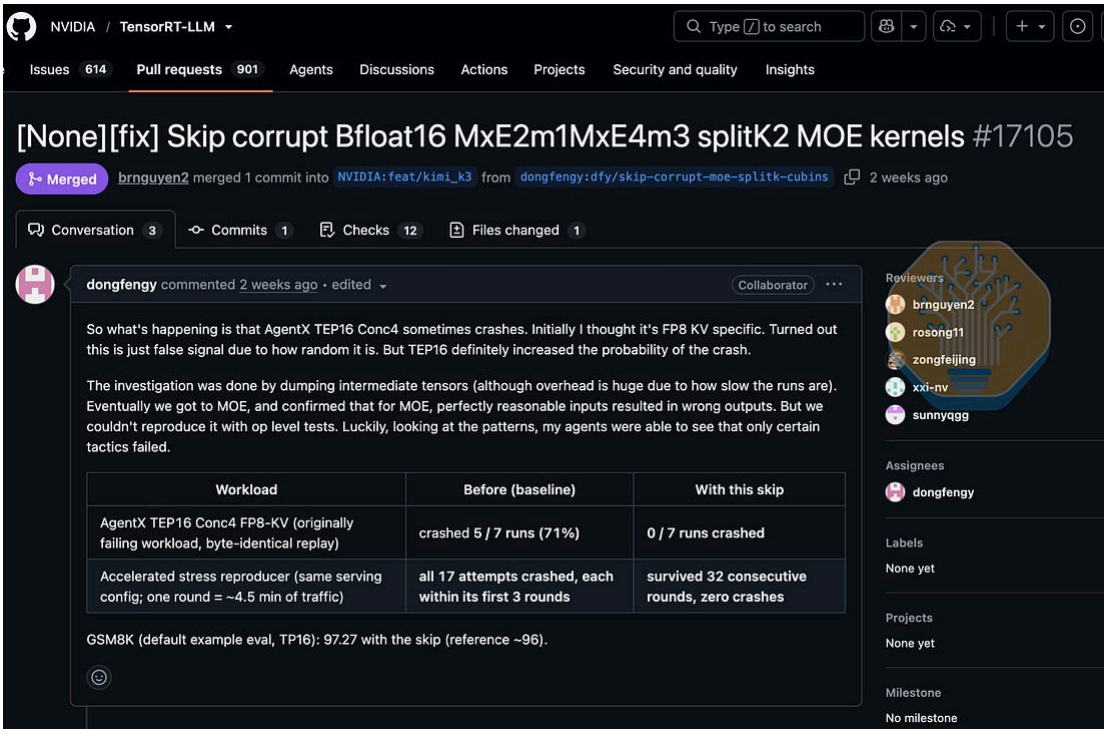
Nonblocking context-transfer polling protects the same path by reaping completed transfers even when scheduling stalls. This breaks a feedback loop in which finished KV blocks stay pinned and prevent new admissions. The stalls themselves also had a removable cause: avoiding implicit device-scalar syncs in DeepSeek-V4's context sparse-attention metadata eliminated 18 four-byte device reads per step that each forced a cudaStreamSynchronize, on a GB300 disaggregated context worker. The fix threads host-side counts through as plain Python ints, so the executor thread no longer holds the GIL for most of a step while the KV-transfer and response threads wait behind it.

TensorRT-LLM also moved irregular long-context work onto more efficient execution paths. Context graph producers for MiniMax-M3 capture stable sparse producers while leaving request-dependent attention eager, and per-user output throughput improved by 12.58 percent in its AgentX test. An open native KV-event production change reduces allocation and conversion work on the KV-aware routing path.

AgentX also exposed kernel-selection and scheduler-lifetime failures that only appear at scale and duration. Two are about which kernel gets picked. MiniMax-M3 added CuTeDSL choices to MXFP8 autotuning, widening the candidate set and improving output throughput per GPU by roughly 7 to 10% at low-concurrency aggregate points. In the opposite direction, TensorRT-LLM disabled corrupt split-K MoE tactics after it crashed five of seven AgentX runs, with no crashes in seven matched runs afterwards. A tactic that is fast and wrong is worse than one that is merely slow, and an autotuner will select it enthusiastically unless it is removed from the pool. Selection is not the only way a kernel goes wrong: MiniMax-M3's legacy sparse-attention path for short queries could hand the SM100 kernel a non-contiguous, head-

QQ:1413221435

major block-index view that it read as contiguous, selecting the wrong KV pages and producing incorrect or non-finite output. Honoring the block-index strides for q_len ≤ 32 fixed the indexing without materializing the tensor or adding a kernel, and five matched full AgentX pairs on GB300 afterwards completed with zero serving errors and no non-finite markers.



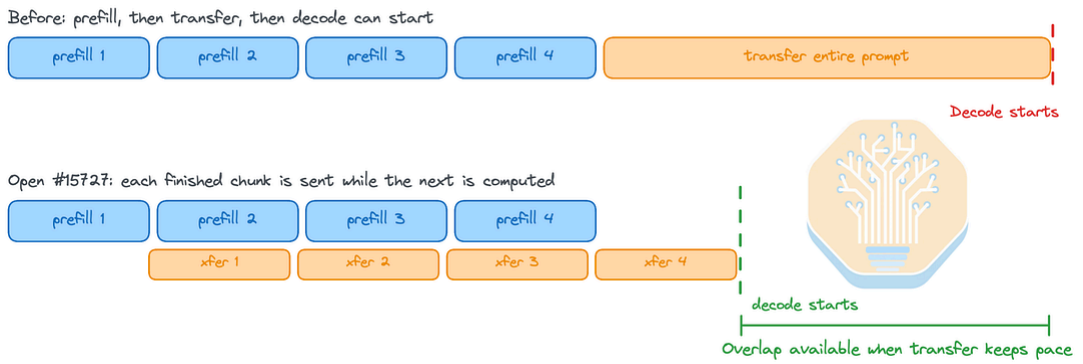
Source: TRT Github

The other two are lifetime bugs, which are the characteristic failure of long runs rather than large ones. Sequence-slot headroom and consistent slot-indexed buffer sizing handle the transient overlap where a completing request and a newly admitted one both need a slot, a window that a steady stream of arrivals and departures hits constantly and a fixed batch never hits at all. A later attention-data-parallel dummy-request fix kept nine Qwen3.5 disaggregated cells alive where most earlier cells had failed within minutes, the difference between a configuration that benchmarks and one that survives a session.

Two open transfer changes target very long disaggregated prompts, and together they show how a fix can create the next bottleneck. In the default arrangement, a decode worker cannot start until the entire prompt has been prefilled and then transferred, so two expensive phases run back to back even though the first produces its output incrementally. Pipelined KV transfer begins sending each completed prefill chunk as it lands, so transfer overlaps prefill compute and only the final chunk is on the critical path.

QQ:1413221435
QQ:1413221435

That change makes chunk handling frequent, which exposes work that used to happen once. Its follow-up retrieves only the block IDs belonging to the current chunk rather than the whole prompt's block list each time. For a 128,000-token prompt split into 1,024-token chunks, that is the difference between building a 4,096-entry list once and rebuilding it 128 times for every layer group. A per-chunk cost that scales with total prompt length is a cost scaling shape that eats the gain the pipelining just bought.

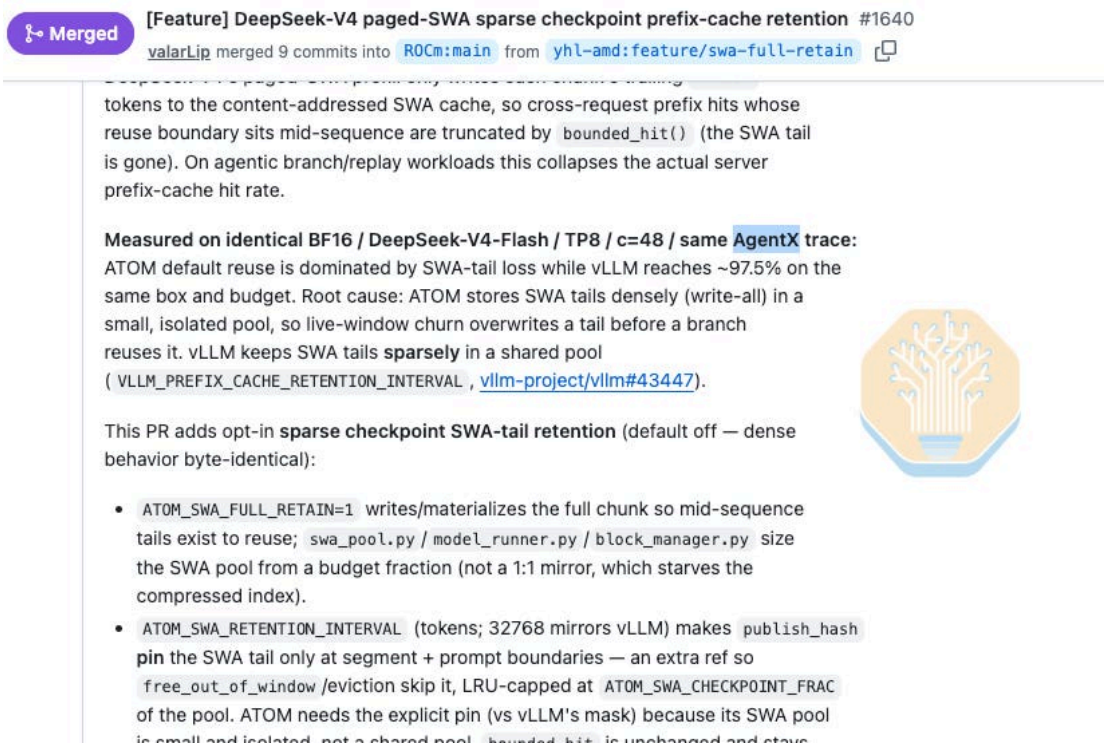


Source: SemiAnalysis

AMD ATOM Agentic Optimizations

AMD's ATOM engine was originally designed only for single turn workloads instead of real world agentic multi-turn production workloads, so there were a lot of changes needed to the core fundamental ATOM engine and kernels to enable good support for long context multi turn workloads. ATOM still has a long way to go to support agentic workloads relative to where vLLM/SGLang are at present. AgentX is used as the realistic north star target for ATOM's refactor to support agentic workloads. The first optimization we will chat about that ATOM implemented is smartly using sparse checkpoint retention for DeepSeek-V4 paged sliding-window attention. The merged implementation keeps selected window tails alive so branch and replay requests can resume at useful boundaries. Its measurements separate the two effects cleanly: on the same AgentX trace at concurrency 48, the actual prefix hit rate rose from 5.6% to 96.45%, and losses at the sliding-window gate fell from 91.35% to 0.16%. The second number is the mechanism behind the first. Nine out of ten prefix matches were being found and then discarded for want of a window tail, so the cache was not missing but being overruled.

QQ:1413221435
QQ:1413221435



Source: GitHub

Two earlier cache-manager fixes had to land before any of this could be measured, and both fixes are worth noting as examples of a cache that reports itself healthy while doing nothing. One stopped free-pool hits from destroying shared cache entries; the other, a deferred-output fix, restored prefix hashing in the default scheduler mode and moved repeated long prompts from zero cached tokens to reuse of every complete prefix block. A separate change lets prefix-hit prefill stay on the optimized sink attention kernel rather than falling back to the generic path, so a cache hit does not quietly cost part of what it saves.

Hybrid models also carry a recurrent or compressor state, which differs from ordinary KV in one decisive way: it cannot be reconstructed from the tokens around it. A window tail can be recomputed from neighboring context, but recurrent state is the accumulated result of everything that came before, so if it is dropped, the only way back is to replay the sequence. ATOM gave this per-request state a content-addressed checkpoint lifecycle, letting generated turns leave reusable resume points without reserving a separate protected cache for them. In one test, a request reused 512 generated tokens and computed only a two-token suffix.

The tuning detail matters as much as the feature. Publishing a checkpoint unconditionally costs 17.5% throughput on zero-hit traffic, the price paid by every session that never comes back, in order to help the ones that do. Spacing checkpoints by token interval avoided that penalty, and fixed 1k1k throughput stayed within measurement noise, which is the relevant safety property: a feature aimed at agentic reuse should not tax workloads that will never use it.

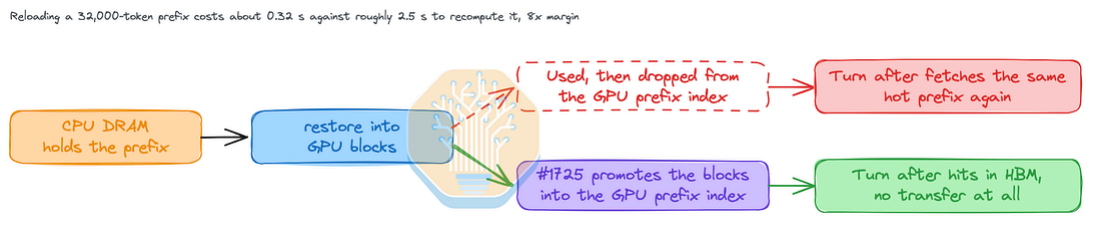
QQ:1413221435

ATOM's AgentX-relevant CPU path starts from the arithmetic that justifies offloading at all. Standalone LMCache offload reloads a 32,000-token prefix from CPU in about 0.32 seconds against roughly 2.5 seconds to recompute it, an eight-fold margin. This makes crossing the bus worth doing at these context lengths and would not hold for a short prompt.

The rest of the path is about ownership and index placement rather than bandwidth. ATOM copied vLLM's multi-connector design which lets a prefill worker send KV to a remote decode worker and save the same prefix to CPU at once, without freeing the blocks until both consumers are finished; two independent readers of the same blocks is a situation where single-turn won't get.

Promoting restored blocks back into the GPU prefix index fixes a subtler waste: without it, a prefix loaded from CPU is used and then not registered as resident, so the next turn fetches the same hot prefix across the bus again, paying the transfer repeatedly for a cache that was already in HBM. Follow-up work fixed asynchronous save ordering, packed-KV geometry, unaligned handoffs, and remote request accounting together, eliminating reload corruption across a two-round, 2,638-request validation. This bug surfaces only when the same blocks are saved, evicted, and restored many times over.

QQ:1413221435



Source: SemiAnalysis

The distributed path repeats, in a different codebase, and the pattern is already visible in SGLang and Dynamo: routing has to know where state lives. ATOM's router called ATOMesh is a fork of SGLang's router with most of the features removed. Unfortunately, ATOMesh required SGLang's cache-aware routing feature, so that feature had to be added back. ATOM gained KV lifecycle events for cache-aware routers, so the router can know where state lives at all. It also gained multi-node prefill and decode routing, and session-sticky data-parallel routing. The sticky policy is a two-sided compromise worth stating explicitly: a conversation returns to the healthy worker that owns its state, but idle assignments expire so that stickiness does not permanently unbalance the cluster on behalf of sessions that have gone away.

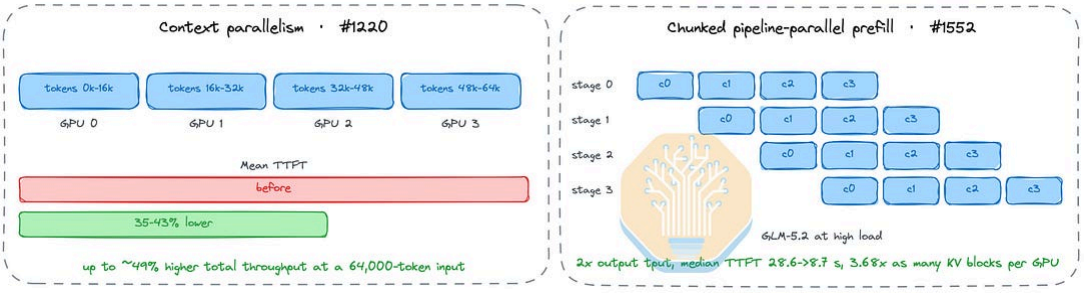
QQ:1413221435

Disaggregation then has to move whatever the model actually keeps, which is not always one uniform cache. DeepSeek-V4 transfers both buffers of its mixed FP8 and

BF16 cache layout, and EAGLE disaggregation moves the draft model's independent KV cache alongside the target cache, the same second-cache problem TensorRT-LLM and SGLang each had to solve. Remote-KV admission and backpressure closes the loop by stopping the decode side from accepting more parked transfers than it can safely resume, which is the disaggregated form of accepting work you cannot finish.

On ATOM, PCP reported 35 to 43% lower mean TTFT, with total throughput gains of up to about 49% at a 64,000-token input - a gain that grows with input length rather than with batch size. Making that usable in practice required it to compose with everything else a session relies on, so DCP was made compatible with prefix caching, chunked prefill, and FP8 KV and then extended to MTP. Parallelism that cannot coexist with the prefix cache would trade one long-context win for another. The same scarcity of parallelism exists inside a single GPU: a batch-1 MLA decode has no head or query dimension to spread, only the KV walk, and a hardcoded split budget of 16 left that walk running on 16 of a gfx950's 256 CUs. A still-open change stops overriding the kernel's own split derivation, so Aiter cuts the walk into as many parts as the machine has clusters.

Chunked pipeline-parallel prefill attacks the same problem from the memory side, replacing repeated tensor-parallel collectives with streamed layer-stage handoffs. Its GLM-5.2 result at high load is the most complete in this section: output throughput doubled, median time to first token fell from 28.6 seconds to 8.7 seconds, and each prefill GPU held 3.68 times as many KV blocks. That last figure is the one to read first, because capacity per prefill GPU is what decides how many long sessions can be in flight before the deployment hits the HBM cliff at all.



Source: SemiAnalysis

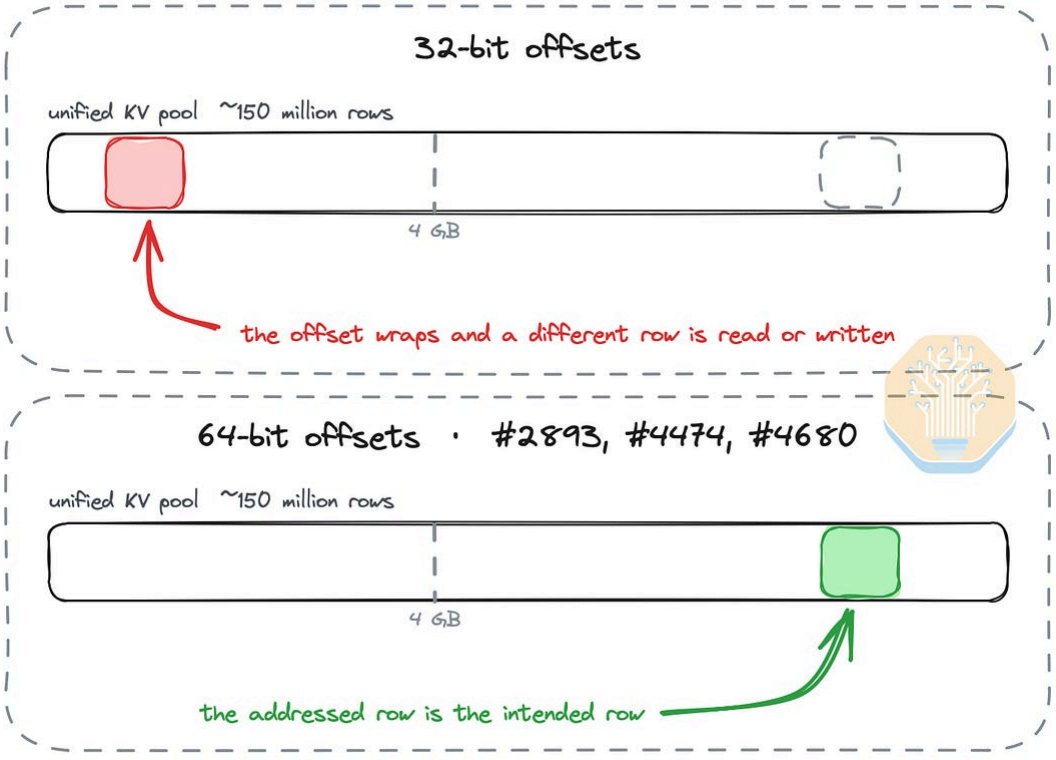
ROCm AITER Agentic Optimizations

ATOM/AMD vLLM/AMD SGLang's long-context execution depends on matching lower-level AITER kernels, because a parallelism strategy at the engine layer is only real if the kernels can express it. Prefill context-parallel process groups provide the extra query-sharding dimension that prefill context parallelism needs, and also widen

QQ:1413221435

fused-kernel row indexing for prompts above 131,000 tokens. Decode context parallelism (DCP) shards KV across the tensor-parallel GPUs already present, so a longer sequence or a larger batch fits without replicating the whole cache on every rank.

Large caches also exposed a class of failure that short fixed requests essentially never reach: address width. A 32-bit offset is entirely adequate until a single cache pool crosses the boundary, at which point the arithmetic wraps and the kernel addresses the wrong row without any error being raised. AITER added runtime 64-bit dispatch for batch prefill above 4 GB, 64-bit MLA offsets above 2 GB, and 64-bit addressing throughout DeepSeek-V4's unified cache paths, the last preventing silent reads and writes to the wrong row in pools of roughly 150 million rows.



Source: SemiAnalysis

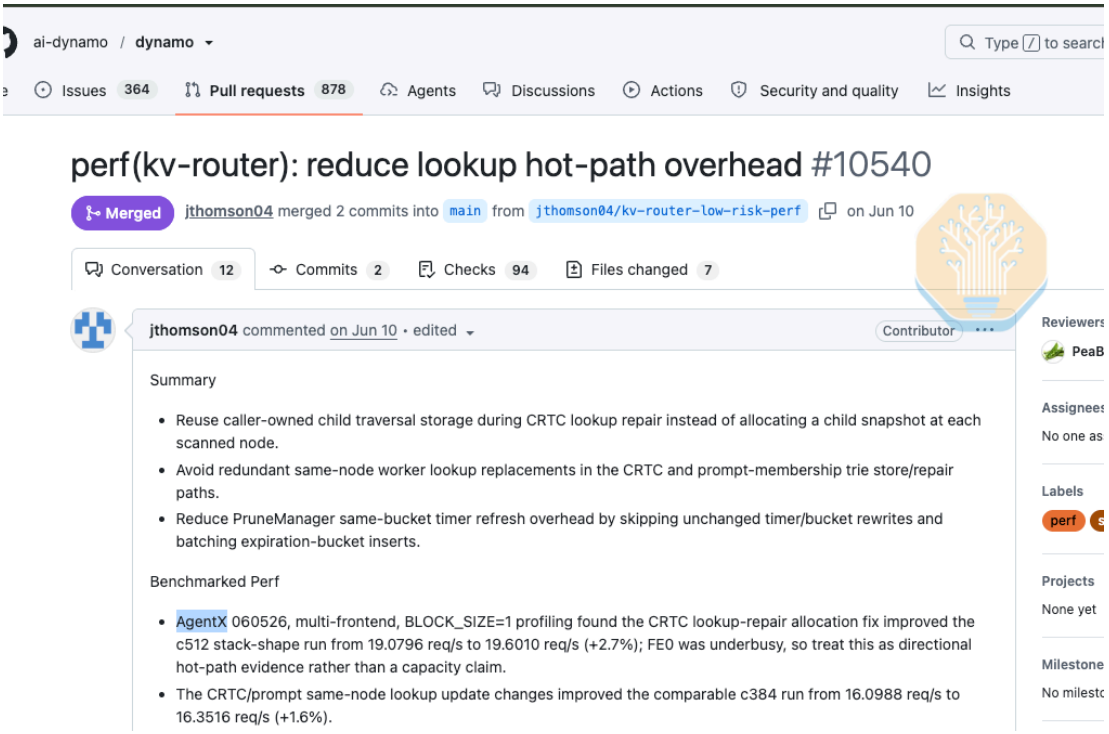
DeepSeek-V4 decode also gained a persistent MLA kernel for 64-head and 128-head MTP packings. Those two head counts are what ordinary decoding and speculative verification actually produce, so this gives the engine a dedicated long-context path for its common shapes instead of treating them as incidental variants of a kernel written for short contexts. It is the same argument as the vLLM AITER sparse-MLA selection above: in long context, the generic path is not a modest compromise, it is the wrong kernel.

QQ:1413221435
QQ:1413221435

Dynamo Agentic Workload Optimizations

A good chunk of Nvidia submissions use the Dynamo Inference Orchestration and Router Systems. Dynamo's AgentX series shows that the distributed serving layer can become the bottleneck once engine kernels improve. The router's work is proportional to the number and length of live prefixes rather than to the number of tokens generated, so a workload of many long, overlapping, long-lived sessions loads it in a way that fixed-shape traffic never does. The first series of PRs reduced the cost of each routing decision: less work on the lookup hot path, no redundant suffix invalidation, and finally batched KV matching, registration, ownership, and terminal dereferences, which reported a 22.2% median output-throughput gain at concurrency 512. Batching helps here for the same reason it helps in an engine: the per-item overhead was dominating the item.

QQ:1413221435

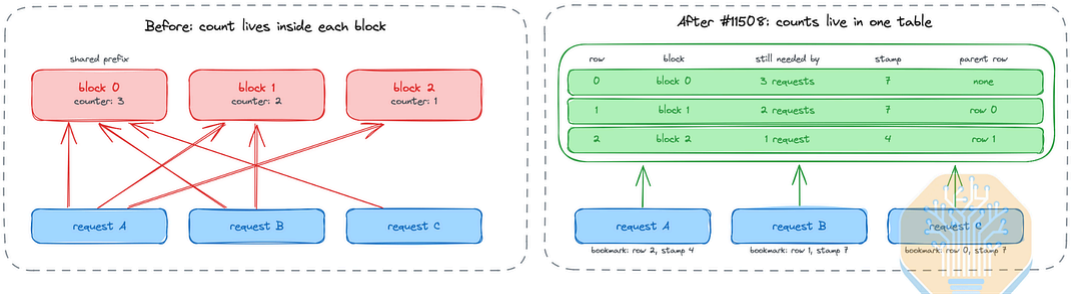


Source: GitHub

The second series of PRs changed how ownership is represented, which is the harder problem underneath. Every cached block needs to be attributed to the requests relying on it, so it is not freed while still in use and not pinned after everyone has finished. With thousands of concurrent sessions sharing overlapping prefixes, the bookkeeping itself becomes significant. Dynamo moved from shared block chains to arena-level ownership counts and finally to backend-specific request leases, each step coarsening the unit being tracked. The lease design reduced AgentX replay time by 23.7% for the

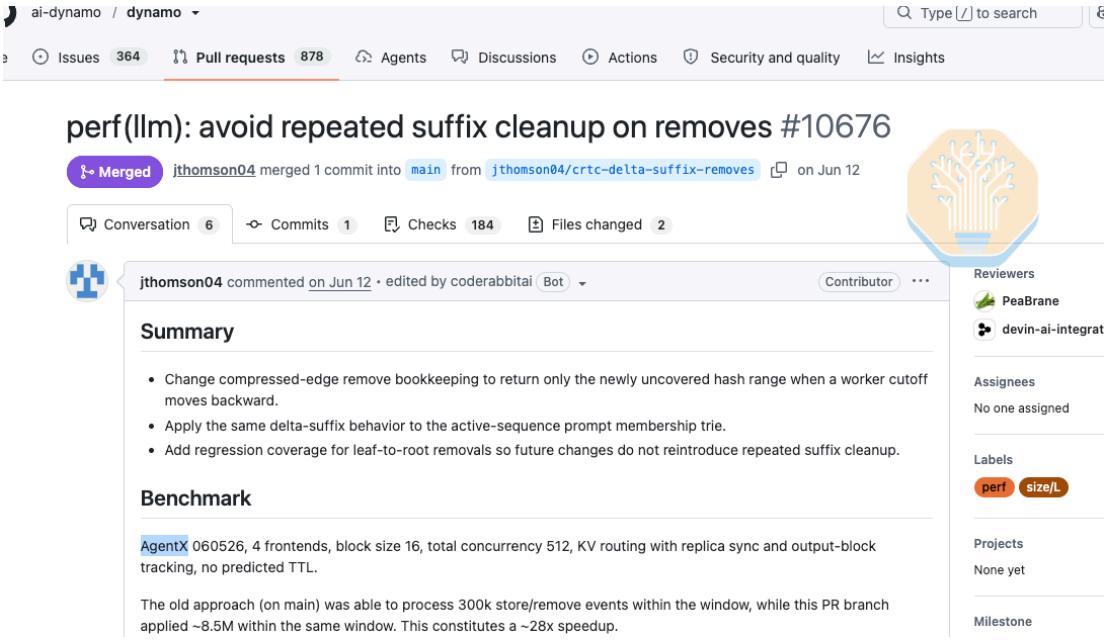
QQ:1413221435

vLLM backend and 22.0% for SGLang, and lowered peak memory at the same time, a sign that the previous representation was the problem rather than the traffic.



Source: SemiAnalysis

Further router profiles removed costs with the same shape, where a periodic sweep or a full recomputation had been acceptable only because live state used to be small. Bucketed expiry pruning replaced a scan proportional to everything tracked and improved high-churn AgentX throughput by 13.7%. Delta-only suffix cleanup processes only what changed and absorbed about 28 times as many store and remove events in the same window. Compressed prompt paths cut front-end CPU by 35.3% and materially improved tail time to first token, which matters because prompts in this workload are long and largely repeated. Overload state is now tracked incrementally rather than recomputed.



One routing change is a deliberate trade rather than a pure win. Dynamo can now charge active decode requests in its routing score, so a worker already committed to long-running decodes looks more expensive than its queue depth alone suggests. That improved median AgentX latency at a small throughput cost in the reported tuning point, which is the kind of choice that only becomes visible when requests occupy a

QQ:1413221435

worker for a long time. An open follow-up packages that trade into an new agentic router preset that pushes further in the same direction, crediting prefix overlap at 2, scaling prefill load by 4, and weighting active decode requests at 64. At that tuning point, the trade stops costing throughput: on an 8xH200 AgentX run, the preset improved fixed-window completed-output throughput by 8.26% over the default cost function, cut run-level p95 time to first token by 43.1% and p95 inter-token latency by 22.6%, and completed one more full trajectory.

The request plane was optimized next, because an agentic trace does not send one request and one response. It sends many related requests carrying largely identical prompts, and streams every token back as its own frame, so serialization and copying are paid per turn and per token rather than once. Switching to MessagePack request payloads improved throughput by 8.1% and reduced average time to first token by 9.7% in its AgentX test, and direct Python transcoding removed an intermediate value tree from that path entirely.

QQ:1413221435

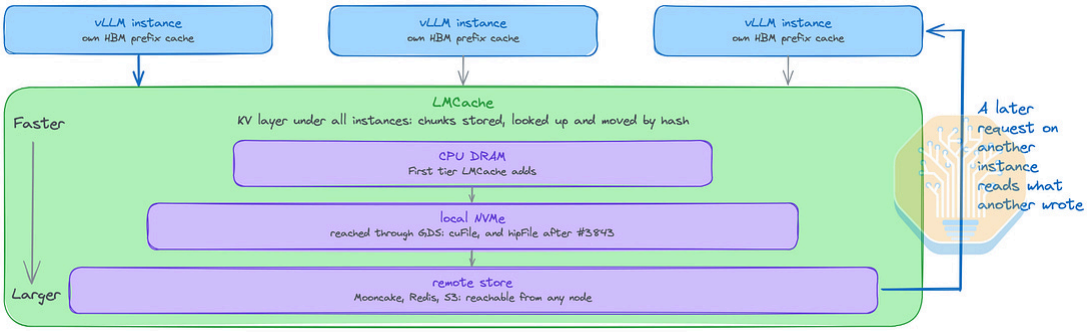
What followed is a sequence of changes that all remove a copy rather than speed one up: not copying MessagePack event payloads, not copying received ZeroMQ frames, and not paying full inter-token-latency metrics overhead on every token. The chat streaming hot path was shortened for the same reason. Individually, these are unremarkable; multiplied by every streamed token of every concurrent session, they are what determines how many requests per second a frontend can sustain.

High-concurrency profiling then found costs that had nothing to do with moving data. Static logging filters removed a shared span-matcher lock, a contention point rather than a volume problem, and raised reported frontend throughput from 932 to 1,133 requests per second. Simpler positional radix buckets reduced peak memory in the mocker by 5.51 GiB in a 32-worker run. An open change flushes detokenization metrics once per response rather than updating cumulative counters on every streamed chunk, approximately halving frontend CPU time in its matched diagnostic profile. That last one is the clearest example of the category: the instrumentation was cheap per call and ruinous at one call per token.

LMCache Agentic Optimizations

LMCache is an open-source KV cache layer that sits under inference engines like vLLM, storing reusable KV chunks keyed by prefix hash across CPU DRAM, local NVMe, and remote backends (Mooncake, Redis, S3). LMCache can be used as an alternative to vLLM’s native offloading connectors.

QQ:1413221435



Source: SemiAnalysis

LMCache's multiprocess path was changed for the volume and shape of agentic cache movement, beginning with a failure that is not a slowdown but a stop. When each of many requests with contexts above 100,000 tokens reserves the blocks for its whole load before starting, the pool is exhausted by requests that are all waiting and none progressing. Chunked external-cache loading reserves per chunk instead, so loads interleave and drain. At concurrency 32, the validation completed 120 requests where the old path deadlocked after 28, and concurrency 48 kept running with the KV pool 98.5 percent full.

The other changes reduce how much is moved and how often the runtime gets in the way. Storing only the useful portions of DeepSeek-V4's hybrid groups cut storage per token by almost twenty times, and sliding-window prefetch now loads only the live window rather than window state that will never be read, the same reachability argument vLLM applied to offload, approached from the storage side. One native transfer call per object group then removes repeated Python lock handoffs across staging copies and kernel launches, which is overhead proportional to the number of pieces rather than to the bytes in them.

Two current LMCache changes are especially specific to AgentX but remain open. The hybrid lock-accounting fix stops one request from releasing another request's read locks on shared sliding-window or recurrent-state chunks. Several requests must share the same chunks, the accounting must be per-chunk rather than per-holder, and eviction must actually start. Sustained Kimi-K3 runs with DRAM offload supplied all three and produced tens of thousands of warnings, corrupt generations, and eventually GPU crashes once eviction began. Anything short of a long, shared, memory-pressured run leaves it dormant.

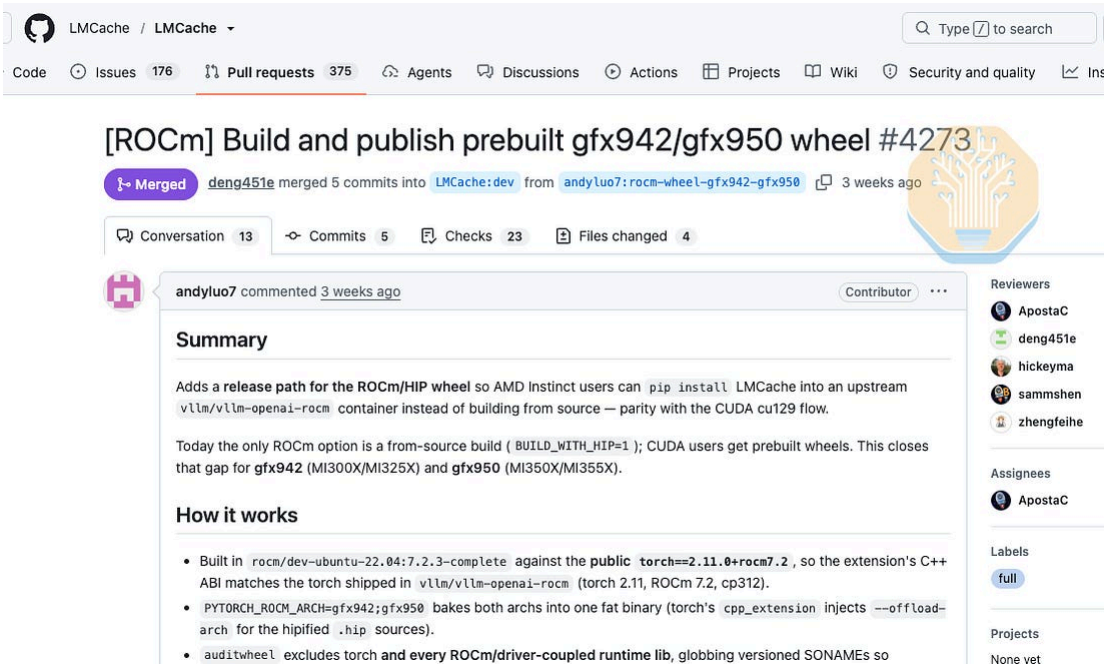
A parallel line of LMCache work made all of the above reachable on AMD Instinct hardware. CacheBlend's non-prefix reuse depended on flashinfer, which is CUDA-only, so a Triton block-sparse attention backend reimplements the three kernels it needs: block-sparse attention with CSR indices and log-sum-exp output, causal prefill, and log-sum-exp output blending. It then routes to them automatically when

QQ:1413221435

ROCm is detected or flashinfer is missing. ROCm Dockerfiles mirror the CUDA build and lightweight images. An AMD hipFile backend extends the GDS L1 slab-file tier, which reached storage only through NVIDIA cuFile, by binding ROCm's hipFile through ctypes and dispatching on torch.version.hip; the cuFile path is unchanged.

Distribution was the remaining gap. CUDA users installed a prebuilt wheel; AMD users built it from source. We worked together with AMD to publish a prebuilt gfx942 and gfx950 wheel which closes that. It installs into the upstream image and passes all 56 KV-transfer kernel tests on MI350X, and it publishes to a GitHub release rather than PyPI so a plain pip install lmcache stays the CUDA build. A one-line follow-up marks the bind-mounted repository as a git safe directory, which only fails in CI because the container runs as root over a runner-owned checkout and the version introspection in setup.py refuses to read it.

QQ:1413221435



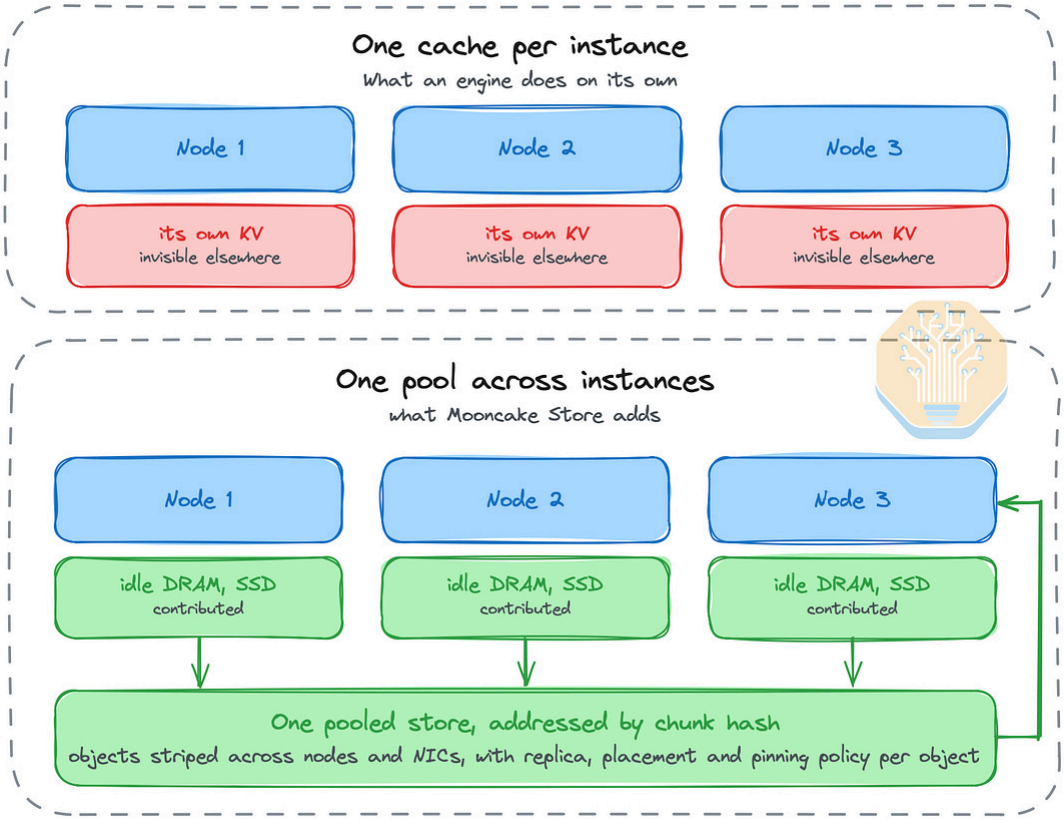
Source: GitHub

DCP-aware CPU offload resolves a straightforward incompatibility between two features that long contexts make mandatory together. With decode context parallelism enabled, each rank holds only a stride of the KV, so what any one rank could save is not a usable prefix; the fix gathers the strided shards before saving and redistributes them after loading. Without it, enabling context parallelism silently disables CPU cache hits for exactly the long prefixes that motivated both features. Its validation recorded more than 30,000 CPU hit events, with single-request loads reaching hundreds of thousands of tokens.

QQ:1413221435

Mooncake Agentic Optimizations

Mooncake serves Moonshot's Kimi production traffic along with production traffic at many labs, and is a transfer engine underneath disaggregated vLLM and SGLang configurations. Until recently, Mooncake's AMD support stopped short of both RDMA registration and offering installable packages.

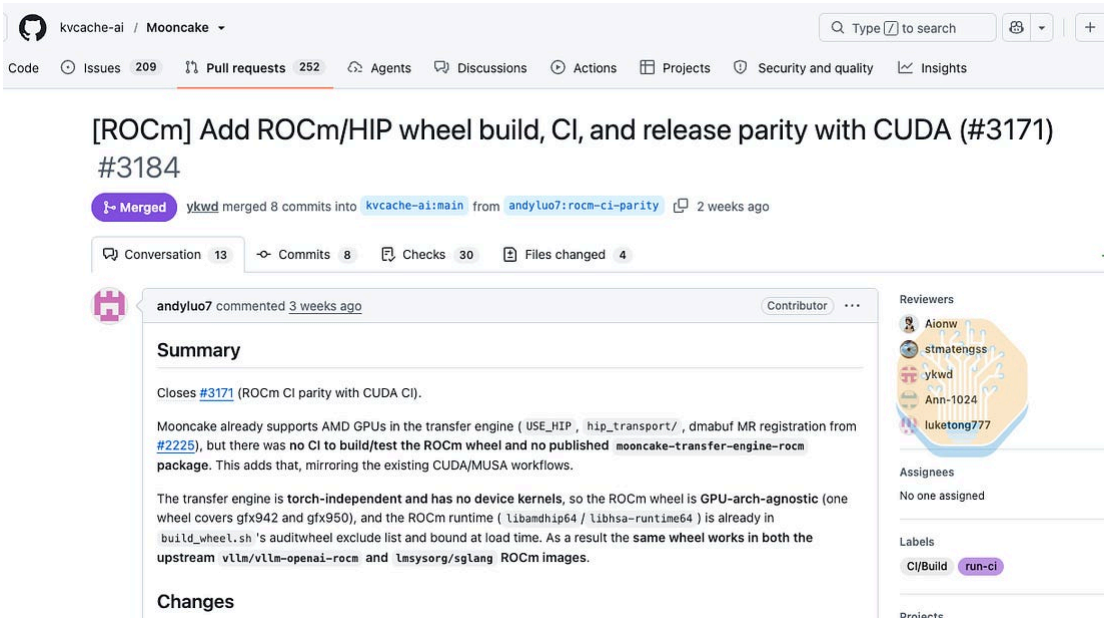


Source: SemiAnalysis

Registering GPU memory for RDMA on Nvidia either uses the nvidia-peermem kernel module or exports a dmabuf file descriptor. AMD has no nvidia-peermem equivalent, so GPU-direct RDMA had no path at all and deployments fall back to staging KV through host DRAM. A HIP dmabuf registration branch adds the mirror of the existing CUDA dmabuf path, exporting through ROCm instead of the CUDA handle call, and resolving the true allocation base first because caching allocators pack tensors at an offset inside a larger allocation. Host memory still registers directly.

Support that cannot be installed is not support. Mooncake published CUDA and MUSA wheels but no ROCm package, so AMD users built the engine from source inside every image. A ROCm wheel, CI, and release path publishes mooncake-transfer-engine-rocm to PyPI alongside them. This workstream from Andy Luo, AMD engineer, was due to noticing a pattern when dogfooding agentic workloads with AgentX that building MoonCake from source in ROCm is not an first class citizen pattern.

QQ:1413221435



Source: GitHub

QQ:1413221435

The transfer engine has no device kernels and does not depend on torch, so one architecture-agnostic wheel covers gfx942 and gfx950, and the ROCm runtime is bound at load time rather than vendored, which means the same wheel works unmodified in both the upstream vLLM ROCm image and the SGLang ROCm image. That was verified as a full cross product: MI300X and MI355X, each under vllm/vllm-openai-rocm and lmsysorg/sglang, running the master binary and a HIP buffer transfer test with data verification. The pull request adds a tag-triggered publish across Python 3.10 through 3.13. An open follow-up adds a self-hosted two-node MI350X external prefill and decode tier so the ROCm disaggregated path is exercised on real hardware rather than only compiled.

Together, these PRs mean an AMD AgentX run can now install the transfer engine and the KV cache layer from published artifacts into stock upstream images, and move KV directly between GPU memory and the fabric.

Other Optimizations

The changes above address long-context costs: a prefix that has to survive, a hybrid cache that has to stay correct, a transfer that has to keep up. But there are a whole host of day-zero enablement and correctness bugs that break requests just as badly as a million-token session.

MiniMax-M3 tested whether that ROCm work compounds into day-zero readiness, and the Advancing AI writeup draws the comparison directly: AMD’s first public disaggregated recipe, MI355X FP4, reached InferenceX in January months behind Nvidia, while M3 FP4 disaggregation landed on day zero. This is an improvement

QQ:1413221435

from the DeepSeek-R1 period, when parity took months. Three vLLM fixes sat on that day-zero path, and each was a correctness failure rather than a performance one.

Disaggregation was blocked first. NixlConnector's handshake asserted that the SPLIT-region block_len scales with the prefill-to-decode TP ratio, but block_len follows per-rank KV heads. M3 has 4 KV heads, so a TP4 prefill paired with a TP8 decode is GQA-capped to one head per rank on both sides and the two lengths are equal where the assertion demanded a factor of two. The handshake was rejected, no KV moved, decode regenerated everything from scratch, and gsm8k scored 0. Validating against the actual head ratio fixed this.

The other two were platform splits. M3's sparse-attention backend read the byte-backed FP8 cache as float8_e4m3fn for every E4M3 configuration. But gfx942's platform dtype is e4m3fnuz, and the two encodings differ. K and V were therefore altered before the kernels consumed them. The prefill and decode wrappers had also omitted the FNUZ types from their FP8 checks. Using the platform dtype for the cache view fixed both halves. Separately, M3 ships as separate NVIDIA and AMD model files, and only the NVIDIA one implemented the EAGLE3 interface, so speculative decoding aborted at engine init on ROCm with a model-does-not-support error. Bringing the AMD model to parity restored it, with MI355X gsm8k matching both the non-EAGLE3 MI355X run and B200.

The TensorRT-LLM section above covers the M3 work that is long-context specific: descriptor explosion in disaggregated KV transfer, context graph capture, sparse block strides, autotuner candidates, and the corrupt split-K MoE tactics that had to be removed from the pool.

The local AgentX matrix combines session-aware or KV-aware routing, long and variable conversation histories, MTP, hybrid attention, aggregate and disaggregated serving, and concurrency sweeps that cross the HBM capacity cliff. It includes GPU-resident comparisons and CPU DRAM offload through vLLM SimpleCPU, Mooncake, LMCache, and SGLang HiCache. That combination is what activates the upstream work above. The old fixed-sequence matrix usually creates one prompt, performs one prefill, decodes one fixed continuation, and discards the request. It therefore does not measure cache survival across turns, repeated tokenization, session affinity, cache-event traffic, offload churn, transfer progress during scheduler stalls, or long-lived ownership bookkeeping.

The allowed optimization policy treats CPU KV offload as optional. A vendor may use vLLM connectors, LMCache, SGLang HiCache, Mooncake, Dynamo KVBM, or another CPU DRAM connector, or disable offload when the resulting latency and

QQ:1413221435
QQ:1413221435

throughput point is better. NVMe offload is deferred. CPU DRAM must scale with the fraction of GPUs used, including the 3 TB cap for non-standardized-DRAM systems. Standardized-DRAM systems have no hard cap but retain the same proportionality rule. The local generator currently applies the 3 TB cap to every runner, so it does not yet implement the standardized-DRAM exception.

The net new optimization surface is not simply longer attention. It is the preservation, movement, routing, reconstruction, and repeated processing of a growing session state. AgentX made those costs large enough to drive generic upstream changes across vLLM, SGLang, TensorRT-LLM, ATOM, AITER, Dynamo, and LMCache. Direct searches of NIXL and Mooncake did not identify additional AgentX-tagged runtime PRs, so their relevant effects remain represented through the engine connector changes above.

QQ:1413221435

AgentX Methodology Deep Dive

AgentX is a massive shift in open source real world long context multi turn agentic trace replaying and we collected traces over $3M worth of tokens within our own dataset consisting of real world traffic from Claude Code, OpenAI Codex, etc. In addition to the dataset, we developed a comprehensive methodology for replaying the traffic patterns fairly. The goal is to stay as authentic to the organic traffic as possible while being equitable about GPU resource requirements.

We will deep dive into the agentic trace datasets, replay methodology, and agentic behavior in general. It is recommended reading for readers that would like to better understand the overall shape of agentic workloads as well as how harnesses orchestrate requests under the hood.

$3M USD Agentic Traffic Trace Collector

When initially designing AgentX, our north star goal was to make the benchmark as realistic as possible in terms of KV workload shape and KV reuse patterns. We began experimenting with replaying some existing datasets, such as SWE-bench, Qwen-Bailian, and other random Claude Code traces from HuggingFace. At the time, these datasets did not include significant use of subagents, 1M context, compactions, dynamic workflows, or many other of the recent defining characteristics of an agentic trace. At SemiAnalysis, most of the team are AI power users and use agents for a broad variety of tasks including coding, analyst research, excel modeling, social

QQ:1413221435

media operations, and many more. Therefore, we decided that the most achievable and realistic traces could be captured in house.

To collect a large number of traces, we created a proxy that intercepts HTTP requests to Claude / Codex. Then, users that wished to upload traces simply changed the base URL in their Claude / Codex setup to point to the proxy. At the time of writing, we have collected over 8,000 sessions, 3.4 million requests, and 610 billion tokens. Together these represent more than $3M USD in spend. We open sourced a representative subset of these sessions for the AgentX v1.0 benchmark.

Although agentic harnesses can appear complex, they ultimately orchestrate a sequence of HTTP requests. Each request contains some combination of system instructions, tool definitions, and accumulated conversation history. As a session progresses, this history grows and is repeatedly sent back to the model, creating the long contexts and high prefix reuse that AgentX is designed to reproduce.

QQ:1413221435

Our proxy records these requests and responses as they occur. It also extracts metadata/HTTP headers such as timestamps, conversation IDs, and subagent IDs, which lets us recover the *structure* of the conversation (request ordering, concurrent branches, and the approximate parent/child structure of each session). This metadata is what allows us to replay the traces approximately how they would have been seen by the original Anthropic API server.

To protect employee privacy, the replay dataset contains no original prompts, source code, tool arguments, or tool results. Instead, we tokenize each requests' content and then group it into 64-token blocks, finally replacing each block with a session-scoped chained hash. Matching prompt prefixes therefore produce matching hash prefixes without revealing their contents (this paper talks more about this strategy). During replay, these hash blocks can then be replaced with tokens from, say, a coding dataset. So we preserve the approximate context growth and conversation KV-reuse patterns of the original workload.

It's worth noting that this process is necessarily imperfect, mostly due to the fact that when using a frontier model provider's API, much of the content that the end LLM server actually sees is hidden. For instance, thinking/reasoning content for SOTA models are now encrypted in HTTP requests and replaced with a deterministic hash in an attempt to hinder distillation attacks. However, it seems like this didn't work out exactly as well as the big labs had planned…

QQ:1413221435

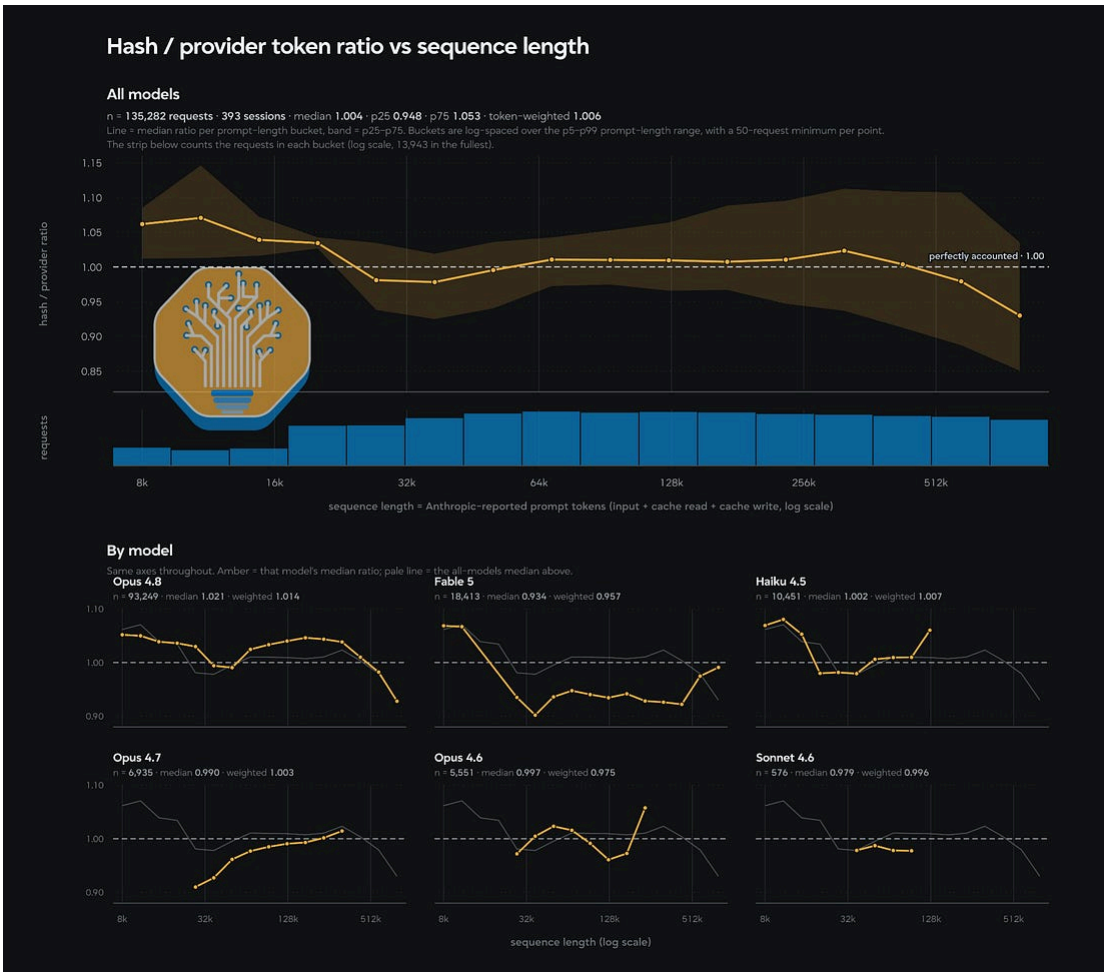


Source: SemiAnalysis

Furthermore, while we have access to all the raw *content* of the user/assistant messages, system prompts, tool usage, etc., API providers apply additional chat templating server side that aren't transparent. We also cannot observe Anthropic's proprietary tokenizer or context introduced by server-side tools. Images and documents also do not have a straightforward correspondence between their wire representation and the number of tokens processed by the model. We use deterministic placeholders and empirically calibrated, model-specific padding to bring reconstructed prompt lengths to best estimate the content that is actually seen by the server.

We can't *perfectly* capture and replay Claude Code / Codex traces as they are *actually seen* by the Anthropic / OpenAI servers due to incomplete information, but we can get pretty close. The chart below shows the ratio of hash tokens (after our approximations/processing) and the true API provider token count across all request lengths and models.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis

To summarize, collecting real Claude Code traces in the exact way that they would have been replayed against the original server is not easy due to incomplete information. However, we have enough context to collect and replay traces with extremely high fidelity to match original traffic patterns, timing, prefix caching, and DAG patterns.

AgentX $3Mil Dataset

The dataset used for AgentX v1.0 can be found on HuggingFace. It is a 393 session subset of 8.3k session proxy corpus mentioned in the previous section. Additionally, we applied some post-processing to clean up anomalies, such as:

- Removing Claude Code security monitor (auto mode) requests and title generation requests as these are specific to Claude Code and not necessarily representative of general agentic traffic
- Removing requests with a reconstructed input length greater than 990k tokens (where our approximation overcounted)
- Remove duplicate requests (sometimes the proxy received identical requests if the connection was dropped)

QQ:1413221435

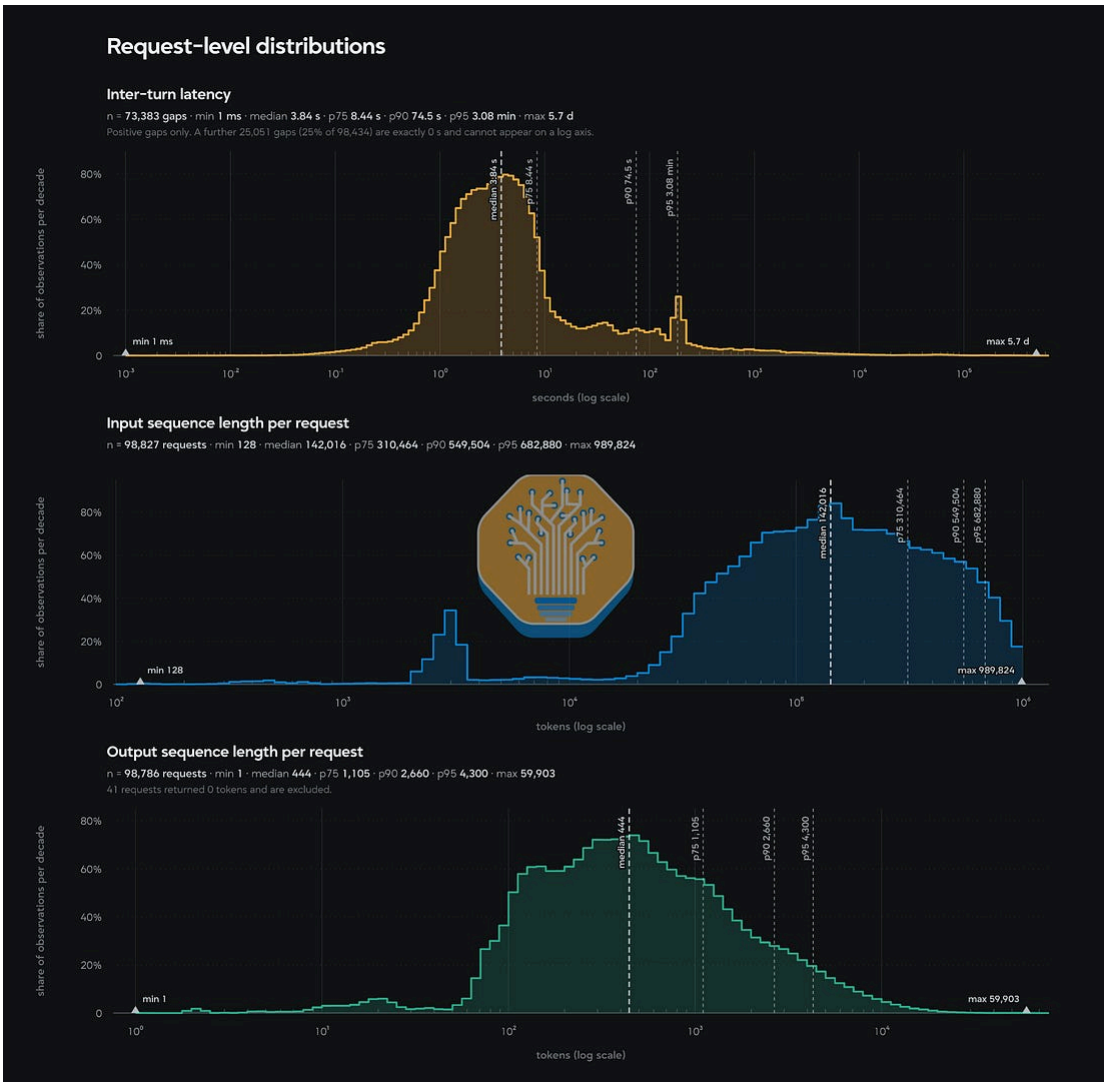
Additionally, each conversation is formatted into the WEKA trace format, proposed by Callan Fox as part of his kv-cache-tester project. We chose this format for storing trace information primarily because we worked closely with Callan to develop the benchmark and found it intuitive for storing per-session traces. All in all, the trace format is quite arbitrary and our proxy dataset could be mapped to other formats such as Mooncake.



Source: SemiAnalysis

After these are applied, we get the following dataset. Note that not all X-axes are the same.

QQ:1413221435
QQ:1413221435



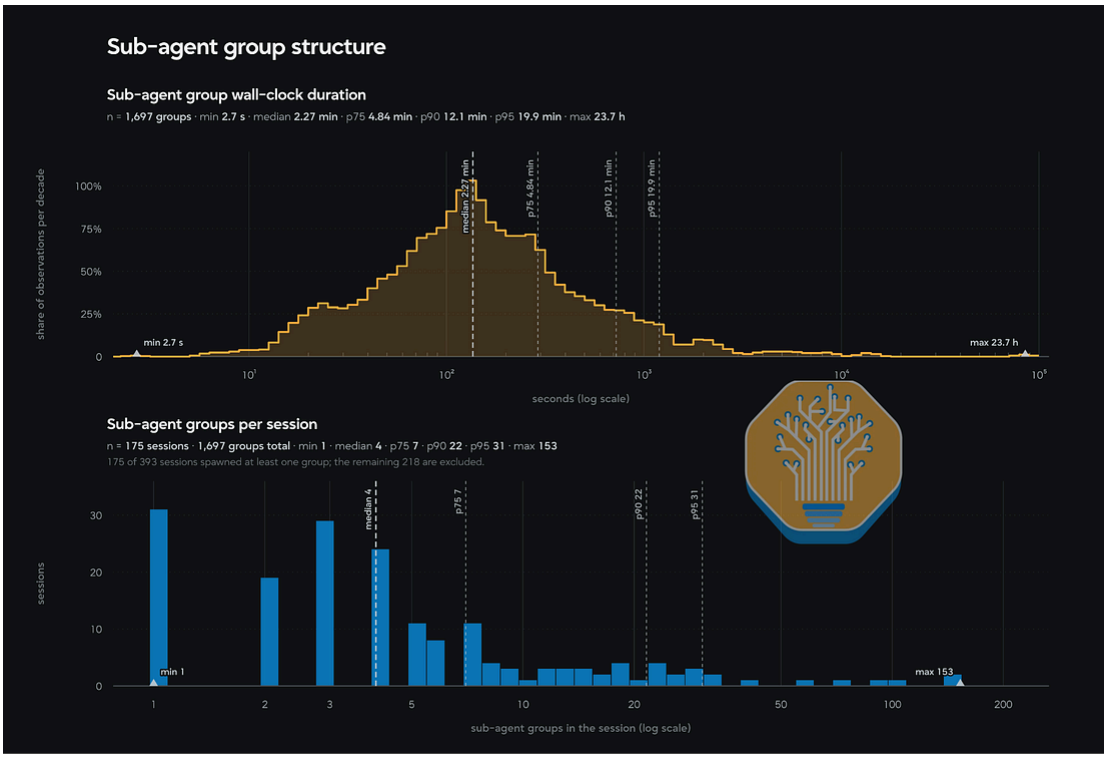
Source: SemiAnalysis

The distributions for ISL/OSL and inter-turn latency (in agentic work, this is mainly time taken for tool use) are relatively log-normal. The median ISL is 142k tokens and the median OSL is 444 tokens. The median inter-turn latency (or “tool use time”) was 3.84 seconds. Only ~10% of inter-turn latency was greater than 1 minute. These are likely made up of gaps where the harness is waiting on an actual response from a human.

One thing worth mentioning is that these request distributions will look different depending on which harness is being used, since different amounts/types of context are injected (for instance Pi is known to be minimalist in terms of harness-injected context while Claude Code is known for the opposite. Additionally, the ISL/OSL distributions will depend on the model, as different models have tokenizers that can produce either more/fewer tokens. However, given a significant portion of the world’s agentic coding traffic goes through Claude Code, we believe this is rather representative.

QQ:1413221435
QQ:1413221435

The dataset also has 175 sessions with at least one subagent (~44% of all sessions). There are 1,697 total subagent rollouts in the dataset, with a median of 4 per session. The median wall-clock time for a subagent (beginning of first request to end of last request) is 2.27 minutes. This distribution again follows a relatively log-normal distribution.



Source: SemiAnalysis

This dataset includes context up to 1M context, meant to test the more recent frontier open-weight models. Additionally, we have a truncated 256k context length dataset which we replay against models with a max context length of 256k or less.

Agentic Traffic Trace Replayer

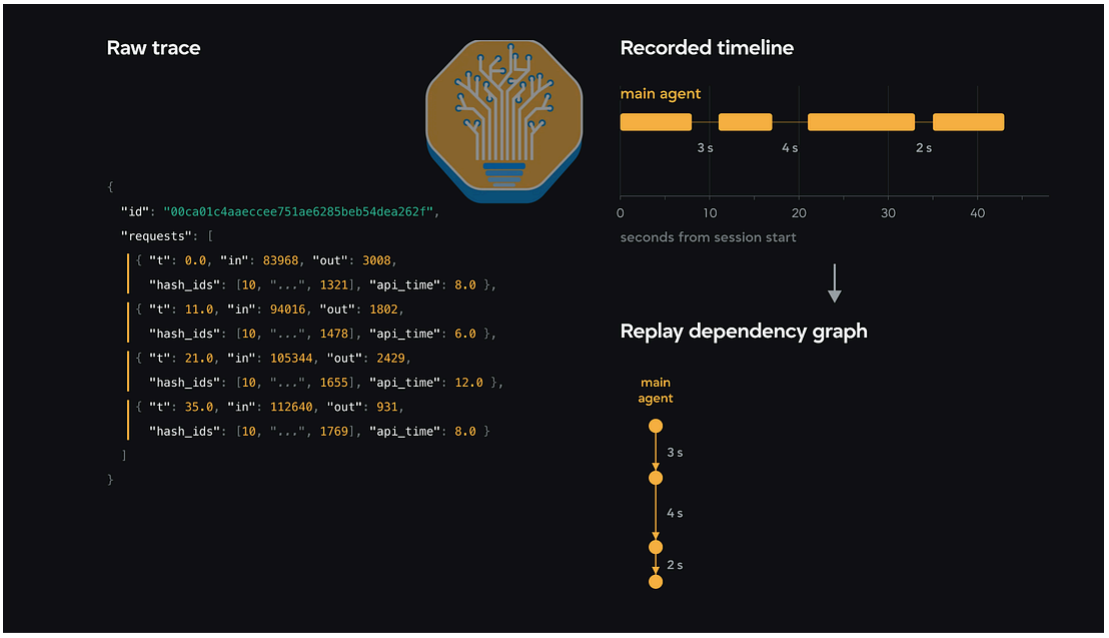
Rather than build a replay solution from the ground up, we decided to partner with AIPerf, a vendor agnostic HTTP replayer tool from Nvidia that is adopted by many in the industry including tenstorrent, AWS, AMD, etc. While the intention is to integrate AgentX features into the upstream repo, we maintain a separate fork to be even more vendor neutral such that we have control over allowing even more 3rd more contributions. Again, thank you to the AIPerf team, especially Anthony Casagrande, for the help and dedication to building a realistic and representative agentic benchmark.

An agentic session is naturally described as a directed acyclic graph (DAG). Each request is a node, and an edge means the request at its head cannot be issued until the

QQ:1413221435
QQ:1413221435

one at its tail has completed. Every edge additionally carries a delay, specifying how long to wait once that precondition is met.

The simplest session is completely linear, with no subagents and no parallel requests, each request depends on exactly one predecessor. The graph degenerates to a line and the only thing an edge encodes is the inter-turn latency (aka, tool use time or “think” time), which is the client’s local work rather than the model’s.



Source: SemiAnalysis

In agentic traces, subagents can also be spawned. A subagent is a separate stream of requests that has its own context, typically to do a focused task. Multiple subagents can run in parallel to do more aggregate work, and subagents can run in parallel to the main agent in some cases. The main agent then waits on groups of subagents to finish, and then incorporates their outputs back into the main agent’s context (while this is not *always* the case, this is the most common pattern).

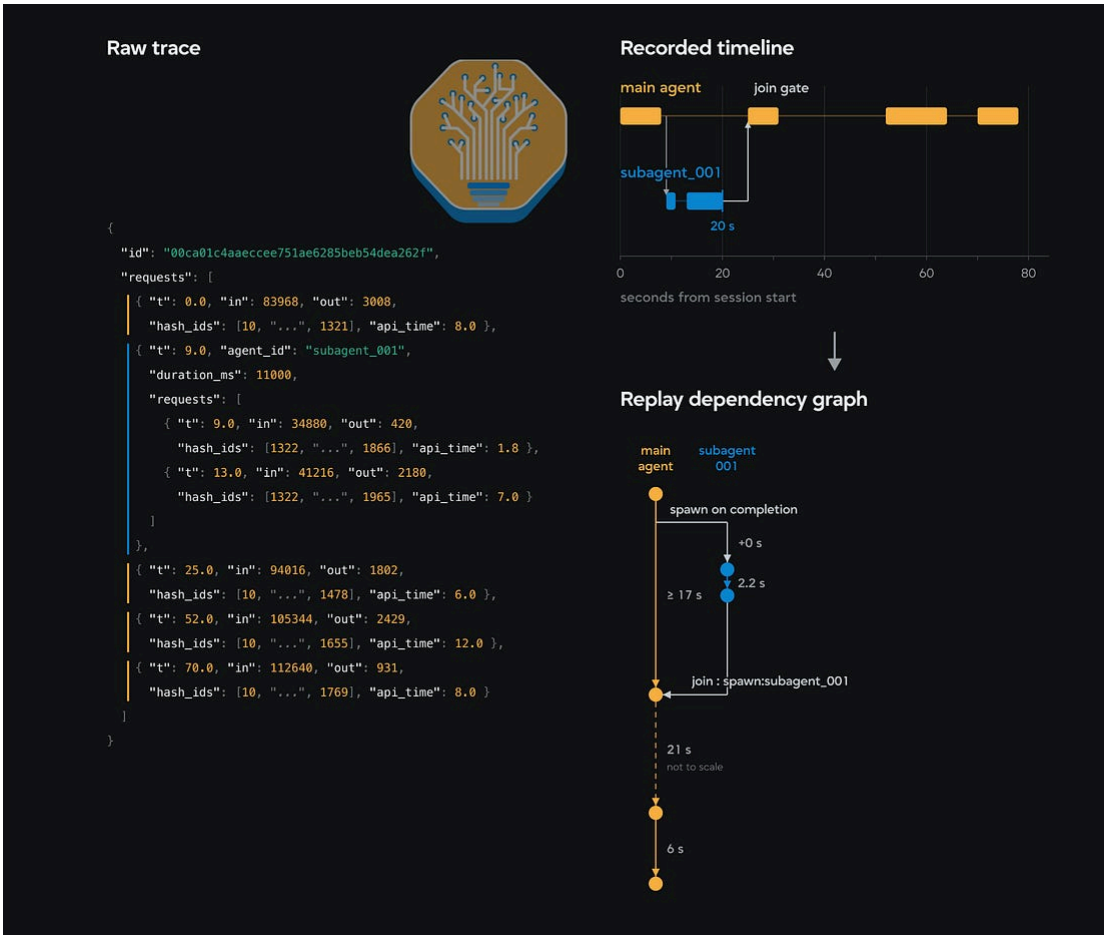
This behavior is responsible for turning the linear chain of requests above into a DAG, where certain requests are dependent on others. When a group of requests belonging to one subagent is identified, AIPerf finds the most recent main agent predecessor and designates it as the “spawning” request. Similarly, the “join” request is identified by the subsequent main agent request after the duration of the subagent group completes.

In the example below, the subagent group consists of a single subagent (001), which runs two requests. Its first request goes out as soon as the main agent’s opening request completes. When that request completes, a 2.2-second inter-turn delay stands in for tool-use wall-clock time, and then the subagent’s second request is sent. The

QQ:1413221435

main agent's second request is the join point for subagent 001. It goes out once both conditions are met: at least 17 seconds have elapsed since the main agent's first response *and* subagent 001's second request has completed.

One small limitation is that HTTP timestamps reveal timing, but not always causality. In the example below, if subagent 001 finishes with seven seconds remaining before main-agent request 2's recorded start, we cannot tell whether those seven seconds represent work performed after the subagent returned or independent work already underway. AIPerf therefore preserves both constraints: request 2 waits for its recorded main-path delay and for subagent 001 to finish. This reproduces the observed timing and workload topology, but not any dependencies hidden inside the harness. These are things we hope to improve on in subsequent versions of AgentX.



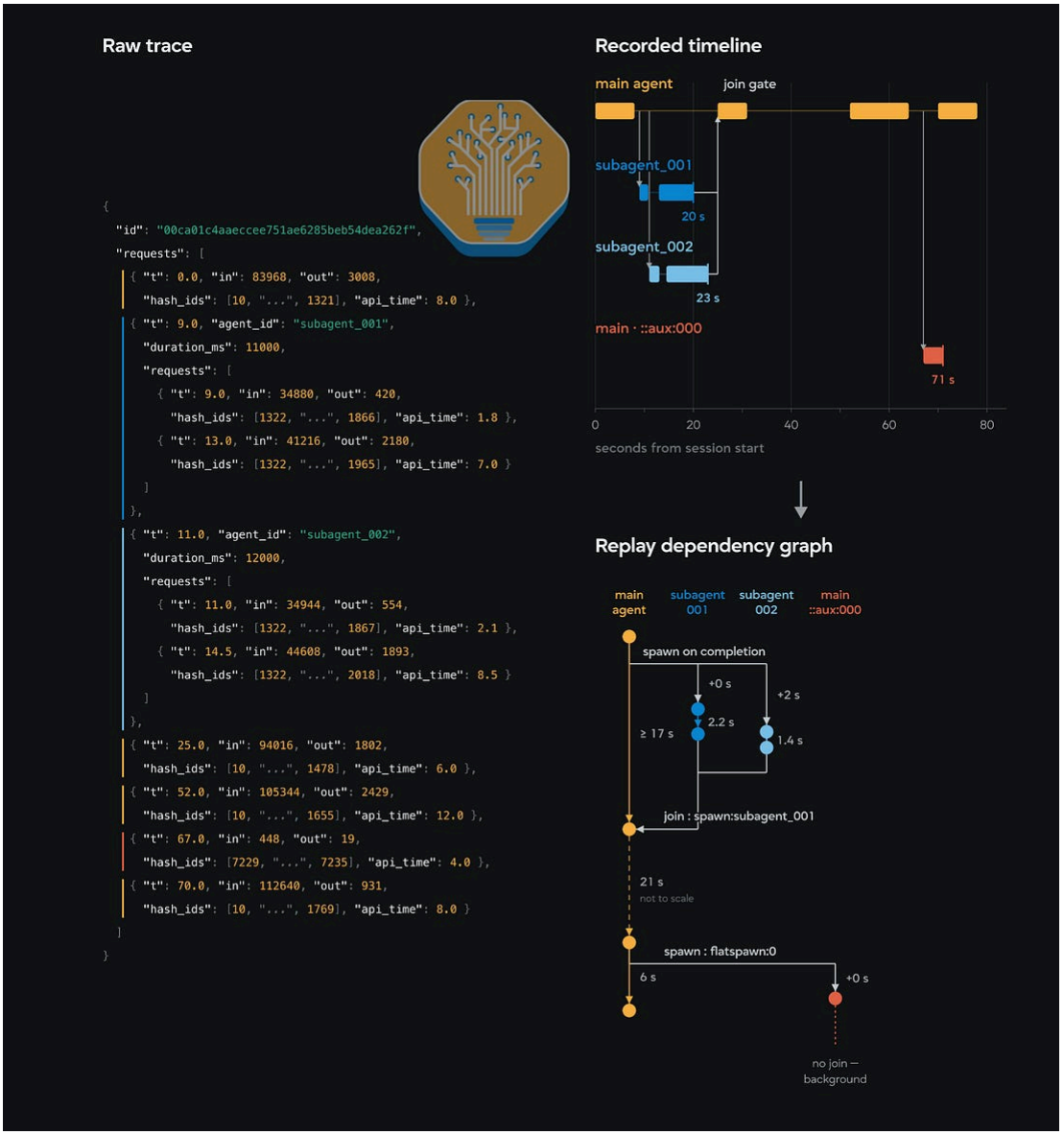
Source: SemiAnalysis

Multiple subagents can also be spawned from a single request. In the example below, subagents 001 and 002 both identify their spawning parent as main agent request 1, and then join at main agent request 2.

AIPerf can also identify "auxiliary" requests, which are one off requests that do not share context with any other requests in the stream. These branch off of the main agent and never join back. In practical terms, these are requests like Claude Code's

QQ:1413221435

“summarize this session” requests that are unrelated to the conversations context. Another good example is Claude Code’s “/btw” feature.



Source: SemiAnalysis

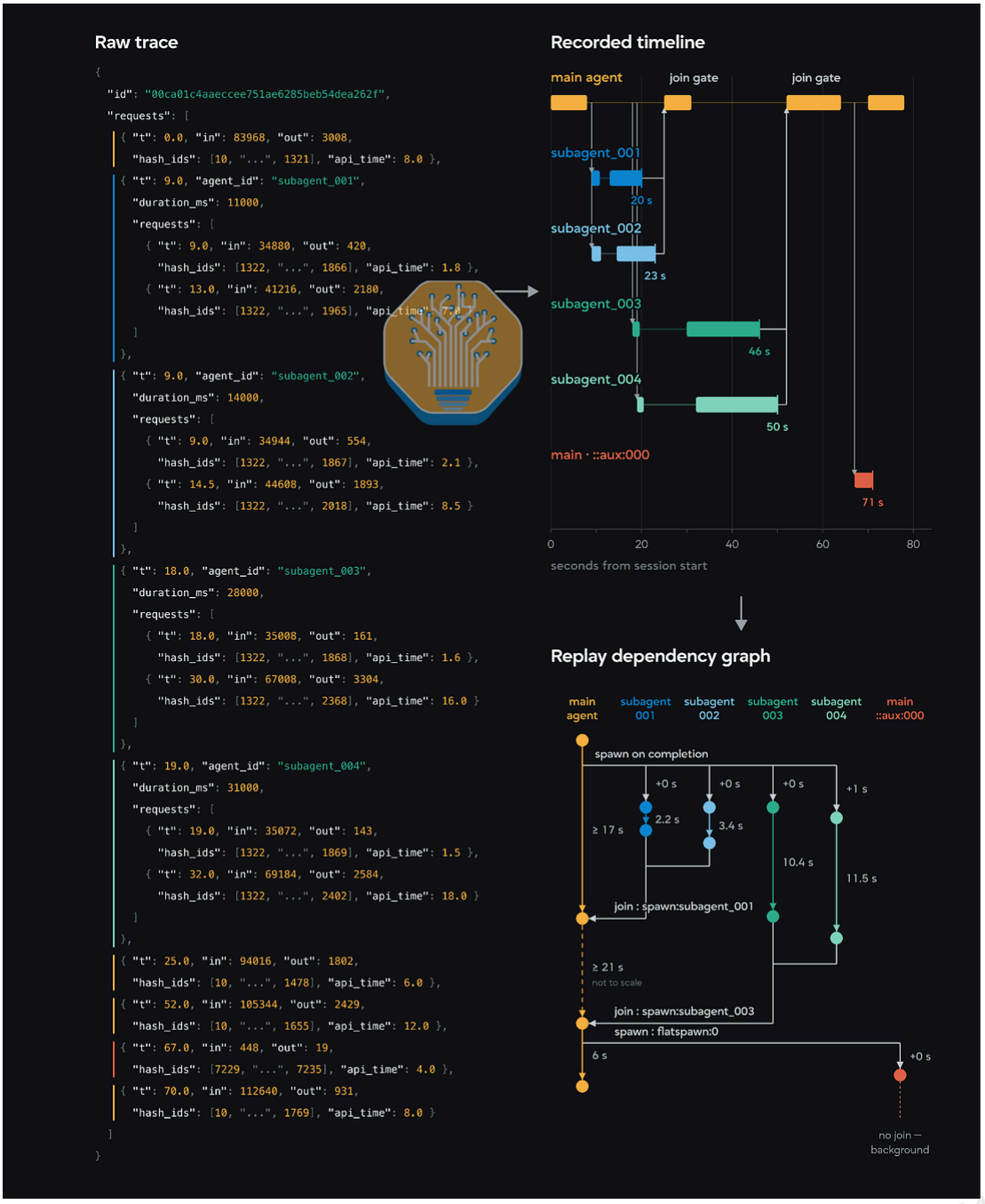
The example below brings it all together. This is the type of trace snippet you’d see in the actual dataset. We have five parallel streams leaving a single main agent request, separated into groups defined by what main agent request they join upon.

Subagents 001 and 002 finish at 20 and 23 seconds, so the first main agent request starting after that, at 25 seconds, is their join. Subagents 003 and 004 are spawned in the same gap but run much longer, finishing at 46 and 50 seconds, so they join at 52 seconds instead. AIPerf keys each subagent by the pair (spawning request, join request), which means these four streams collapse into two branches even though all four leave the same node. The branch takes the name of its first member, which is why the join edges are labelled with subagent 001 and subagent 003.

QQ:1413221435

This is also the first case where the main agent overlaps its own subagents. The request at 25 seconds goes out while 003 and 004 are still running: the main agent is blocked only on the group that joins it, not on every subagent in flight.

The auxiliary chain attaches to whichever main agent request most recently preceded it, which here is the request at 52 seconds rather than the one that opened the session. That node therefore does two things at once — it receives the second subagent group's join, and it spawns the one-off. Of course the auxiliary request never joins back.



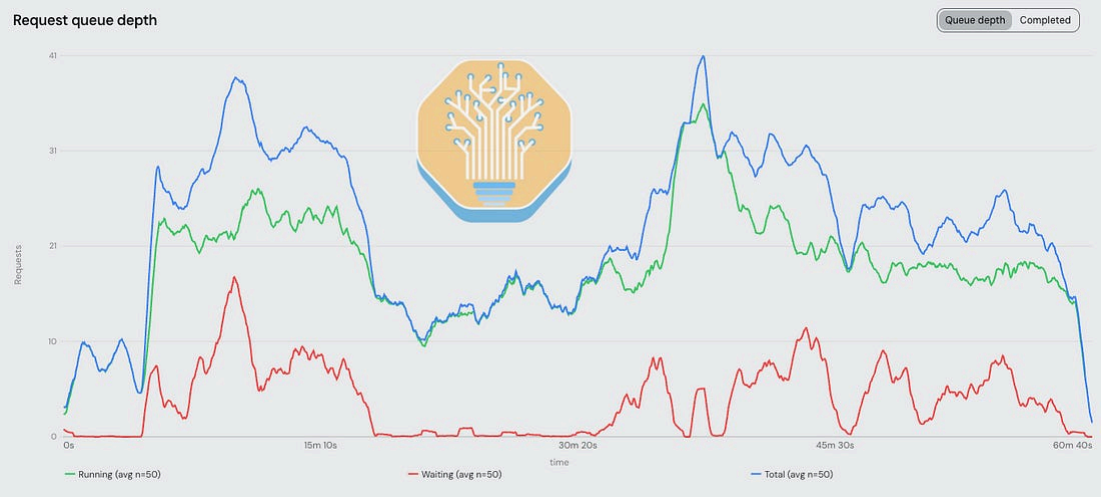
Source: SemiAnalysis

QQ:1413221435

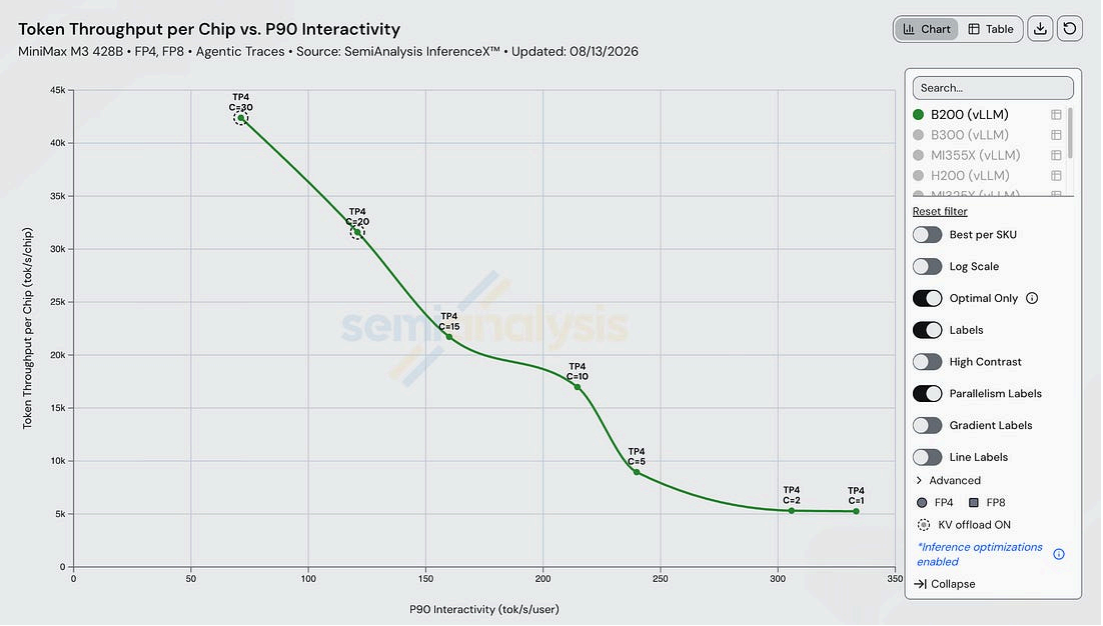
Further AgentX Deep Dive

Pareto Frontiers Sweeping

In the AgentX workload, in order to generate a Pareto frontier, we sweep over the number of concurrent Claude Code sessions against a single deployment. Since each conversation has realistic inter-turn delays and subagent usage, we get a spikier, more realistic traffic pattern. The example below is an example of replaying 40 concurrent clients against a B200 TP4 vLLM server running MiniMax M3.



Source: SemiAnalysis InferenceX



Source: SemiAnalysis InferenceX

A final point worth discussing is what metrics are important to consider when evaluating agentic workloads. We believe interactivity (TPS - tokens per second) and time-to-first-token (TTFT) are still important, and these are the industry standard for evaluating SLOs. When viewing AgentX results, it is *extremely important* to consider both TPS and TTFT together, since there are inference optimizations that can improve one at the cost of the other.

QQ:1413221435

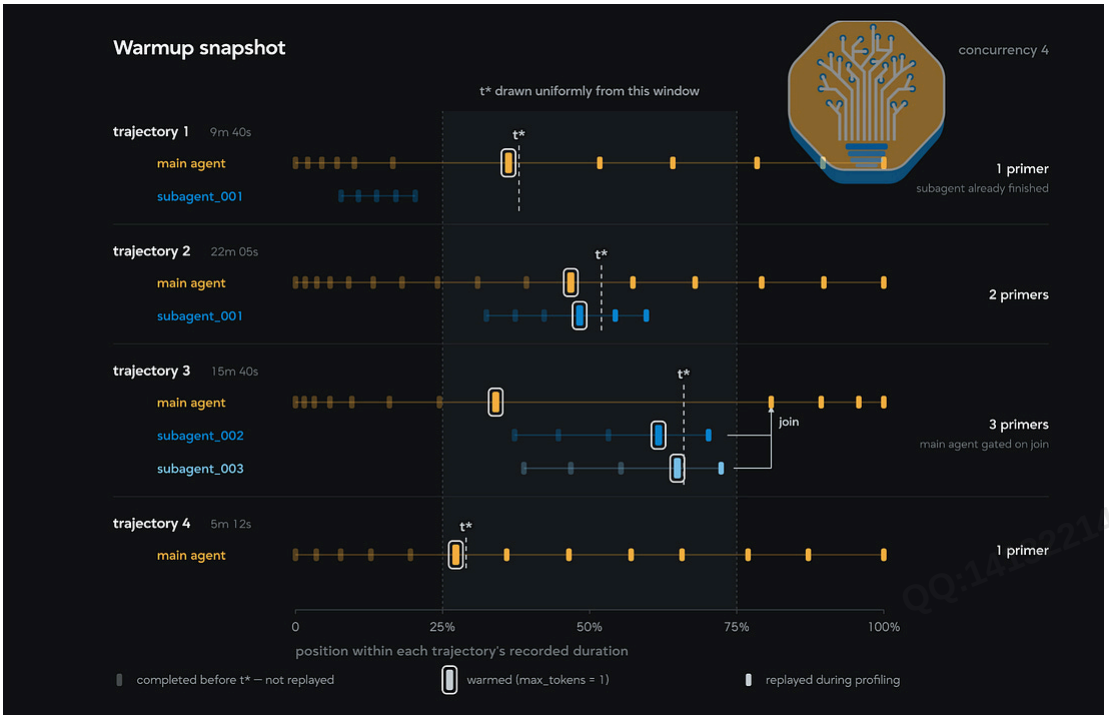
We are currently working on defining a new metric that combines TPS and TTFT in a meaningful way. This should also take into account that in agentic workloads, people often care more about the end-to-end speed at which a *task* finishes rather than how fast they receive tokens or TTFT.

Finally, it's worth mentioning that the end-to-end latency, in its current form, is now less meaningful because end-to-end latency is directly proportional to OSL. Therefore, P90 E2E latency is heavily affected by the 10% tail of longest output sequence lengths. While it can still be good to holistically compare the overall performance of certain configurations, we recommend instead looking at a combination of TPS and TTFT.

Warmup, Timing, Determinism, and Conversation Reuse

The goal of AgentX is to benchmark systems that are already in a steady state. With agentic workloads, this means that profiling should start from a point where some context trajectories are already cached. To mimic a steady state, it is also desirable that not all conversations start at turn 0, which may cause a "thundering herd" effect.

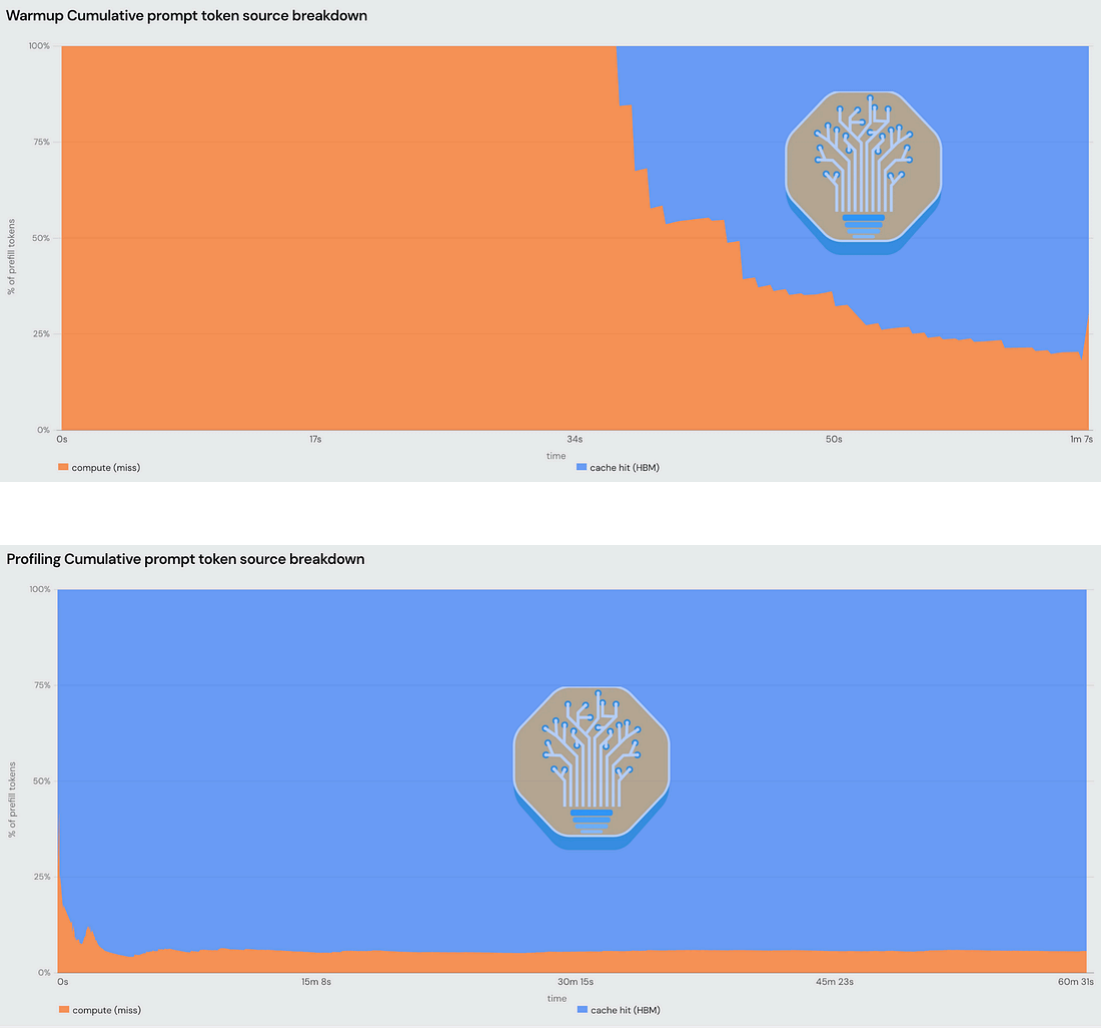
Warmup proceeds in two stages. First, AIPerf uses a fixed random seed to select a wall-clock point between 25% and 75% of each conversation. At that point, it identifies every active request stream, including the main agent and any active subagents, and sends the most recent request before the selected point for each stream. These primer requests reconstruct the conversation state at that point and are dispatched together. AIPerf waits for them to drain before continuing.



QQ:1413221435

Source: SemiAnalysis

In the second phase, each replay lane is advanced by 10 additional requests to give additional opportunity for the KV cache to materialize. All warmup requests omit inter-turn delay and use a maximum output length of one token, substantially reducing warmup time.



Source: SemiAnalysis

When warmup is complete, profiling begins and lasts one hour. All metrics are collected strictly over this duration. For reproducibility, AIPerf accepts a seed that ensures each run samples conversations deterministically, conversations start at the same point, and that each conversation is reconstructed with the same synthetic content run-to-run. During profiling, we impose a 5 minute idle time cap on each stream, so that long inter turn gaps don't "acquire" a worker lane for the duration of the benchmark. We enforce this so that we can effectively run the benchmark in 1 hour. In later versions of AgentX where we include NVMe offloading, we may choose to increase this so that we can measure a longer TTL. For now, 5 minutes is reasonable as this is Anthropic's default KV cache TTL.

QQ:1413221435

This level of determinism ensures that runs using the same inference engine, hardware, concurrency, and server settings are reproducible. However, because AgentX is a closed-loop benchmark, different configurations will complete different numbers of requests at different rates, introducing some natural variation in the workloads they encounter. This is most noticeable at lower concurrency, where fewer requests are completed (naturally) and the workload has less opportunity to converge toward the dataset's overall distribution.

When a conversation completes during profiling, its replay lane selects another conversation from the dataset sampler. Each replay receives a unique, deterministic cache-bust marker that is prepended to every independent prefix chain, including the main-agent chain and any fresh-context subagent or one-off chains. Forked subagents inherit the marker from their parents. The marker stays the same within a replay, preserving its KV-reuse patterns, but changes between replays to prevent artificially high cache-hit rates. This also enables scenarios to run where concurrency is greater than the number of conversations in the dataset (393).

QQ:1413221435

AgentX's Fair Speculative Decoding Methodology

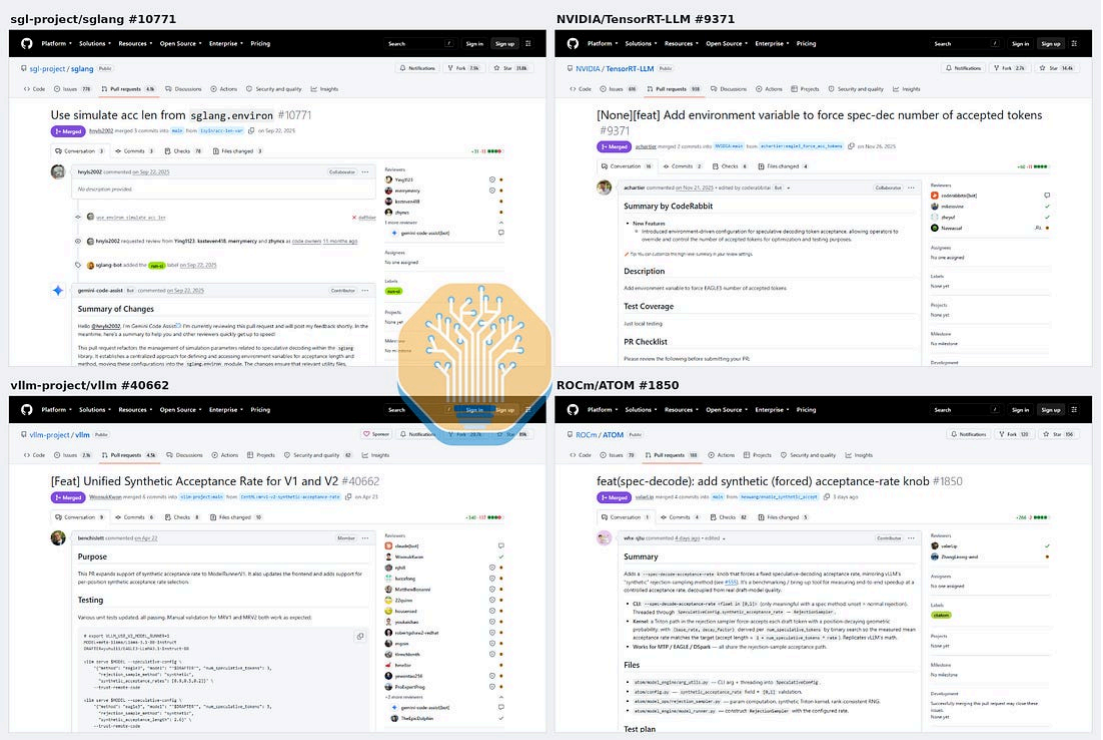
As mentioned, the dataset is anonymized upon collection. This means that the 64-token hash blocks must be synthetically filled in before replay. AIPerf accomplishes this by deterministically sampling from a synthetic coding/tool-use token pool.

Importantly, the KV reuse patterns as well as request timing are maintained, however the synthetic request data does lead to some additional considerations. Namely, running speculative decoding methods on synthetic data may lead to the speculator rejecting/accepting an abnormal number of tokens when compared to non-synthetic data (since the speculator is not trained on synthetic data).

We talked about this shortcoming in our InferenceX v2 article, and have since then improved on our methodology. We have worked closely with the community to ensure a mechanism exists in most OSS inference engines that allows users to force how many draft tokens to accept from the speculator (aka, "acceptance length" or "acceptance rate"). Then, for each (model, speculator, draft length, and thinking mode) combination, we collect the average AL on the SPEED-Bench agentic coding dataset, a "unified benchmark designed to evaluate speculative decoding (SD) across diverse semantic domains and realistic serving regimes."

QQ:1413221435

Then, at runtime we apply these realistic Speculative decoding acceptance lengths to AgentX to ensure vendor neutral fairness.



Source: GitHub

QQ:1413221435

Exploring Agentic Workloads Detailed Telemetry Tutorial

AgentX required more than a new benchmark harness and dataset. We also spent some time rebuilding parts of the InferenceX visualization to make agentic results easier to explore and digest. A single AgentX datapoint represents thousands of requests across growing conversations, subagents, warmup periods, cache states, and dynamically changing in-flight load. Due to this, having a single point on a Pareto curve can hide a lot of useful information. As we have said many times - there is never just a one size fits all solution for inference serving.

QQ:1413221435

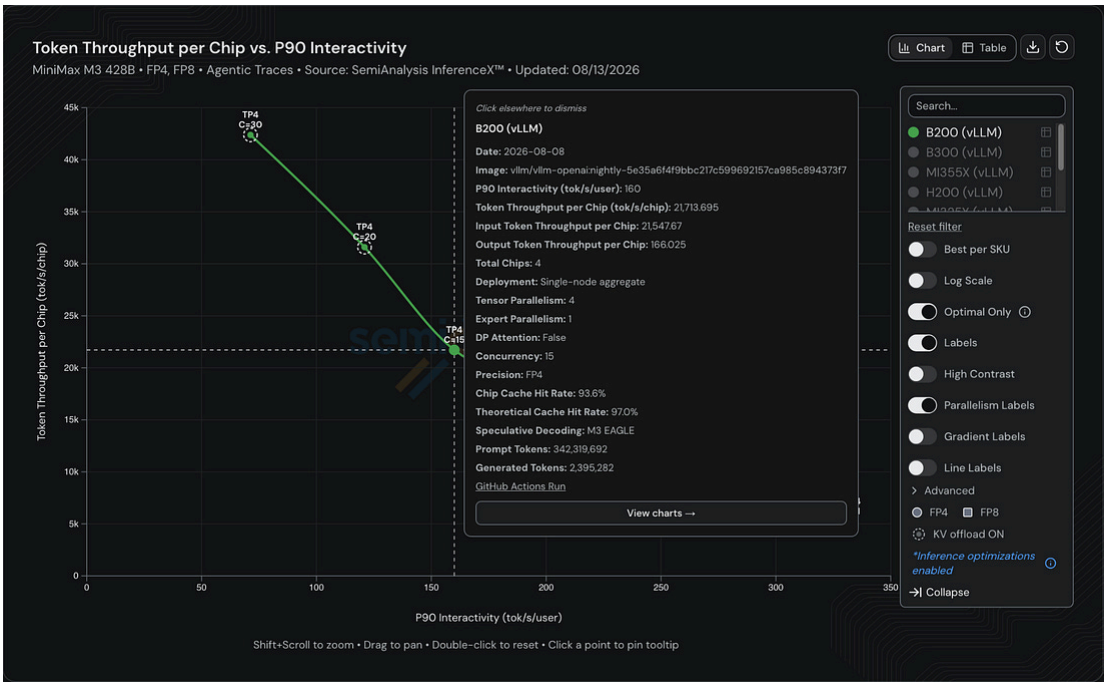


Source: [SemiAnalysis InferenceX]

One of our major changes is how we construct the curves themselves. In previous versions of InferenceX, configurations with speculative decoding enabled and disabled were often displayed as separate curves. However, we are now moving away from this approach. The frontend now combines allowed inference optimizations and displays the best available curve for each model, SKU, and inference engine combination. Due to this, individual points along a single curve may use different optimization techniques and configurations, including speculative decoding, disaggregation, or KV cache offload.

Our goal is to show the best production performance available from each hardware and software stack, rather than creating a separate curve for every possible combination of optimizations. However, we still expose the underlying configuration and provenance for every point. Clicking a point shows a tooltip with a detailed view showing exactly which configuration produced it, along with the run metadata, links to the publicly viewable CI provenance, and AgentX specific statistics. From there, the “View charts” link opens the full point-detail page with AgentX specific statistics.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis InferenceX

QQ:1413221435

The detailed point view provides a much deeper look into the selected AgentX run. It includes input and output sequence length distributions, interactivity and TTFT over time, KV cache utilization, request queue depth, prefix cache hit rate, input and decode throughput, prompt-token source breakdown, and unique input tokens over time. These metrics make it easier to understand why two points with similar aggregate throughput may behave differently throughout the replay.

The page also separates warmup and profiling data. Readers can switch between the two phases to inspect how the system behaves while its cache state is being established and during the profiling period used for the benchmark run.

QQ:1413221435



Source: SemiAnalysis InferenceX

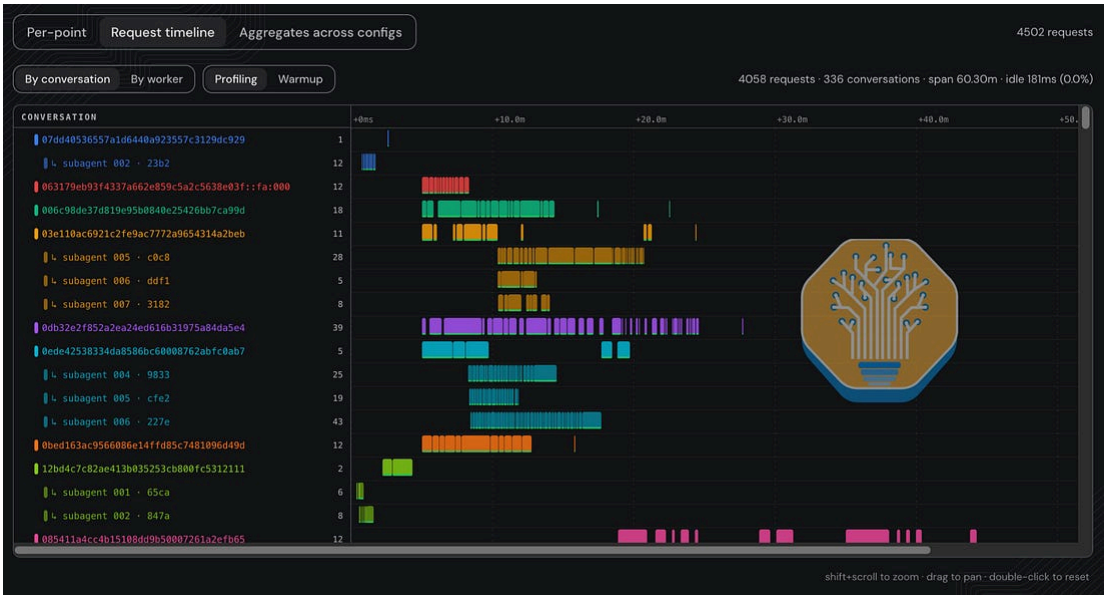
QQ:1413221435

Points using KV cache offload are surrounded by an additional dotted circle on the main chart, which is used to distinguish points with KV offload enabled. When one of these points is selected, the detail page shows the offload type, KV offload engine,

QQ:1413221435

chip cache-hit rate, and CPU cache-hit rate. This makes it possible to see where KV offload contributes to the best curve without creating a separate curve for every offload configuration.

Another new feature is the request timeline. This view shows the individual requests replayed during a selected AgentX run and can be organized either by conversation or by worker. The conversation view groups subagents underneath their corresponding root conversation, making it easy to see when conversations and subagents overlap. Warmup and profiling requests can also still be viewed separately.

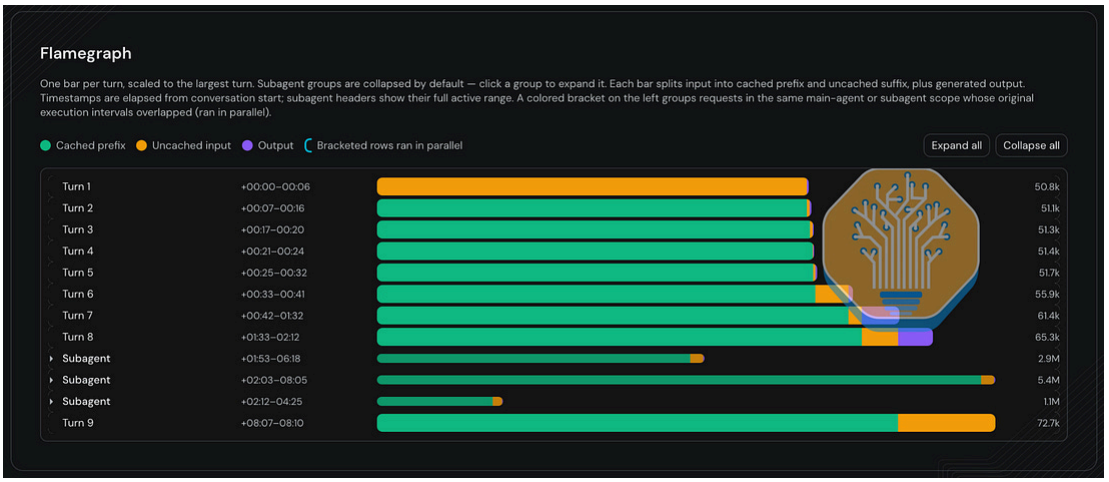


Source: SemiAnalysis InferenceX

Each request in the timeline is clickable and links directly to the corresponding conversation and turns on the InferenceX datasets page. This allows readers to move from an aggregate point on the Pareto curve to the exact anonymized request that was replayed.

The AgentX page also includes a flamegraph for visualizing the structure of an individual conversation. Each bar represents one turn and is scaled relative to the largest turn in that conversation. The bar is divided into cached prefix tokens, uncached input tokens, and generated output tokens. This gives a visual representation of how the context grows throughout a conversation and how much of each request can be reused from KV cache.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis InferenceX

InferenceX/AgentX Future Next Steps

We are excited to continue our goal of making AgentX the most realistic and representative benchmark. For the foreseeable future, we will continue making small bug fixes to the current v1.0.x harness, but have plans to make bigger changes to the v1.1 harness. Submissions for each distinct model will always run the same minor version to ensure all results are comparable.

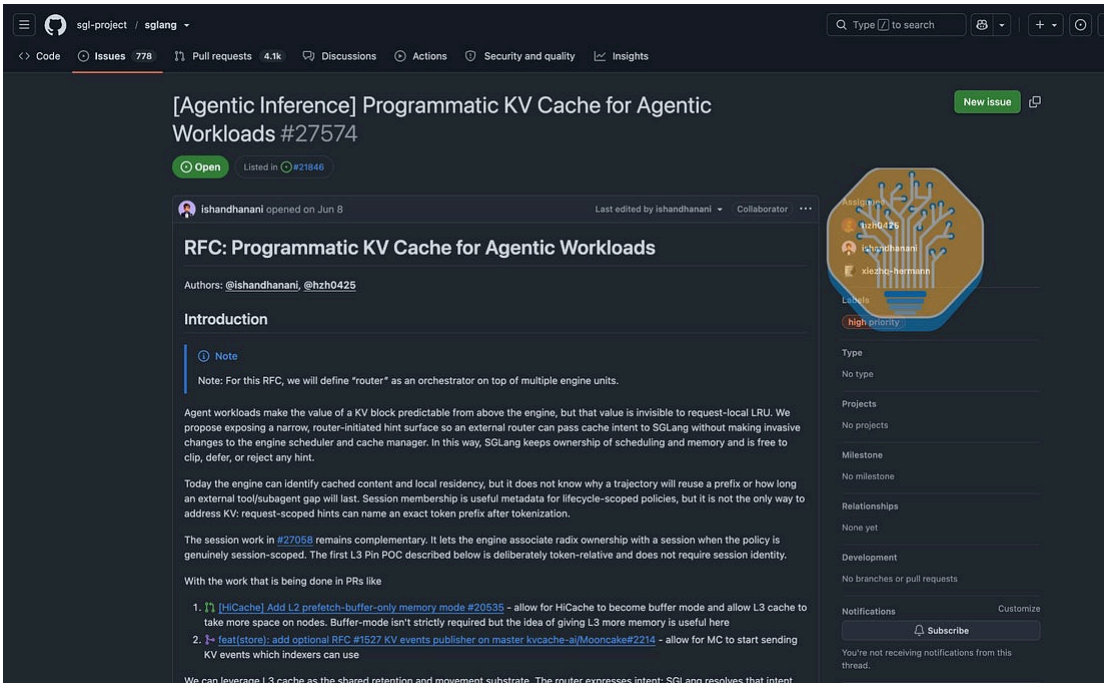
As a fast follow, we will add SSD/NVMe KV offloading. This will allow an even larger KV cache working set size than DRAM allows, which will allow the high throughput left side of the pareto curve.

We will also capture a larger, more diverse, and more recent dataset of agentic traces across a wider variety of models and harnesses. Rather than representing each request as one contiguous list of hash IDs, the next dataset will preserve the boundaries between system instructions, user and assistant messages, tool calls, and tool results. This will allow AgentX to evaluate workload-aware serving techniques that use information available to the agent harness but normally hidden from the inference engine. For example, a router could direct low-reuse tool traffic to dedicated prefill workers, retain an agent's prefix during a long tool call, or prefetch and share prefixes when subagents fork. The current format preserves request sizes and KV-reuse patterns, but it lacks the structure needed to evaluate these optimizations.

The following SGLang RFC by Ishan provides a concrete example of why this richer trace structure matters. It proposes a router-initiated hint interface that uses information such as session lifecycles, shared-prefix boundaries, tool-call duration, and subagent state to tell the engine when KV should be shared, prefetched, demoted, pinned, or retained. Capturing this structure would allow future AgentX versions to

QQ:1413221435
QQ:1413221435

evaluate these workload-aware cache policies instead of only replaying flat token prefixes.



Source: GitHub

Lastly, we are capturing fine-grained and coarse grain power telemetry data to have an even more accurate view of the efficiency of the Joules per intelligence of different software and hardware stacks.

We have so much data and so many possible visualizations. Please let us know any visualizations you would like to see, as well as any more general feature requests!

Suggest Frontend Feature!

Performance Throughout the Model Lifecycle

In the following sections, we turn to our historical single turn data (8k1k), which covers each model from launch day through the point we retired it from active testing. There are strong results here from both Nvidia and AMD, including several fixed sequence length configurations where MI355X comes out ahead.

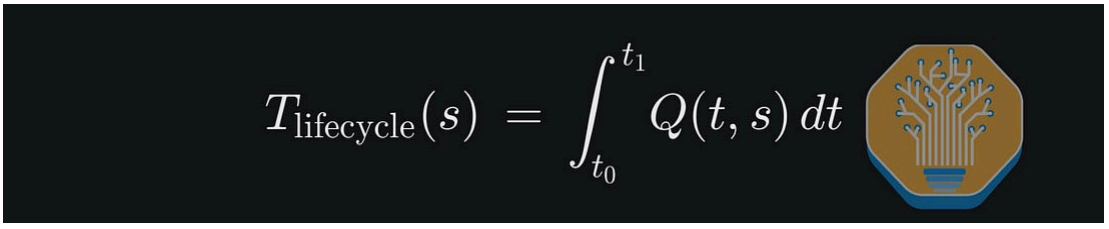
AgentX better represents today's agentic inference workloads. However, the historical fixed-sequence InferenceX results still remain useful for tracking the performance over time. Workloads such as 8k1k and 1k1k strip away most session-level behavior, including prefix reuse, persistent KV cache state, and routing affinity.

QQ:1413221435
QQ:1413221435

This makes them less representative of current production traffic, but still useful for tracking how inference performance improves as software support matures.

Most benchmark coverage focuses on a configuration's performance at one moment in time. We believe that a launch day performance snapshot is important, yet only looking at launch day performance paints an incomplete picture. A GPU's software stack does not stop improving after launch. Over the useful life of a model, serving frameworks add support, kernels improve, quantization and speculative decoding implementations mature, and better parallelism strategies become available. As a result, the throughput that a GPU can deliver on day zero may look very different from what the same GPU can deliver several months later.

A more complete way to evaluate inference performance is to look at the integral of performance over the entire model lifecycle. In other words, instead of only asking how many tokens a GPU can generate today, we should ask how many tokens it could generate from the day a model is released until the end of that model's useful life. This can be represented as the area under the throughput curve over time:



Source: SemiAnalysis

Here, Q(t,s) is the highest validated throughput available at time t while meeting a fixed interactivity target s. Throughput is zero until a production-ready configuration becomes available. t_0 is the date that a production ready configuration first becomes available, while t_1 is the end of the period being analyzed. Every software optimization raises the throughput curve from the day it becomes available for the remainder of the model's lifecycle, increasing the total token generation capacity the same deployed hardware can provide.

This framing captures two things that a point-in-time benchmark misses. First, time to first inference matters. Every day between a model's release and the availability of a production ready serving stack is a day when the hardware cannot generate tokens for that model. Second, the rate of improvement matters. A platform with lower performance on day zero may still produce more tokens over the model lifecycle if its software stack improves more quickly. Conversely, a strong launch day result may be less impactful if the platform takes longer to receive optimizations.

QQ:1413221435

However, maximum throughput cannot be used in isolation. Increasing concurrency can raise the total number of tokens per second while also degrading the experience for each user. To make the comparison meaningful, we hold end-to-end interactivity constant and select the highest throughput result that also satisfies the same latency target at each point in time. This allows us to compare the area under each platform's performance curve without rewarding configurations that generate more tokens by making users wait longer.

InferenceX produces discrete benchmark results rather than a continuous performance curve, so we approximate the integral as a sum over the periods between measurements:

$$T_{\text{lifecycle}}(s) \approx \sum_{i=1}^{n} Q_i(s)\,\Delta t_i$$

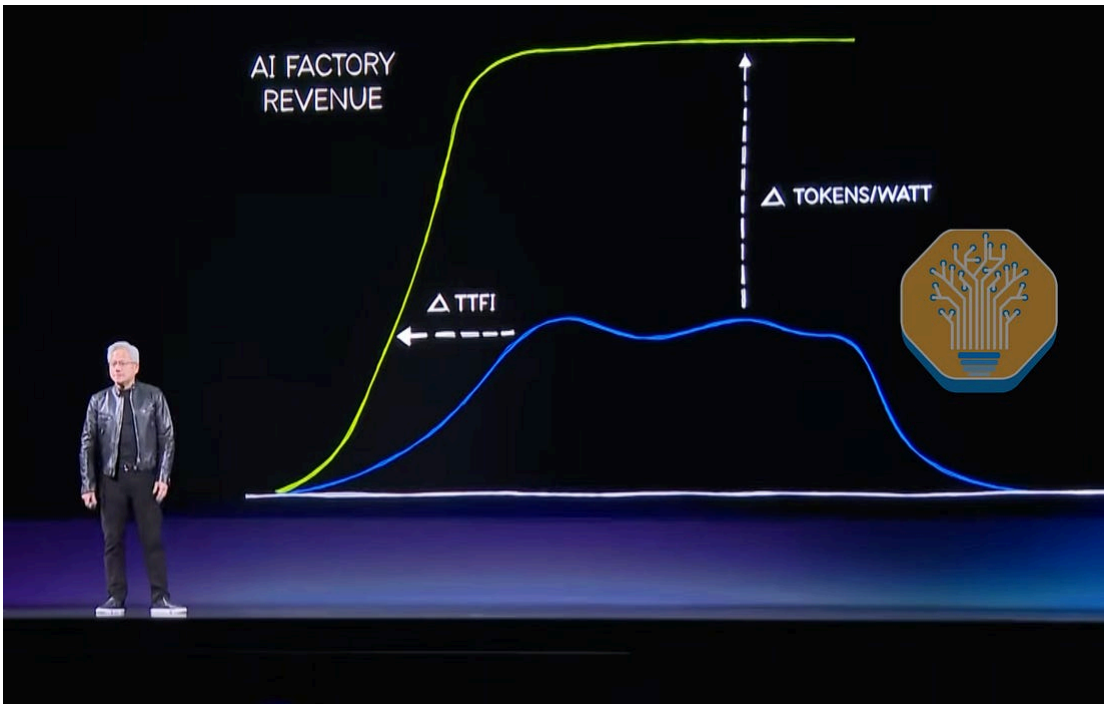
Source: SemiAnalysis

Each validated result is treated as the best performance available from the date it landed until a newer validated result replaces it. This avoids retroactively crediting a platform with an optimization months before it existed, and it also avoids judging deployed hardware permanently by its day zero software stack. For each comparison we use the same t_1 cutoff date across all platforms, since a successor model launching does not immediately eliminate demand for the previous one.

The integral describes token generation capacity assuming continuous operation at the measured throughput. However, real deployments will always produce fewer tokens, since real world numbers depend on utilization, demand, downtime, and workload mix. Due to this, the lifecycle integral is more useful for comparing platforms under identical assumptions rather than predicting the output of any particular fleet.

The same curve can also be expressed in other terms. Instead of integrating throughput, we can integrate cost per million tokens or tokens per megawatt at the same fixed interactivity target. Cost per million tokens includes the hardware cost basis, power, and utilization assumptions, which is closer to the numbers operators actually need. Tokens per megawatt are also valuable data points because the immediate constraint is power, rather than capital, though admittedly capital needs are swelling to the point where it could start becoming a binding constraint as well. Applying a token price to that curve helps us calculate revenue per megawatt.

QQ:1413221435
QQ:1413221435



Source: Nvidia

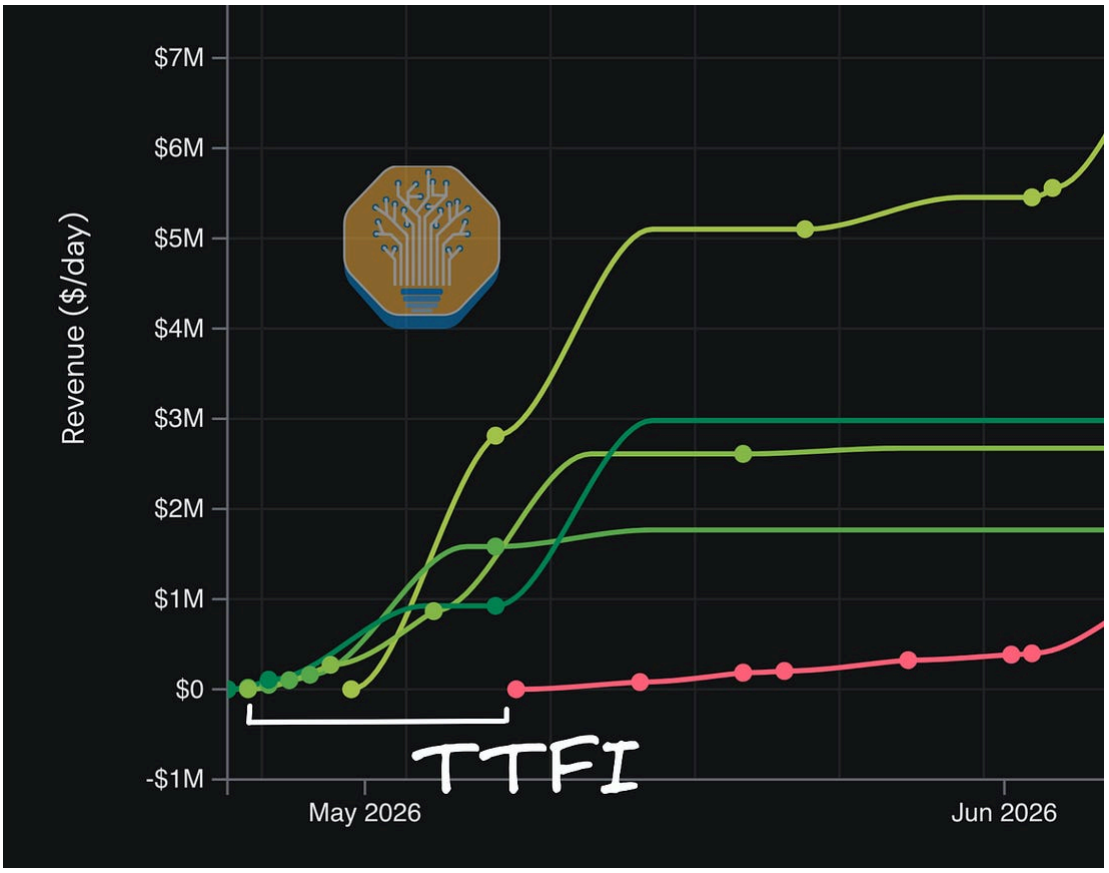
QQ:1413221435

This graph above was shown during Jensen’s 2026 Computex keynote, plotting revenue against time. For InferenceX, we built up the same plot using our configs with time of submission and an artificial ramp time. On DeepSeek v4 8k1k, assuming full utilization, a ramp up duration of 1 week, a 10MW power budget, and OpenRouter DSv4 Pro 0813 prices, revenue are maximized when going with a GB300.

This graph above was shown during Jensen’s 2026 Computex keynote, plotting revenue against time. For InferenceX, we built up the same plot using our configs with time of submission and an artificial ramp time. On DeepSeek v4 8k1k, assuming full utilization, a ramp up duration of 1 week, a 10MW power budget, and OpenRouter DSv4 Pro 0813 prices, revenue is maximized when going with a GB300.

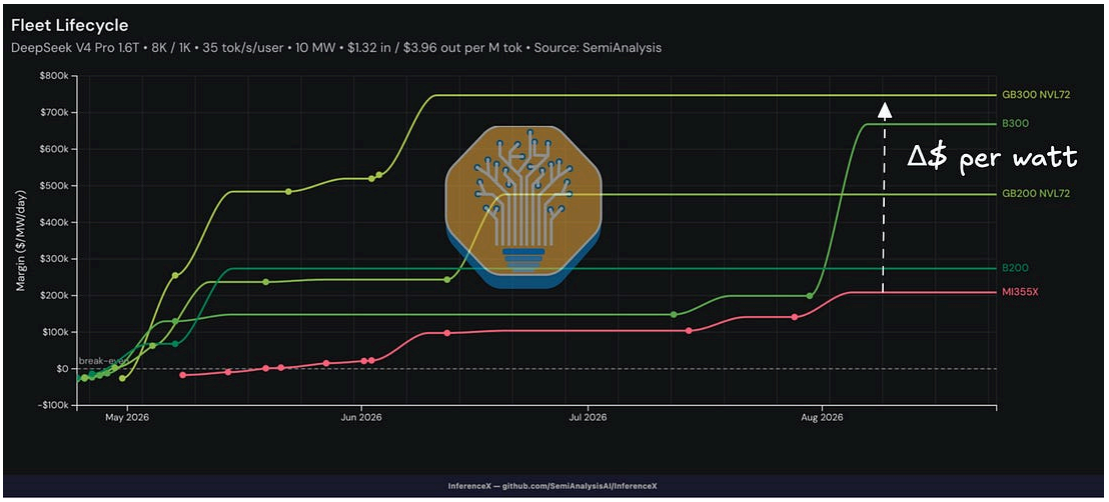
From the graph, we see B200/B300 having support first, and a ΔTTFI of 1 week between that and GB300 support for this particular model. It was only after another week that MI355x support was released. This gave a ΔTTFI of 2 weeks between MI355x and B200/B300.

QQ:1413221435

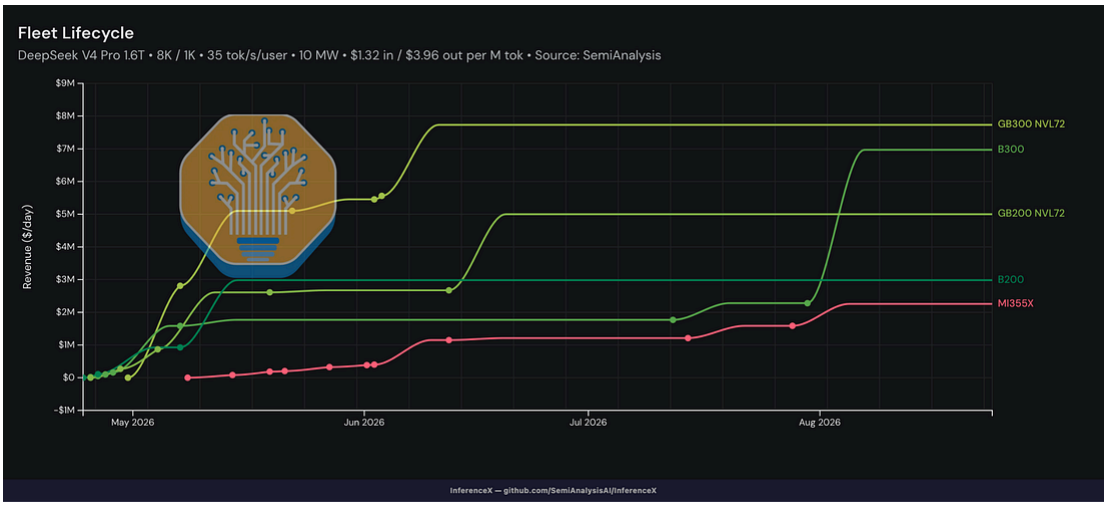


Source: SemiAnalysis InferenceX

Plotting the margin/MW/day gives us Δ$ per watt, showing GB300's performance with power considered as well, more than doubling the Δ$ per watt of MI355x.



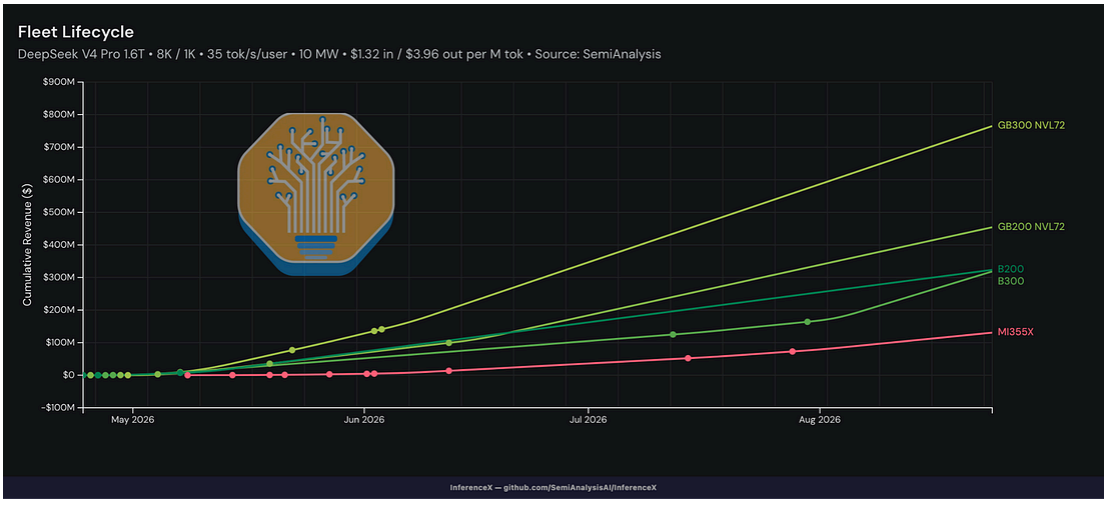
QQ:1413221435
QQ:1413221435



Source: SemiAnalysis InferenceX

Taking the area under the curves gives us accumulated revenue. Even though GB300 had a TTFI longer than the other Nvidia chips, accumulated revenue quickly caught up for the GB300 due to fast software development and lifecycle profit is decisively higher for the GB300 NVL72.

QQ:1413221435



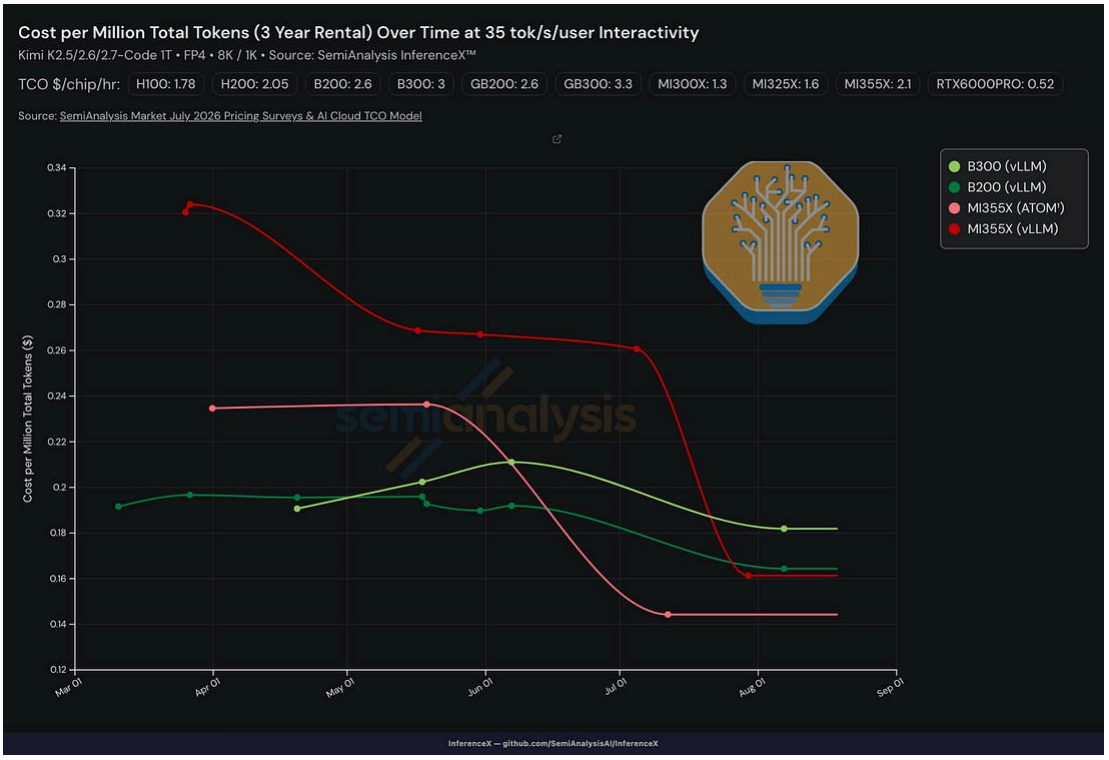
Source: SemiAnalysis InferenceX

In all graphs, GB300 performs the best, a key proof point showing that the rack scale architecture is a decisive advancement, with its larger scale-up world size enabling wideEP and tilting the market towards favoring other optimizations such as disaggregation. Read our InferenceXV2 article to find out more about these optimizations

You can explore these curves on the InferenceX TCO Calculator and Historical Trends pages, which plots interpolated metrics over time at a fixed interactivity operating point. Both the deprecated fixed sequence length scenarios, and the new agentic traces scenarios are available, along with throughput per chip, throughput per

QQ:1413221435

all in utility megawatt, and cost per million tokens across several cost bases including owning as a hyperscaler, owning as a Neocloud, and three year rental.



Source: SemiAnalysis InferenceX

Through this lens, the fixed sequence length archive shows information that a single launch day benchmark cannot. Some configurations improved several fold after release as kernels, quantization, and parallelism support matured, while others arrived late and never recovered the ground lost during that gap.

Next we'll take a quick look at some case studies among several submitted configs by AMD and Nvidia using this integral token generation comparison method.

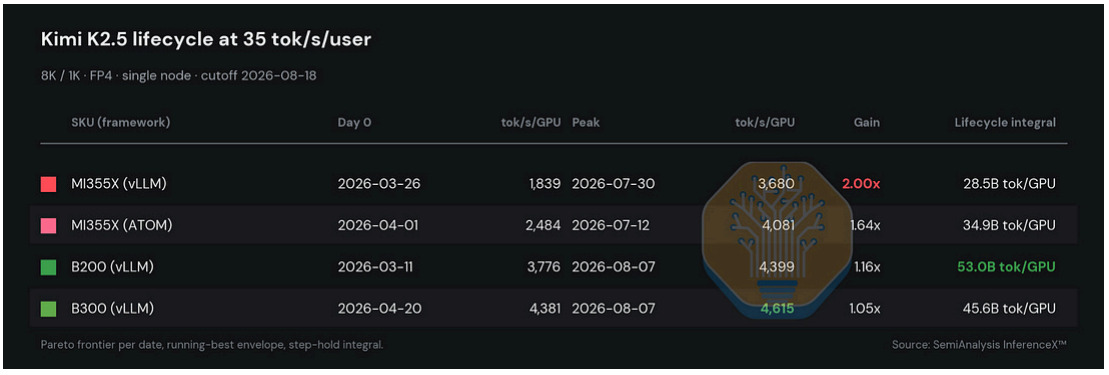
Kimi K2.5 is the clearest case in the archive, since it stayed in testing from February through early August. All numbers below are single node 8k1k at 35 tok/s/user median interactivity, with the lifecycle integral taken from each platform's first validated result through a common August 18, 2026 cutoff.

Our first results were for B200, they landed February 24, 2026 at 1,104 tok/s/GPU, and by March 11 vLLM v0.17.0 had taken it to 3,776, a 3.4x jump inside two weeks. It then sat flat for five months, finishing at 3,813 on August 7. MI355X support arrived 30 days later on March 26, debuting at a throughput of 1,431 tok/s/GPU, with this throughput climbing gradually to 2,873 tok/s/GPU on vLLM ROCm nightlies by July 30.

QQ:1413221435

On July 12, an ATOM FP4 configuration reached 4,081 tok/s/GPU on MI355X, beating every B200 result we have for this scenario. The same holds at a tighter latency target, where MI355X peaks at 2,261 against 2,083 for B200 at 75 tok/s/user. ATOM is still early and has yet to serve production tokens, so we flag it separately from the vLLM numbers rather than folding it into the same claim.

However, lifecycle integrals tell a different story. B200 accumulates 53.8B tokens per GPU against 35.2B for MI355X, and B300 lands at 45.6B off a single strong launch day result of 4,381 on April 20. Judging by the integral analysis, B200 wins not because it was faster at the end, but because it got there earlier in February and held the line for six months. That 30 day head start plus delivering five solid months at a flat 3,800 is worth more than debuting later with a higher number in July.

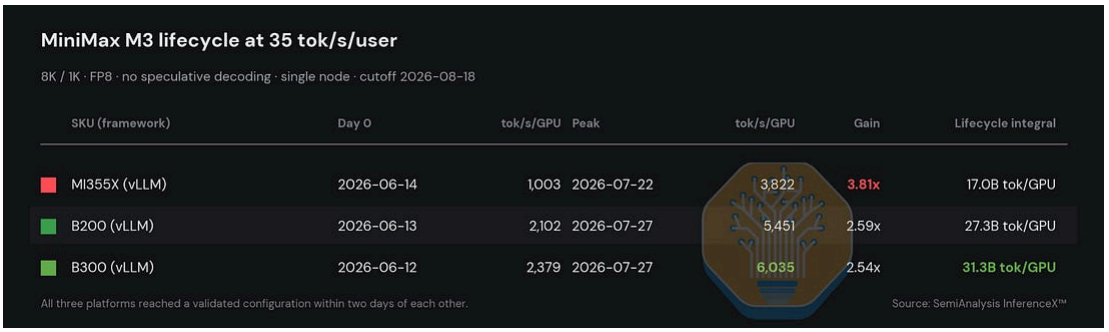


Source: SemiAnalysis InferenceX

Minimax M3 is a better demonstration of what fast software improvement looks like. Both vendors started on the same day having worked overnight to put together high quality recipes. On June 13, MI355X measured 1,072 tok/s/GPU at 35 tok/s/user and B200 measured 1,890 tok/s/GPU. Three weeks later, MI355X hit 8,662 on July 3, an 8.1x improvement, while B200 reached 8,945 on July 6 for a 4.7x improvement.

In this case study, the lifecycle integrals actually end up nearly tied at 42.7B tokens per GPU for MI355X against 42.4B for B200, with B300 nicely ahead at 46.2B. MI355X starting 1.8x behind but finishing with nearly equal token generation is their day zero effort paying off. It took the implementation of several optimizations including disaggregation, FP8 cache dtype, and EAGLE3 parity fixes described in the optimization section above to deliver this performance. However, their work does not hold at every latency target: at 50 and 75 tok/s/user B200 stays clearly ahead, at 43.7B and 36.9B respectively.

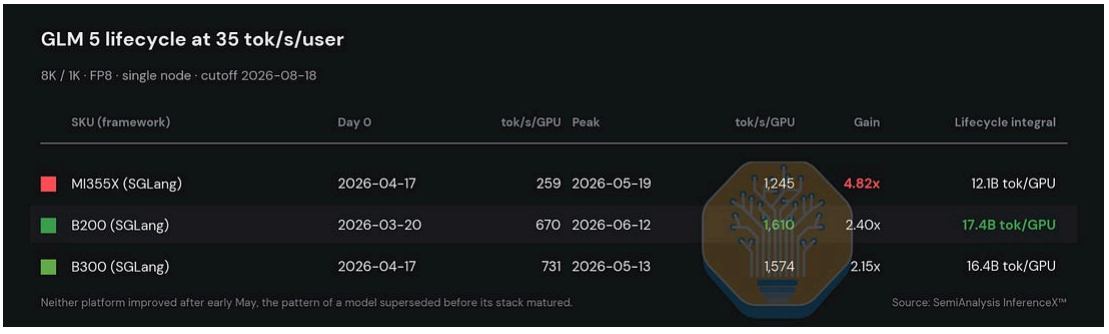
QQ:1413221435
QQ:1413221435



Source: SemiAnalysis InferenceX

With GLM 5 FP8, B200 opened March 20 at 670 tok/s/GPU at 35 tok/s/user and doubled to 1,335 tok/s/GPU by April 18. MI355X opened April 8 at 665 tok/s/GPU, almost exactly B200's launch number, and reached 954 tok/s/GPU by May 1. Integrals come out at 15.7B for B200, 15.3B for B300, and 10.3B for MI355X. Neither platform kept improving after early May, which is a pattern we commonly see when a model gets superseded before its software stack matures.

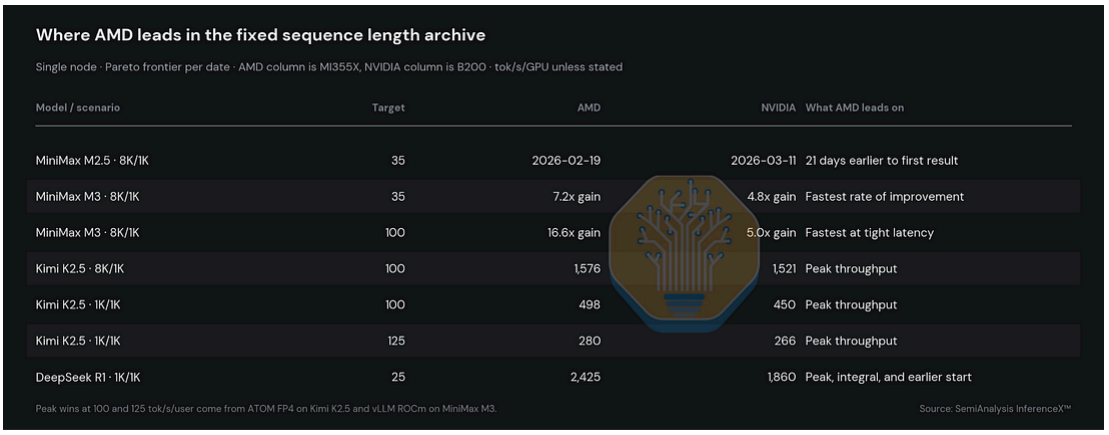
It is worth separating three claims that are easy to conflate. A higher result on a given date, earlier production readiness, and a larger lifecycle integral are three different advantages, and a platform can have one without the others. On Kimi K2.5, MI355X holds the peak while B200 holds both the earlier start and the larger integral. On MiniMax M3 both started together however MI355X improved faster, and the integrals finished nearly level.



Source: SemiAnalysis InferenceX

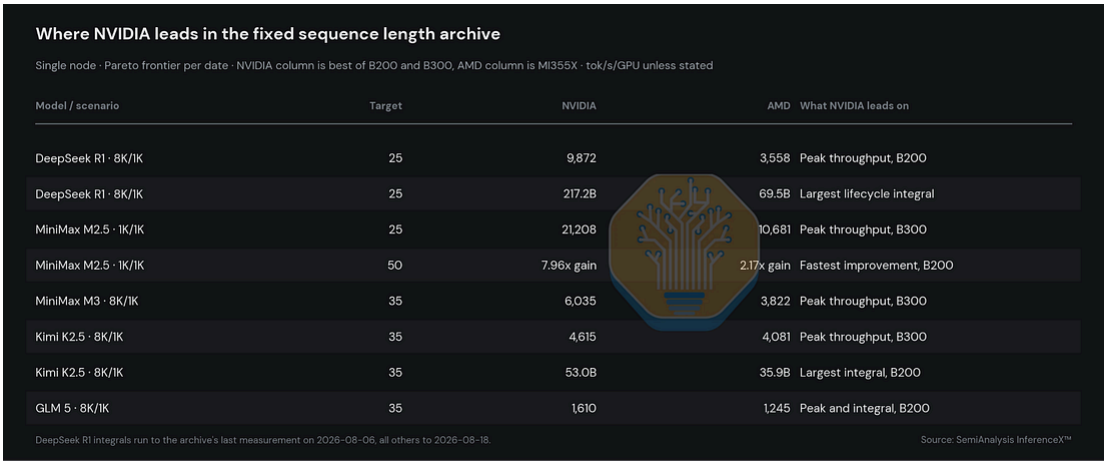
Across the wider archive the same three shapes keep recurring, and neither vendor owns all of them at once. We swept every retired model at several interactivity targets and pulled out the configurations where each platform leads.

QQ:1413221435
QQ:1413221435



Source: SemiAnalysis InferenceX

The reverse view is equally instructive, since Nvidia's leads tend to come from a different source.



Source: SemiAnalysis InferenceX

As AMD and Nvidia submit configurations, we will continue to congratulate them and push them forward. Our goal is not to declare a winner on a single date. As in inference, staying in the lead is a combination of time to first inference, performance at a fixed level of interactivity, the rate of software improvement, performance per megawatt, and how long a model remains worth serving. A platform that wins on launch day does not necessarily produce the most useful tokens over the life of the model, which is exactly why we benchmark continuously.

Single Turn Performance

As mentioned in preceding sections, the main scenario is now AgentX. However, while developing AgentX, we still had open single turn 8k1k submissions. But now since AgentX is live, we have deprecated most of the 8k1k single turn runs now. We will highlight some of the improvements in this section, mainly focusing on

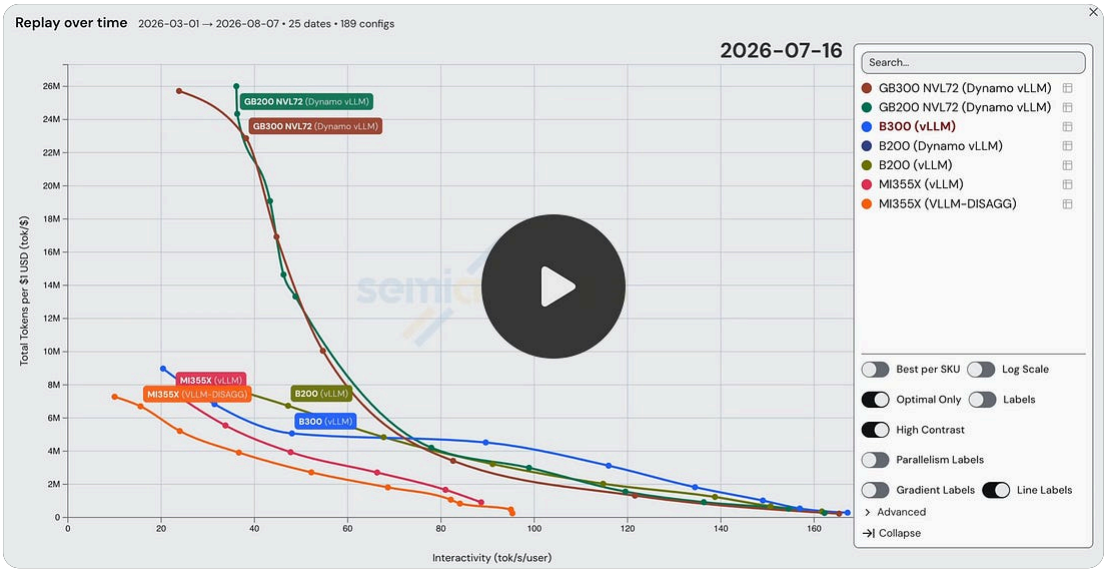
QQ:1413221435

performance over time, so that the reader can understand the **trajectory** and **momentum** of different serving stacks.

Kimi K2.X

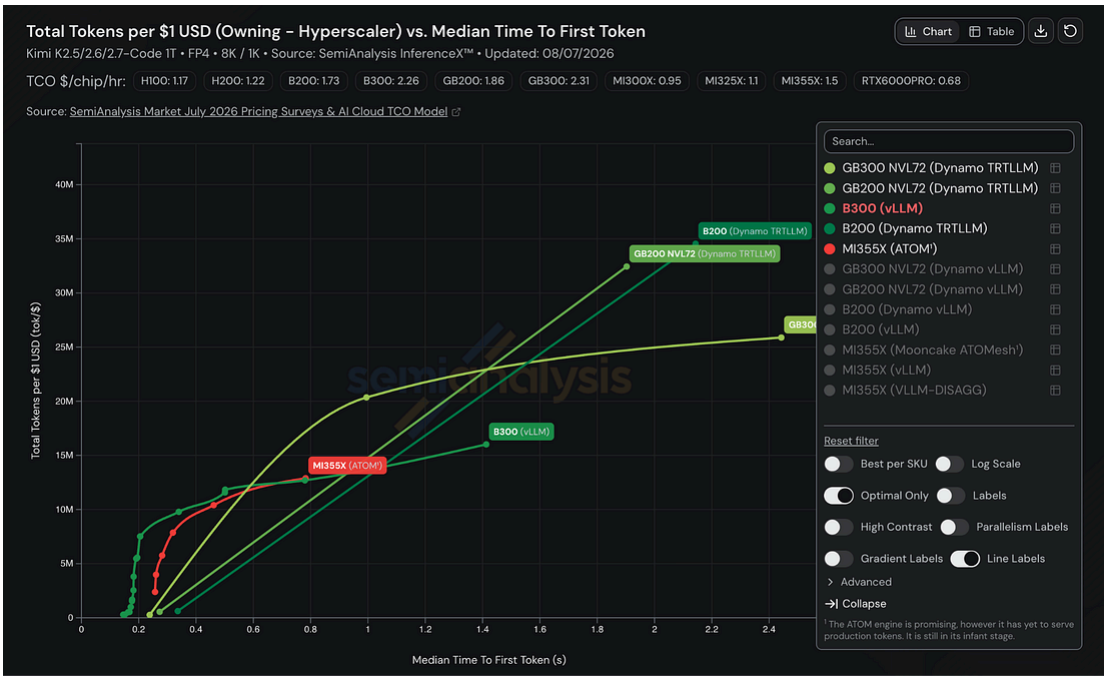
Kimi K2.X InferenceX results started streaming on February 18, 2026 and were continuously updated until August 7, 2026. At this point, the model was deprecated in favor of Kimi K3 AgentX submissions. During this period, we received sixty 8k1k submissions, each of which carried a performance improvement through various optimizations to the inference software.

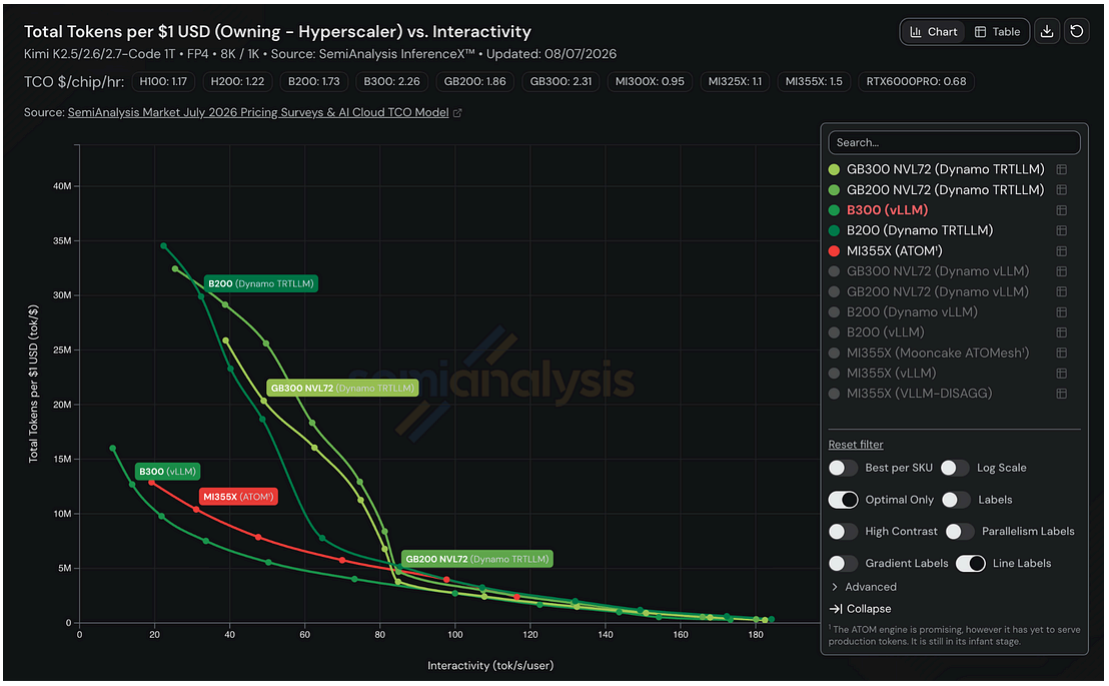
Here is a clip of all of the performance enhancements on vLLM:



At the time of deprecation, this was the state of the best Kimi K2.5 configurations across all SKUs and engines. MI355X ATOM is quite competitive against B300 vLLM, when measuring total tokens per USD (chip performance normalized by cost).

QQ:1413221435
QQ:1413221435

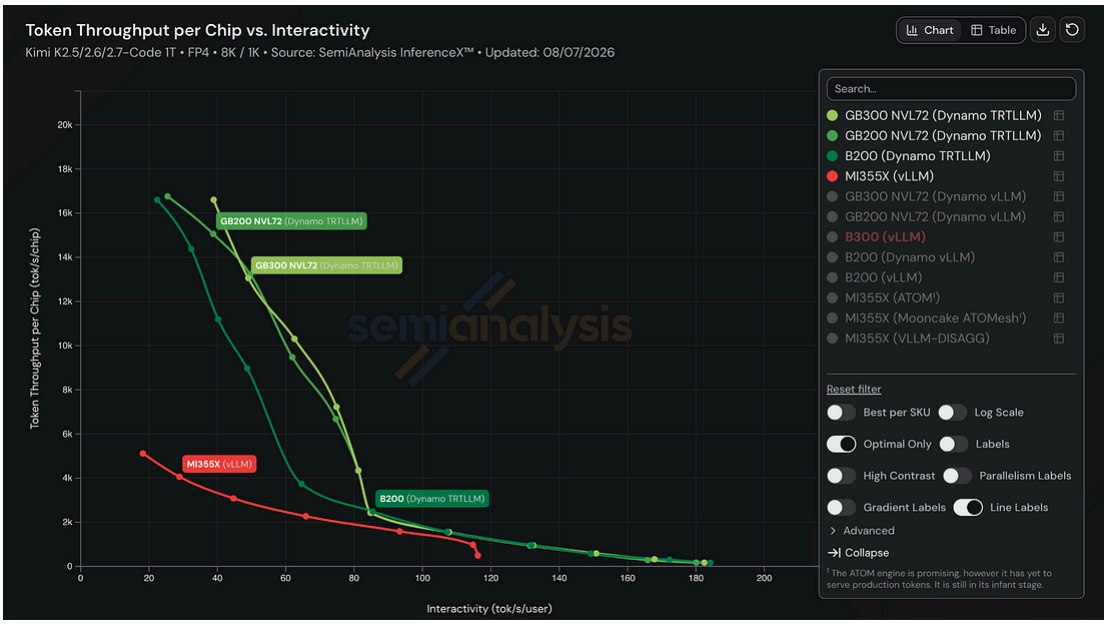


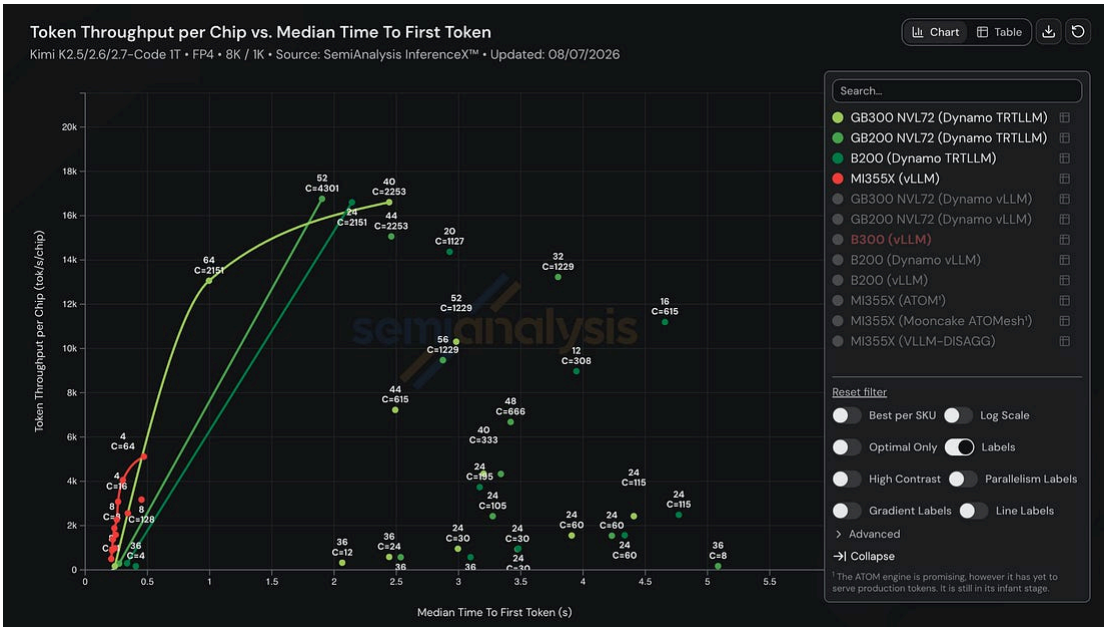


Source: [SemiAnalysis InferenceX]

The above charts highlight why it is important to consider *both* interactivity *and* TTFT charts together. We sometimes see configurations which over optimize for interactivity at the cost of latency. For instance, many of the Nvidia points that fall on the Pareto frontier for throughput versus interactivity don't even come close to throughput vs TTFT optimal.

QQ:1413221435
QQ:1413221435





Source: SemiAnalysis InferenceX

AMD Improvement Over Time

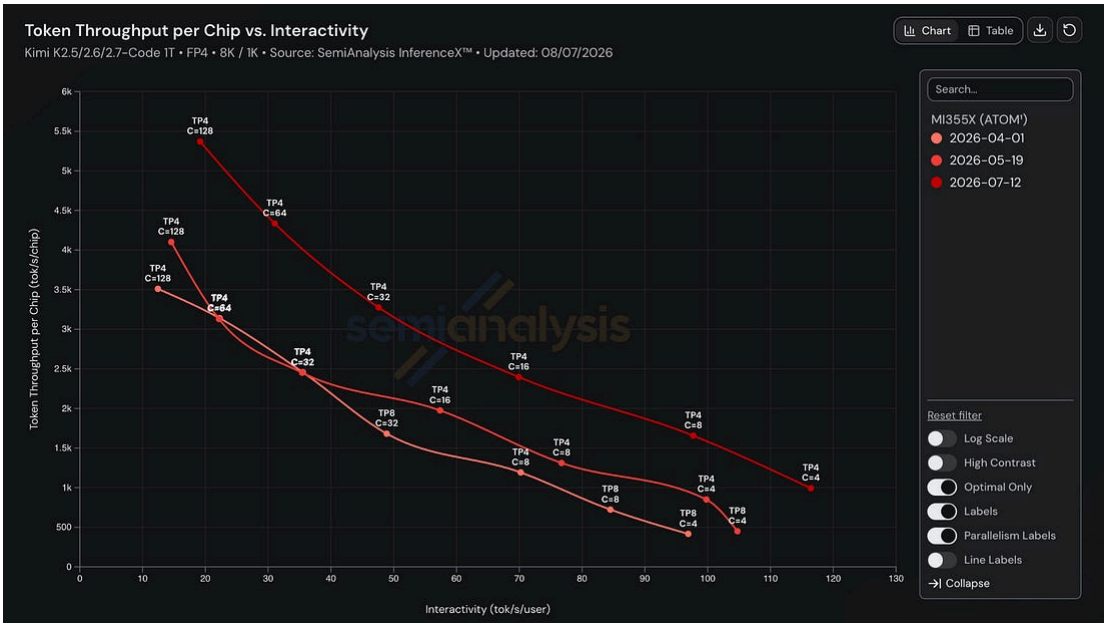
The following graphs show the iterative improvements on MI355X for AMDs ATOM engine.

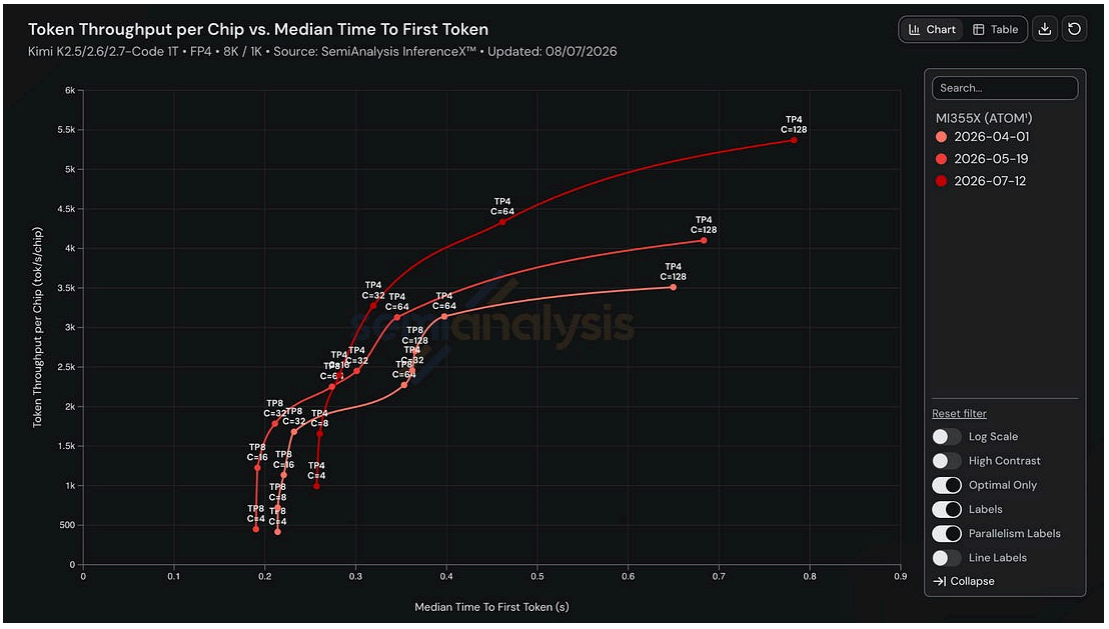
The first image update refreshed ROCm, PyTorch, and ATOM. The next added a faster FP4 MoE backend, lower-overhead expert routing and all-reduce, and FP4 intermediates. These changes reduce MoE kernel time and TP communication during decode, improving throughput and interactivity across the curve. The ATOM recipe also enabled scheduler delay at high concurrency.

Scheduler delay briefly defers new prefills while decode is active, allowing the waiting queue to fill before the next prefill batch. This improves batch efficiency and sustained throughput but adds queueing to TTFT. The high concurrency points show

QQ:1413221435
QQ:1413221435

the precise tradeoff, with throughput and interactivity rising while TTFT worsens. The later sweep also removed TP8, which had provided several of the earlier low TTFT points.

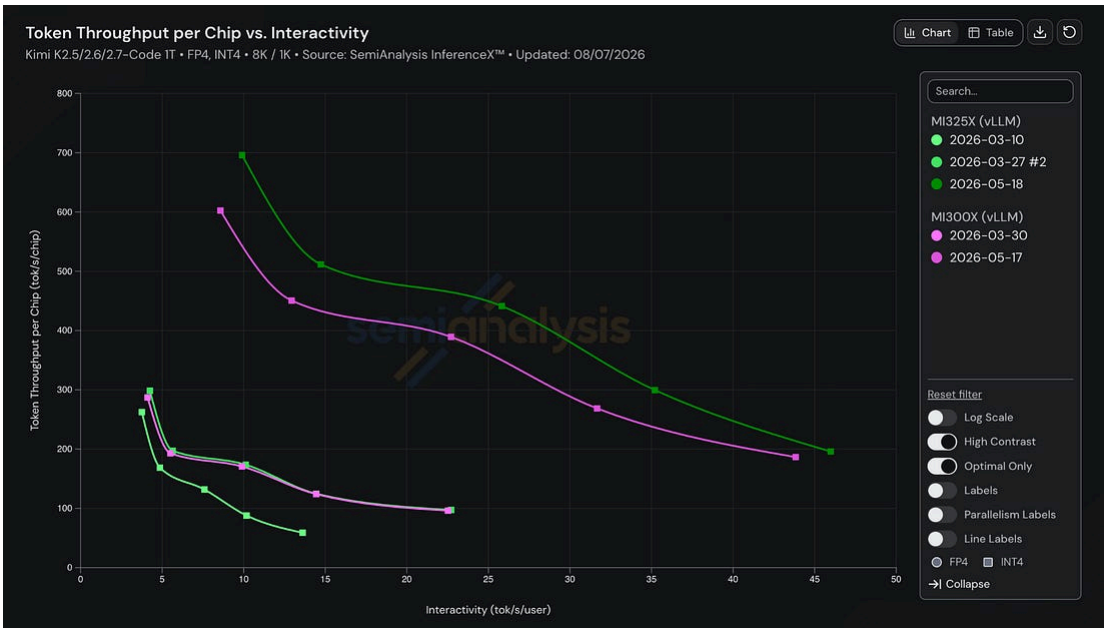


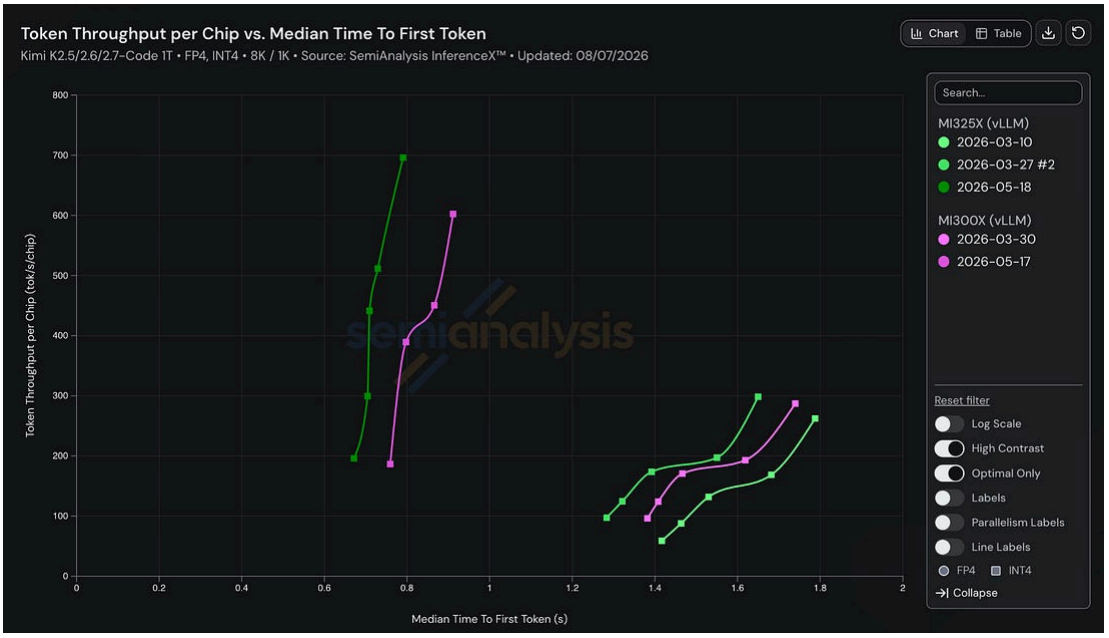


Source: InferenceX

MI325X first improved when the recipe upgraded vLLM ROCm from v0.16 to v0.18, enabled AITER MLA, and raised max-num-seqs to 256. The larger May jump came from the image-only v0.21 upgrades for MI300X and MI325X. The v0.21 logs show the same AITER MLA backend but add fused all-reduce + RMSNorm and fused MLA Q/K RMSNorm; v0.21 also fixes persistent MLA for Kimi-K2.5 TP8 and removes a redundant zero-fill from every MLA decode layer. On 8k/1k, throughput/GPU rises 1.9–2.6× and median TTFT falls 43–53% versus v0.18, so this is a kernel and communication gain rather than a throughput-for-TTFT tradeoff.

QQ:1413221435





Source: InferenceX

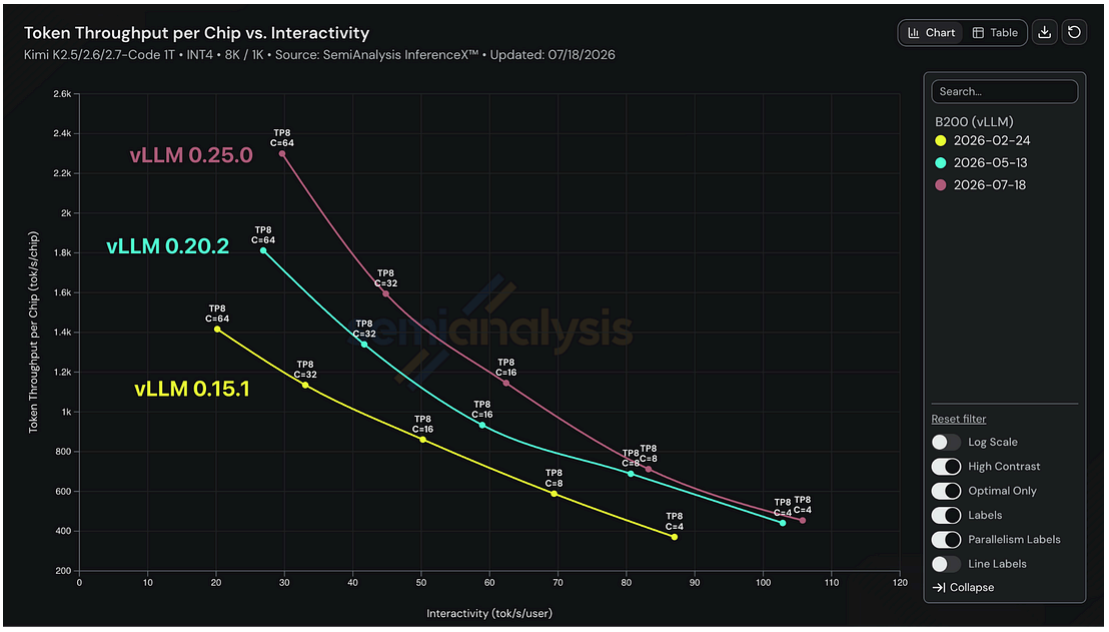
Nvidia Improvement Over Time

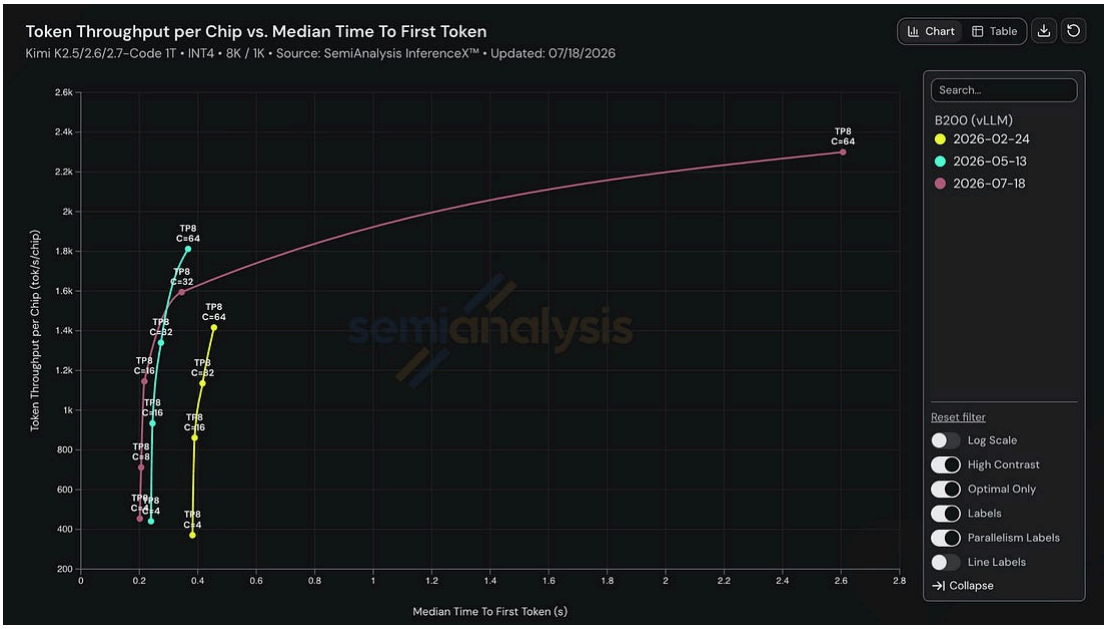
v0.15.1 used Marlin W4A16 MoE and TRT-LLM ragged MLA prefill. Enabling FlashInfer INT4 in the recipe did not change the active kernel because the MXINT4 interface was broken. v0.20.2 is the main step. It fixes that interface, moves the MoE layers onto FlashInfer's TRT-LLM MXINT4 kernel, replaces TRT-LLM MLA prefill with the restored FA4 default for Blackwell chunked prefill, and activates fused FlashInfer all-reduce + RMSNorm. Versus v0.15.1, 8k/1k throughput/GPU rises 8–28% and median TTFT falls 20–37% across concurrency 4–64.

v0.21 and v0.22 keep the same critical kernels and move the curve only marginally. v0.25 pins MLA chunked-prefill metadata for asynchronous H2D copies and switches single-node fused all-reduce from TRT-LLM to MNNVL. At concurrency

QQ:1413221435

4–16, throughput/GPU rises another 2–6% and median TTFT falls 11–13%. Ignore concurrency 32/64. Both logs terminate with the same FA4 MLA output-stride error. The throughput collapse and TTFT spike are failed points, not a valid v0.25 frontier.





Source: InferenceX

DEP8 was added in InferenceX PR #2512 for the August vLLM nightly sweep. It runs attention and KV cache as eight data-parallel engines while sharding the MoE experts across the same eight B200s, removing TP collectives from attention without replicating the full MoE. This targets the throughput end of the curve. At concurrency 512, DEP8 reaches 6.14k tok/s/GPU with 1.15s TTFT, versus ~3.9k tok/s/GPU and ~50s TTFT for TP8/TEP8. Concurrency 1024 reaches 7.66k tok/s/GPU. The recipe lowers GMU from 0.90 to 0.80 to leave memory headroom for the replicated attention/KV path.

QQ:1413221435



Source: InferenceX

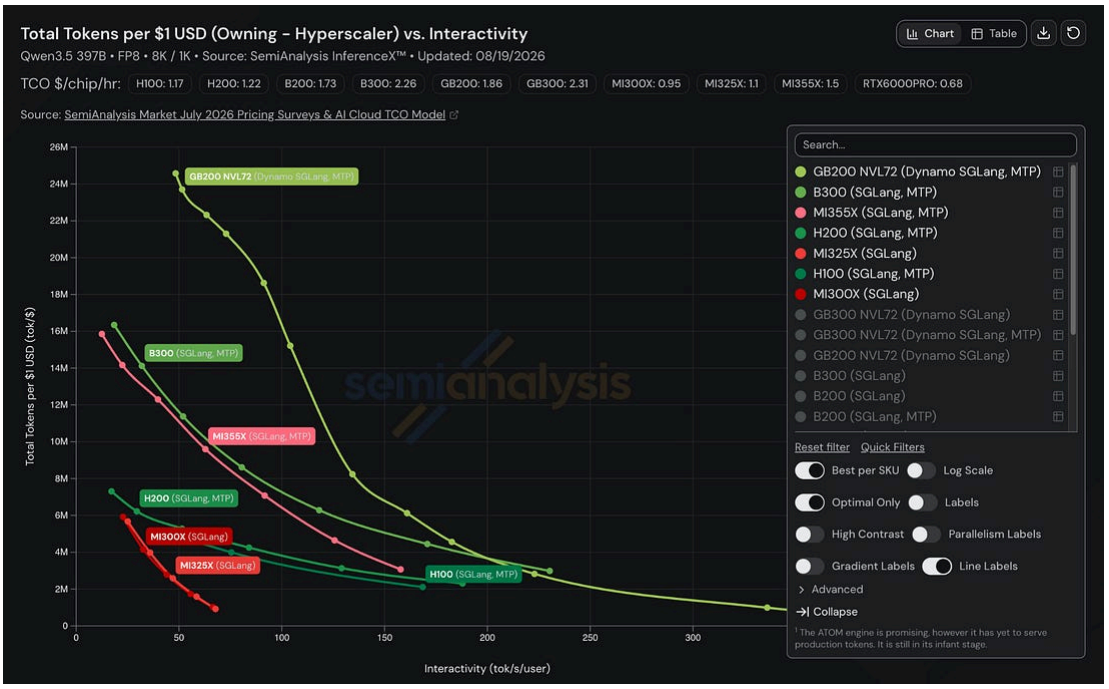
Qwen 3.5

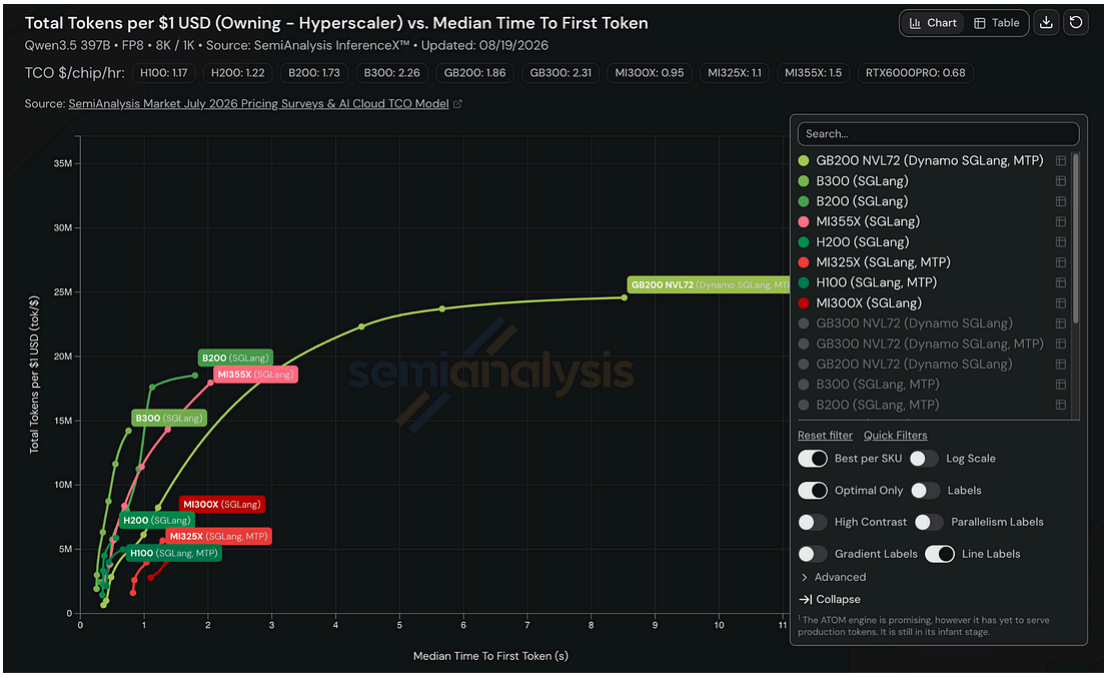
Qwen3.5 8k1k results started streaming on Feb 17, 2026 and continue to be updated. The model has not been deprecated, with the latest validated fixed sequence length rows landing on Aug 19, 2026. During this period we validated 1,028 datapoints across 49 benchmark dates, 97 CI runs, and 42 runtime images. Qwen3.5 was a day-zero launch. vLLM and SGLang merged dense and MoE model support before the public release. B200 posted the first validated rows through InferenceX PR #704, with MI355X following later that day through PR #705. Both initial submissions used SGLang nightly images and BF16 weights. Block-wise FP8 and NVFP4 support followed within two days, opening the later quantized sweeps.



As of August 21, these are the best Pareto frontiers per SKU, normalized by TCO.

QQ:1413221435





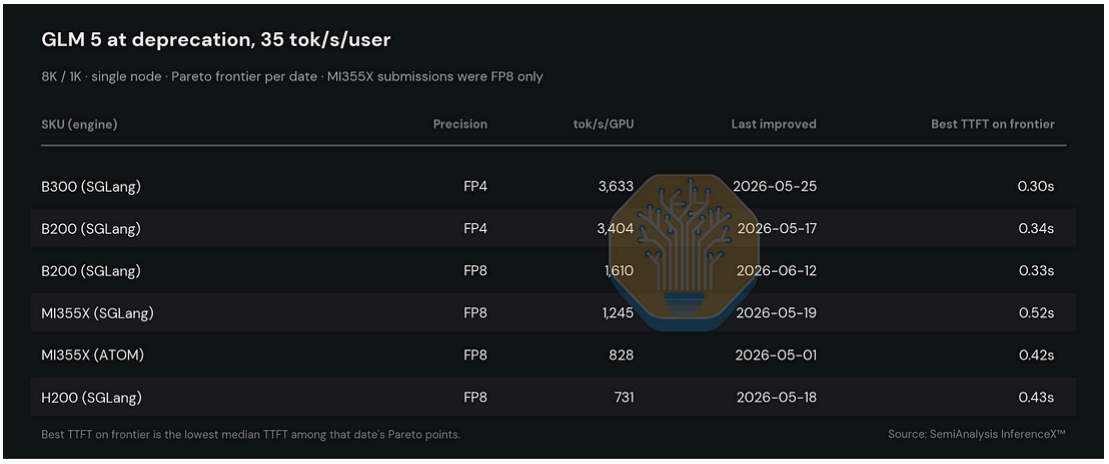
Source: [InferenceX]

GLM 5

Here we focus specifically on the original GLM 5 model, not the later GLM 5.2 release or other models in the GLM family. GLM 5 is now a deprecated model, but its InferenceX history provides a useful comparison between MI355X and B200 because both platforms improved substantially during the evaluation period, while their final positions depended on the interactivity target. At the throughput-oriented end of the curve, MI355X came close to B200. As the target became more interactive, the Nvidia lead widened.

QQ:1413221435
QQ:1413221435

At 25 tok/s/user, MI355X reached 1.7k tok/s/GPU against 1.8k on B200, or about 94% of B200's performance. The gap widened at stricter interactivity targets. MI355X reached about 77% of B200's performance at 35 tok/s/user and about 68% at 75 tok/s/user. MI355X was therefore most competitive when the deployment was tuned for aggregate throughput, while B200 maintained a clear advantage when the target required more responsive per-user performance.



GLM 5 at deprecation, 35 tok/s/user

8K / 1K · single node · Pareto frontier per date · MI355X submissions were FP8 only

SKU (engine)	Precision	tok/s/GPU	Last improved	Best TTFT on frontier
B300 (SGLang)	FP4	3,633	2026-05-25	0.30s
B200 (SGLang)	FP4	3,404	2026-05-17	0.34s
B200 (SGLang)	FP8	1,610	2026-06-12	0.33s
MI355X (SGLang)	FP8	1,245	2026-05-19	0.52s
MI355X (ATOM)	FP8	828	2026-05-01	0.42s
H200 (SGLang)	FP8	731	2026-05-18	0.43s

Best TTFT on frontier is the lowest median TTFT among that date's Pareto points.

Source: SemiAnalysis InferenceX™

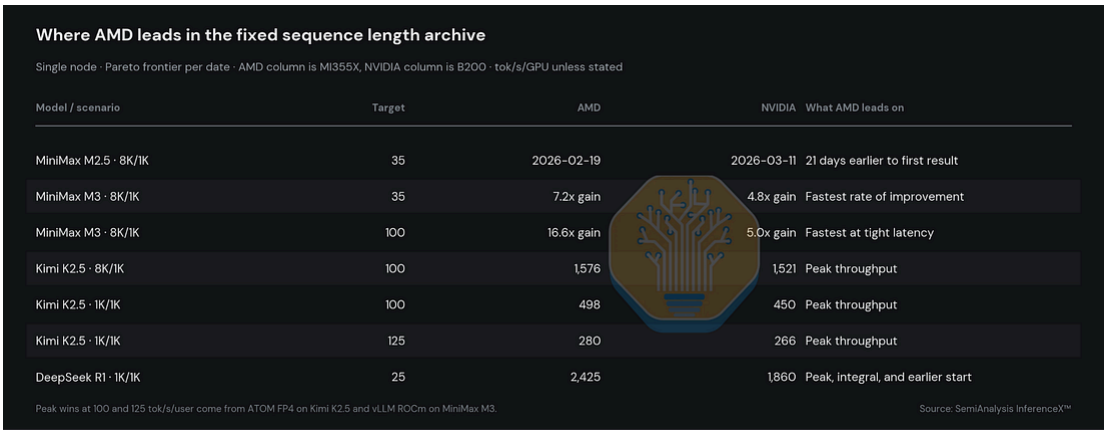
Source: SemiAnalysis InferenceX

Our comparison uses the FP8 SGLang configurations available for both MI355X and B200. There was no AMD FP4 submission for GLM 5, so we use the common FP8 configurations for the hardware comparison rather than mixing FP8 results on AMD with FP4 results on Nvidia. This keeps the comparison consistent and avoids attributing an unsubmitted AMD configuration to either platform.

AMD improved by about 4.8× from its initial GLM 5 result to its best result before the model was deprecated. This was the largest relative improvement from either vendor. The progression came through two major recipe changes rather than a smooth sequence of smaller gains. ATOM opened ahead of AMD's SGLang results, but then plateaued as SGLang continued improving and eventually passed it.

The AMD curve also shows a dip after TP8 was removed from the sweep. This did not necessarily mean that the hardware became slower. It meant that the later sweep no longer included the same TP8 configurations, including some of the earlier low-TTFT points, so the visible frontier changed.

QQ:1413221435
QQ:1413221435



Where AMD leads in the fixed sequence length archive

Single node · Pareto frontier per date · AMD column is MI355X, NVIDIA column is B200 · tok/s/GPU unless stated

Model / scenario	Target	AMD	NVIDIA	What AMD leads on
MiniMax M2.5 · 8K/1K	35	2026-02-19	2026-03-11	21 days earlier to first result
MiniMax M3 · 8K/1K	35	7.2x gain	4.8x gain	Fastest rate of improvement
MiniMax M3 · 8K/1K	100	16.6x gain	5.0x gain	Fastest at tight latency
Kimi K2.5 · 8K/1K	100	1,576	1,521	Peak throughput
Kimi K2.5 · 1K/1K	100	498	450	Peak throughput
Kimi K2.5 · 1K/1K	125	280	266	Peak throughput
DeepSeek R1 · 1K/1K	25	2,425	1,860	Peak, integral, and earlier start

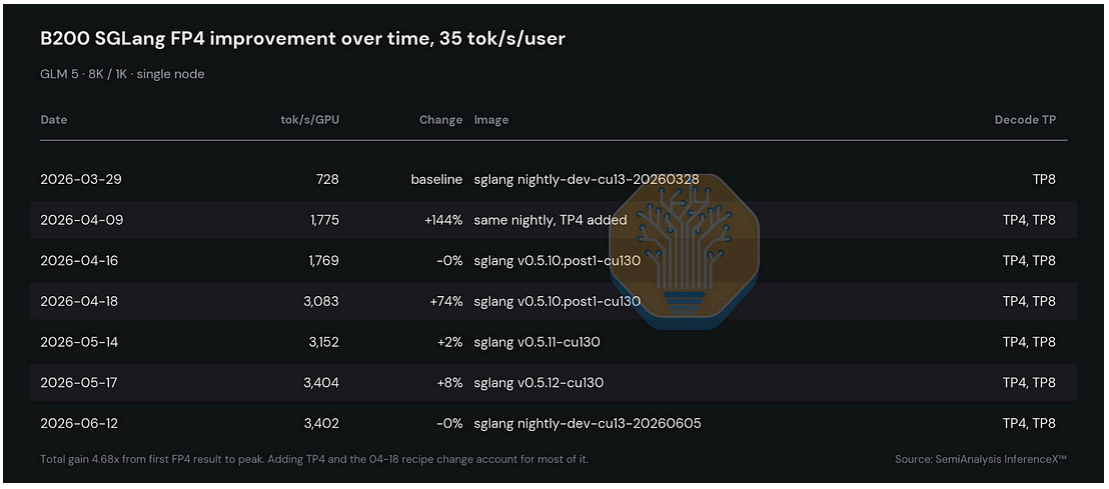
Peak wins at 100 and 125 tok/s/user come from ATOM FP4 on Kimi K2.5 and vLLM ROCm on MiniMax M3.

Source: SemiAnalysis InferenceX™

Source: SemiAnalysis InferenceX

In the meantime, Nvidia improved by about 4.68×, effectively matching AMD's rate of improvement. One of the biggest contributors to Nvidia's improvement was the move to TP4, after which the curve reached a plateau. B300 produced the highest Nvidia result, although that result came from an FP4 configuration rather than the common FP8 comparison. It demonstrates the additional performance available from Nvidia's FP4 stack, but it should not be compared directly with the FP8 MI355X result.

QQ:1413221435



B200 SGLang FP4 improvement over time, 35 tok/s/user

GLM 5 · 8K / 1K · single node

Date	tok/s/GPU	Change	Image	Decode TP
2026-03-29	728	baseline	sglang nightly-dev-cu13-20260328	TP8
2026-04-09	1,775	+144%	same nightly, TP4 added	TP4, TP8
2026-04-16	1,769	-0%	sglang v0.5.10.post1-cu130	TP4, TP8
2026-04-18	3,083	+74%	sglang v0.5.10.post1-cu130	TP4, TP8
2026-05-14	3,152	+2%	sglang v0.5.11-cu130	TP4, TP8
2026-05-17	3,404	+8%	sglang v0.5.12-cu130	TP4, TP8
2026-06-12	3,402	-0%	sglang nightly-dev-cu13-20260605	TP4, TP8

Total gain 4.68x from first FP4 result to peak. Adding TP4 and the 04-18 recipe change account for most of it.

Source: SemiAnalysis InferenceX™

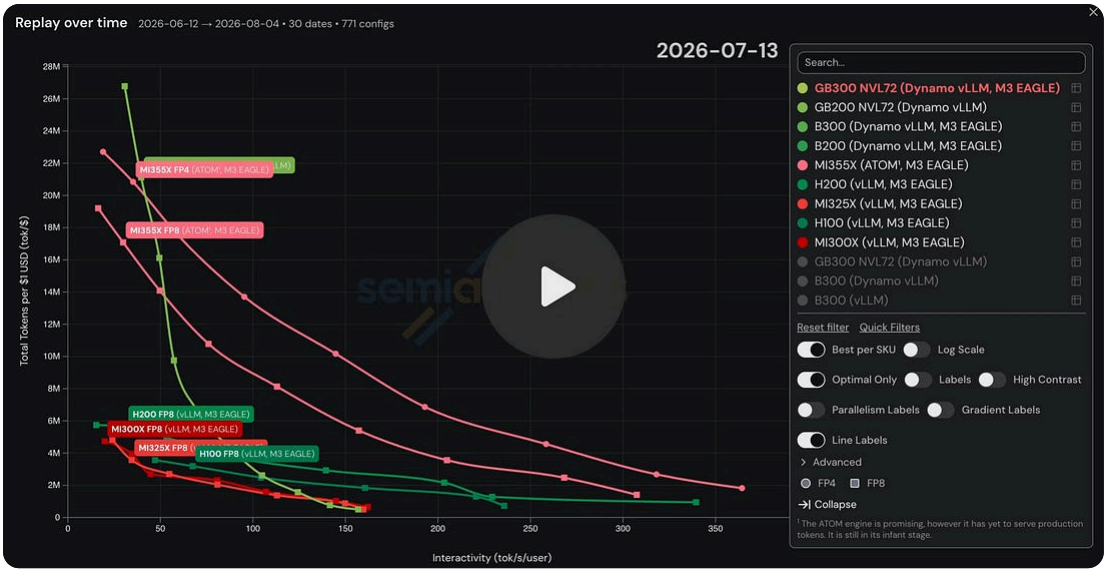
Source: SemiAnalysis InferenceX

The GLM 5 frontier also shows why throughput cannot be evaluated independently from TTFT. The highest-throughput configurations are not automatically the most useful interactive deployment points, and some points trade faster aggregate generation for worse first-token latency. The results also reflect how consistently each software stack composes its available optimizations at different operating points. MI355X was most competitive at the throughput end of the curve, while B200 maintained a larger advantage at stricter interactivity targets. The best configuration therefore depends on whether the deployment is optimized for aggregate throughput or per-user responsiveness.

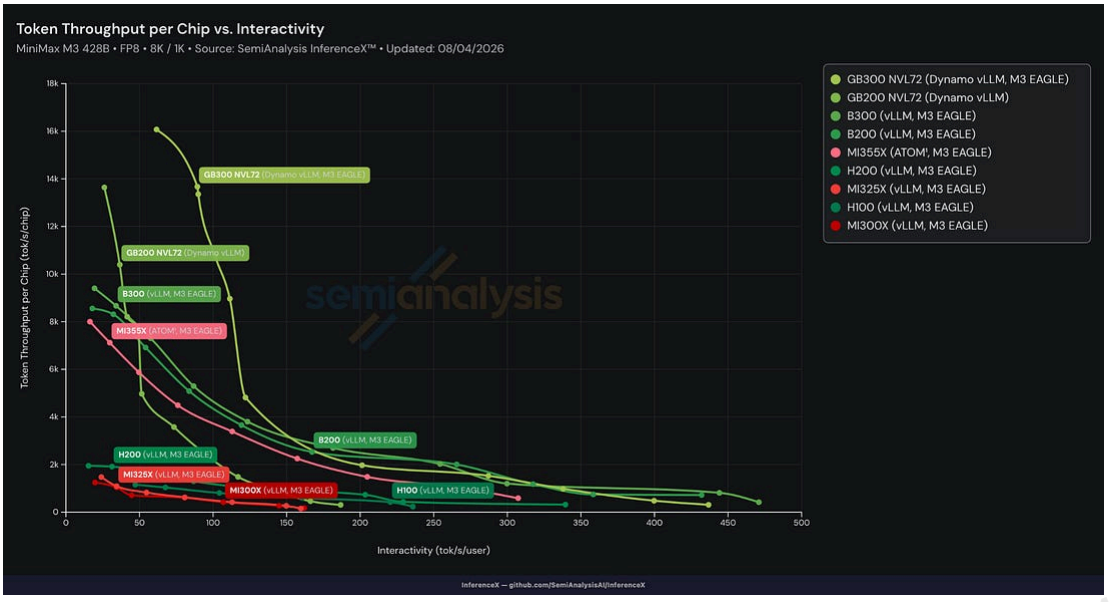
QQ:1413221435

MiniMax M3

MiniMax M3 8k1k results started streaming on June 12, 2026 and continuously updated until August 4, 2026, when fixed sequence length submissions wound down in favor of AgentX M3 agentic traces. During this period we validated 1,977 datapoints across 30 benchmark dates, 89 CI runs, and 25 runtime images. M3 was a day-zero race: vLLM shipped day-0 support with deployment containers for both Nvidia and AMD alongside an open EAGLE3 speculator head, and B300 posted the first validated rows on June 12 with B200 and MI355X following a day later.



At the time of deprecation, this was the state of the best M3 configurations across all SKUs. GB300 leads outright at Dynamo-vLLM MTP disaggregated configuration.



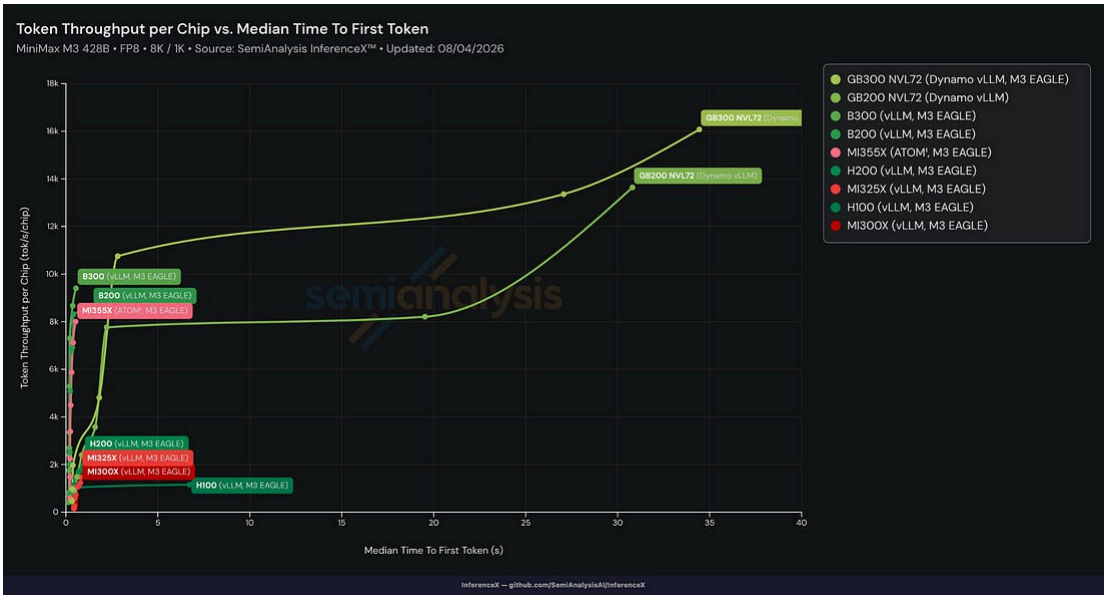
Source: SemiAnalysis InferenceX

As mentioned earlier, throughput-interactivity cannot be seen in isolation. Three of the headline results above ship with median TTFTs that no interactive deployment

QQ:1413221435

would accept, a very different picture from the disagg-dominated headline ordering.

As mentioned earlier, throughput-interactivity cannot be seen in isolation. Three of the headline results above ship with median TTFTs that no interactive deployment would accept, a very different picture from the disagg-dominated headline ordering.



Source: SemiAnalysis InferenceX

QQ:1413221435



RyanLee
@RyanLeeMiniMax

AMD MI355X now is close to Nvidia B200 on MiniMax-M3 serving. Thanks to @AMD team. released dual technical blogs detailing full-stack ATOM + ATOMesh co-optimization for our 428B MSA sparse MoE multimodal base model. • EAGLE3 speculative decoding for lower generation



QQ:1413221435

4:05 AM · Jul 23, 2026 · 55.4K Views

14 Replies · 14 Reposts · 118 Likes

Like Comment Restack

© 2026 Dylan Patel
548 Market Street PMB 72296, San Francisco, CA 94104
Unsubscribe

Start writing

QQ:1413221435

QQ:1413221435