DeepSeekV4 1.6T Day 0 to Day 43 Performance Over Time - Huawei, GB300 NVL72, MI355X, B200

Day 0 Inference Performance, InferenceX, 100x performance improvement in 26 Days, Cost per Million Tokens, Huawei 950DT Inference Trace Analysis

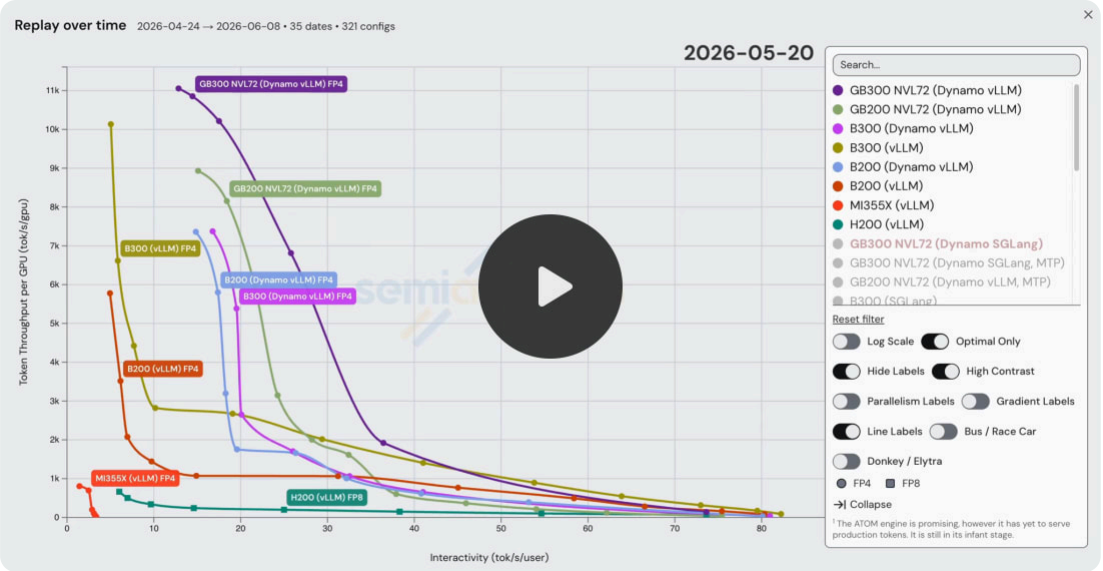
JUN 9 https://newsletter.semianalysis.com/p/deepseekv4-16t-day-0-to-day-43-performance

The release of DeepSeek v4 marks another step forward for the open model community - unsurprisingly, it is the product of a Chinese lab. The evolution of its performance over time is of paramount importance to the AI Ecosystem. The open-source InferenceX engineering team has pulled multiple all-nighters to measure performance results for this model on Day 0, Day 1, Day 2, and beyond and bring these results to the world. In this article, we will highlight DeepSeek v4's Day 0 performance as and explain the significant improvements made the subsequent weeks following the model's release. We will also explain core components of DeepSeek v4's model architecture and discuss how it was co-designed in part for Huawei Ascend inference.

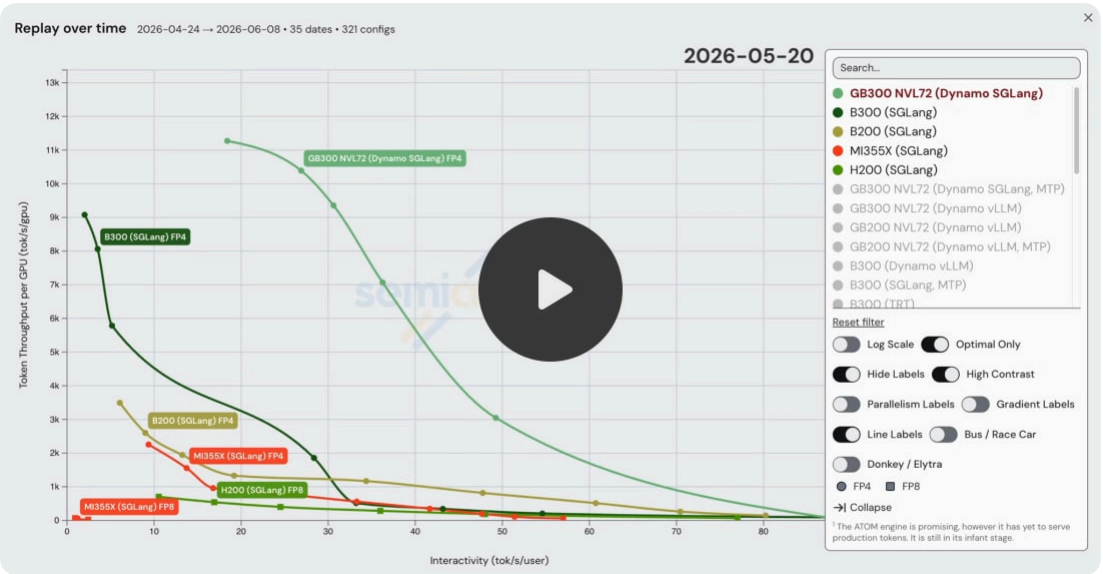
In section 2 of our blog post, we do an comprehensive analysis of DeepSeekv4's inference on Day 0 Huawei Ascend 950DT. This article serves is the first analysis of Ascend 950DT DeepSeekv4 inference and we breakdown the compute<>communication lap & the different compute streams that Huawei did to optimize performance.

A key goal of InferenceX, especially during a model's Day 0 release window, is to record each SKU's performance using open-sourced images and recipes across as many frameworks as possible, regardless of how well these images and recipes perform. This enables us to track improvements over time, which we believe best reflects the real, deployable performance of each chip. The video below shows iterative improvements for non-MTP configs from Day 0 onward for vLLM/SGLang, respectively. visit inference.com to see the MTP configs from day 0 onwards too.

QQ:1413221435
QQ:1413221435



The graphics reflect the thousands of engineering hours that went into tuning DeepSeek v4 inference performance and most of the optimizations are merged into the master branch of SGLang/vLLM. One of the north start goals of InferenceX is to highlight the iterative improvements to performance *over time*, instead of just snapshots of performance, after all when it comes to engineering, the things you learn along the way are often just as important as the end result.



In the early days of DeepSeek v4 Pro, CUDA vLLM and CUDA SGLang and CUDA vLLM disaggregated prefill worked great out of the box, proving the strength of the vLLM and SGLang open ecosystems. These inference engines are so fundamental to the global ML ecosystem that both teams have started their own company, Inferact and RadixArk, with each raising hundreds of millions of dollars to continue to fuel the growth of their open-source inference engines.

Huawei Ascend has also described and demonstrated Day 0 inference performance support for DeepSeekV4 in their documentation. China currently dominates the open model landscape, with Kimi K2.6 still beating Jensen's Nemotron Committee Coalition's Nemotron 3 Ultra on coding. Furthermore, Nvidia's in house TensorRT-

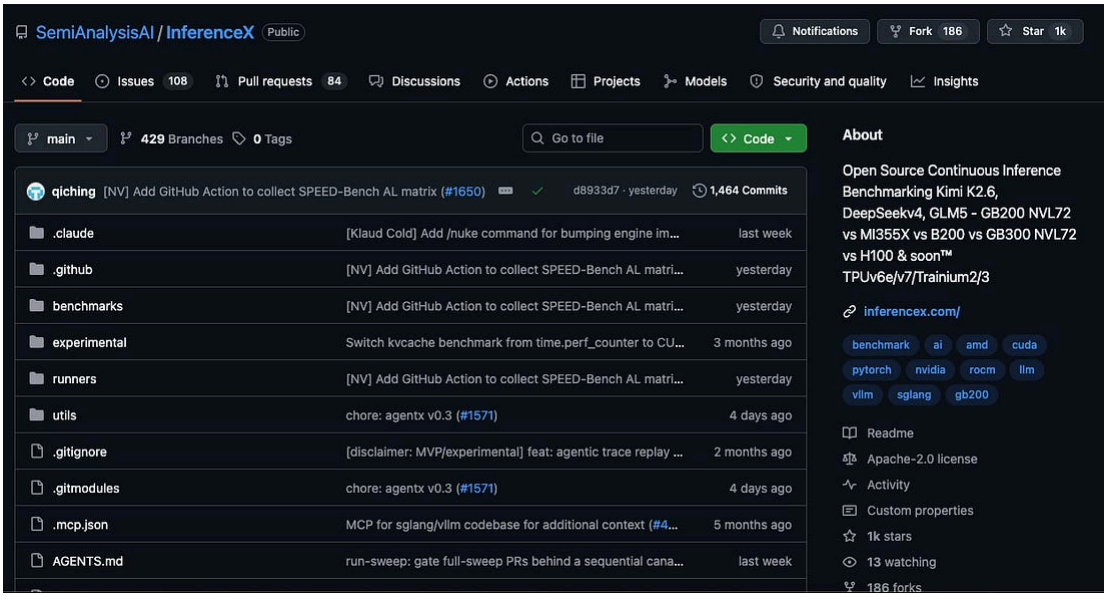
QQ:1413221435

LLM did not work well for DeepSeek v4, and we at SemiAnalysis had to fix their open source mHC kernel launch code. Thank you for NVIDIA engineers for rebasing and merging our patch!

ROCm did not work well either in the first couple of days of DeepSeek v4's launch. That said, the AMD SGLang engineering team, under the technical leadership of HaiShaw, massively improved performance in the first month - achieving a more than 100x performance by Day 26. We will talk more about the good, & the bad, of AMD software progress in our upcoming, comprehensive State of AMD 2026 article.

All performance tracking is documented in our open-source GitHub repo. Feel free to give us a star if you find the repo useful: https://github.com/SemiAnalysisAI/InferenceX.

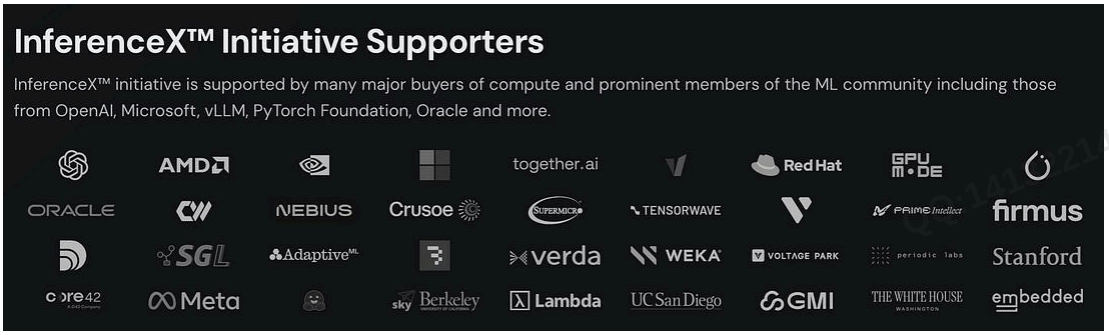
QQ:1413221435



Source: SemiAnalysis - InferenceX GitHub

Our full DeepSeekV4 performance dashboard can also be found here.

The SemiAnalysis InferenceX inference initiative is supported by many in the ML community, including OpenAI, Oracle, Microsoft, Weka, PyTorch Foundation, vLLM, SGLang, and CoreWeave, among others.



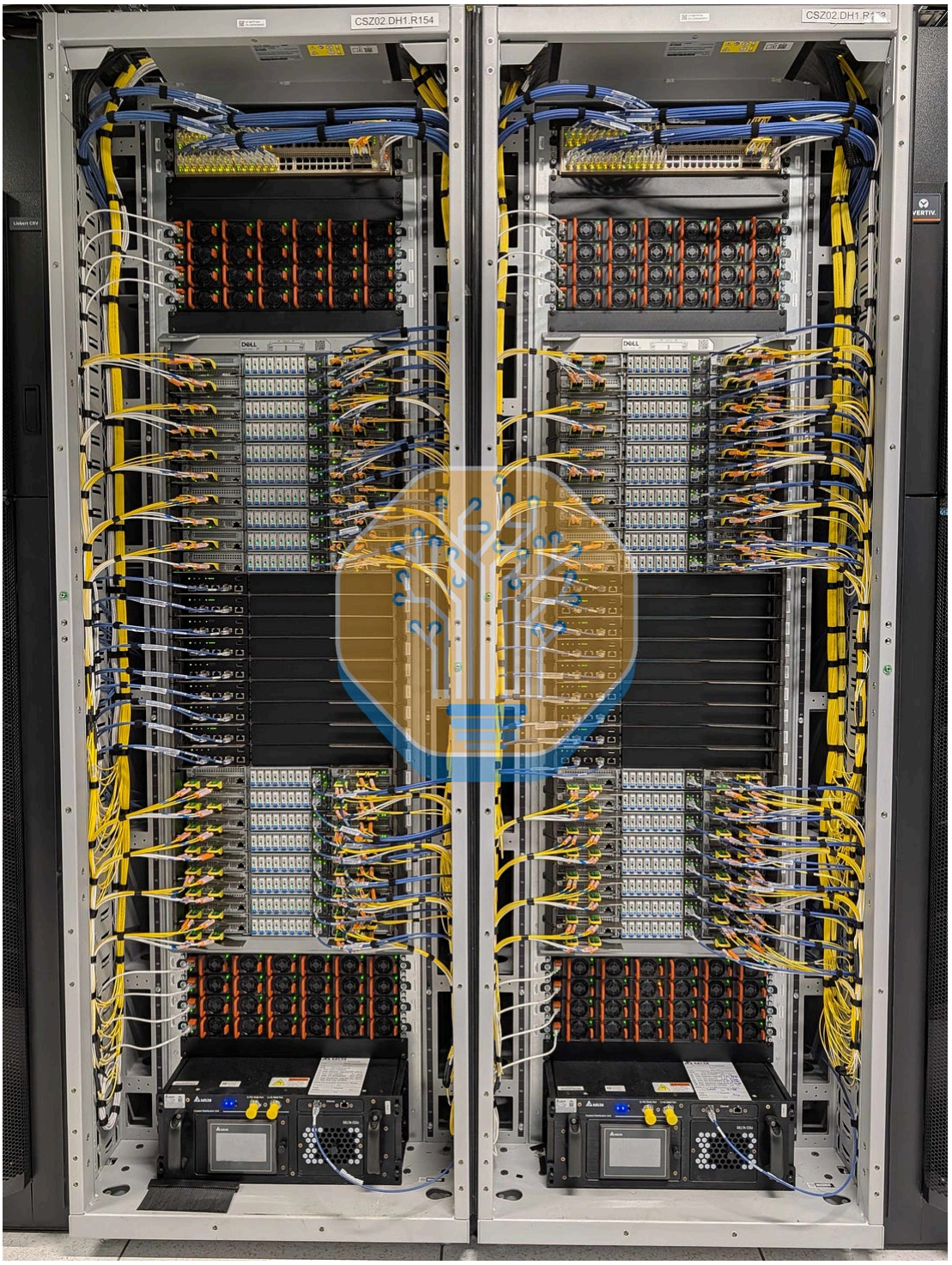
Source: SemiAnalysis InferenceX Supporters

The InferenceX team is hugely thankful for the ongoing engineering efforts carried out by the vLLM community maintainers, Inferact, as well as all of the SGLang maintainers around the world at RadixArk, Meta, and elsewhere. We would also like to shout out to and thank Nvidia engineers Kedar Potdar, Ankur Singh, Xin Li, Alec Flowers and many other Nvidia engineers for their Day 0 support for this project. We would also like to extend our appreciation to the AMD engineering team for their Day X support of DeepSeek v4 Pro on the ROCm stack.

Unfortunately, our GB300 cluster happened to be down when DeepSeek v4 was released. Luckily, CoreWeave came through and contributed compute to the open source community and maintainers, scrambling to find two spare dev GB300 NVL72 racks. Our GB300 results were only achievable because of their support, and we are using it around the clock to drive further improvements to results.

QQ:1413221435

QQ:1413221435



Source: SemiAnlaysis

If you want to work on low-level benchmarking, InferenceX, or other interesting technical work, then send us your resume to letsgo@semianalysis.com with three bullet points demonstrating your engineering abilities. If available, please attach GitHub repo links, websites, or blogs to show off your projects, work, and knowledge.

SemiAnalysis is a reader-supported publication. To receive new posts and support our work, consider becoming a free or paid subscriber.

QQ:1413221435

Subscribed

Section 1: DeepSeekV4 Pro Day 0 Performance

In this section, we will start by discussing DeepSeek v4 Pro's out of the box performance on Day 0. We will reference throughput-interactivity curves, how different parallelisms favor throughput versus interactivity, and other inference optimizations such as MTP and disaggregated inference, which were explained in the InferenceX V2 article.



InferenceX v2: NVIDIA Blackwell Vs AMD vs Hopper - Formerly InferenceMAX

DYLAN PATEL, CAM QUILICI, AND 5 OTHERS·FEB 16

Read full story

> Note that to avert a potential Inference World War 3 and prevent another round of vLLM vs SGLang twitter drama/rap battle, for this article, we will not be showing results for both vLLM and SGLang for same hardware SKU on the same graph.

QQ:1413221435

QQ:1413221435



Source: SemiAnalysis

The following two graphs shows all the Day 0 recipes we managed to record, with most recipes using the native model checkpoint utilizing mixed FP4 MoE-FP8 Attention quantized weights (except for the H200 and MI355X SKUs). Because the native FP4+FP8 checkpoint for DeepSeekV4 Pro was not usable on Day 0 for the MI355X, we were only left with the option of using a full FP8 non-native checkpoint.

Unfortunately, AMD SGLang and AMD vLLM distributed inference still does not work on DeepSeekV4 Pro.

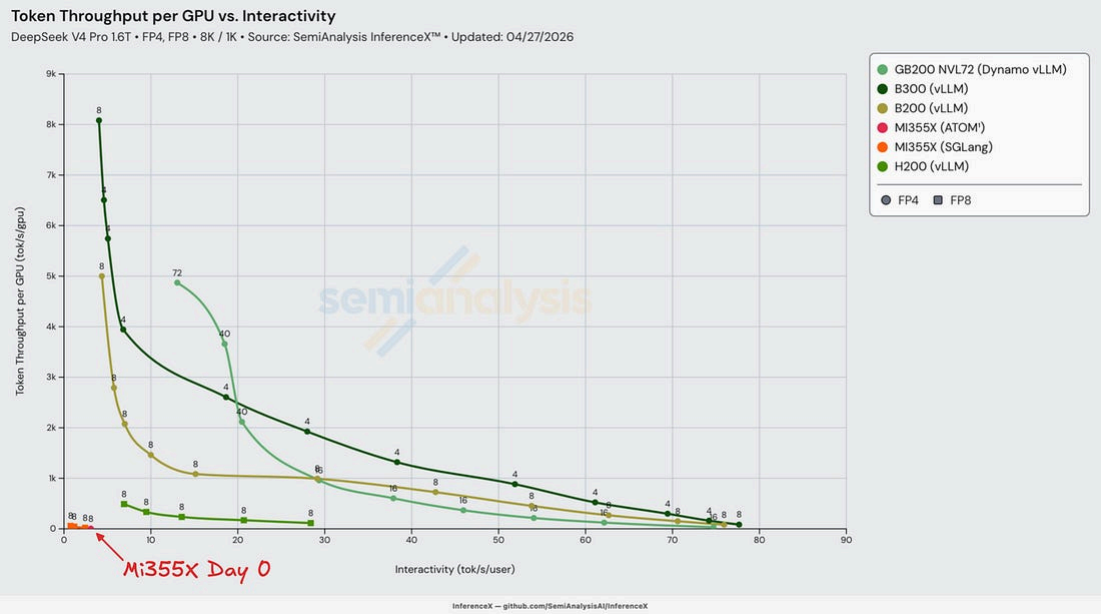
SemiAnalysis is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

Subscribed

QQ:1413221435

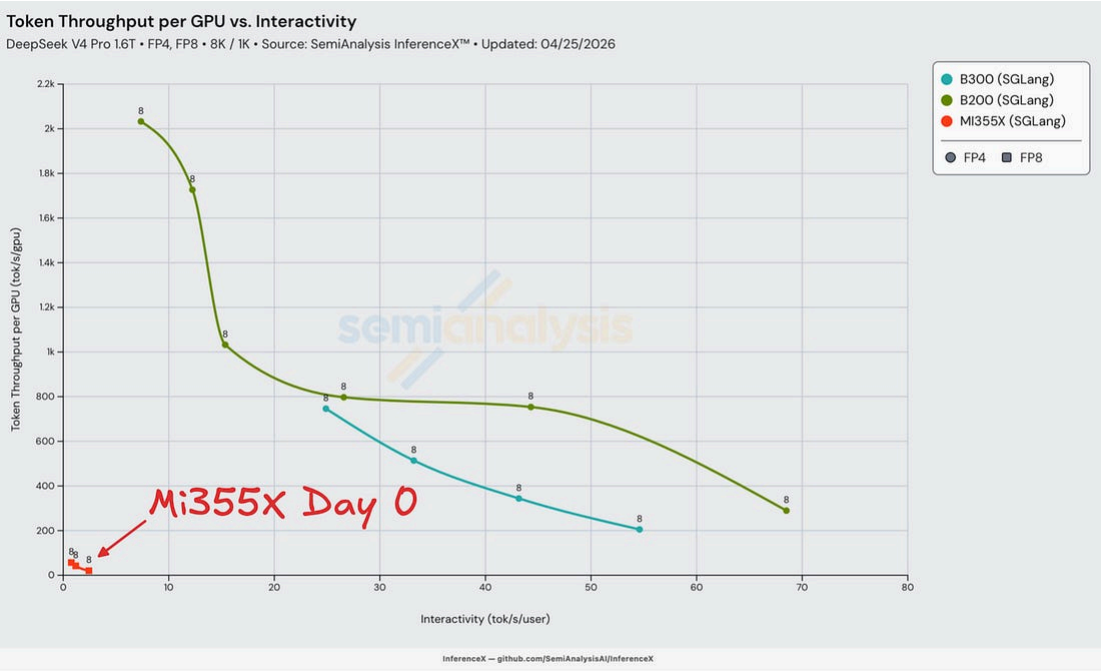
QQ:1413221435

Turing to SGLang and vLLM, both supported native DeepSeek v4 Pro on the CUDA platform the moment the model was released publicly. Most advertised recipes, especially for newer SKUs such as B200/B300, worked out of the box without any major issues.



Source: InferenceX

The below graph shows SGLang Day 0 performance:

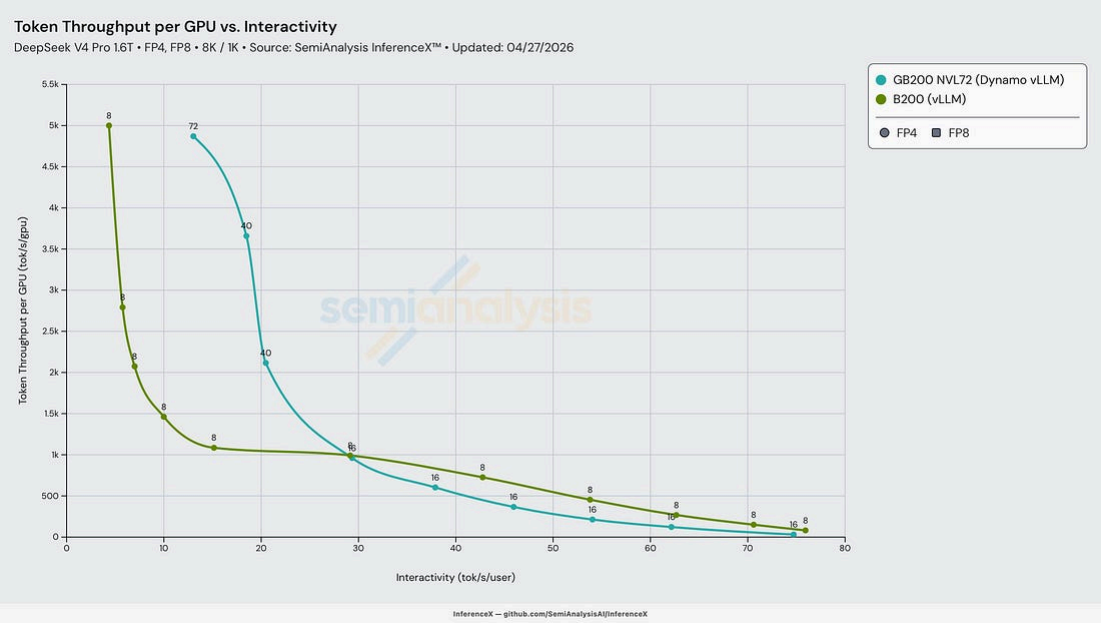


Source: InferenceX

Let’s now dive deeper into each group of Day 0 results.

QQ:1413221435
QQ:1413221435

Day 0 Multi-Node Disaggregated Prefill on GB200 NVL72



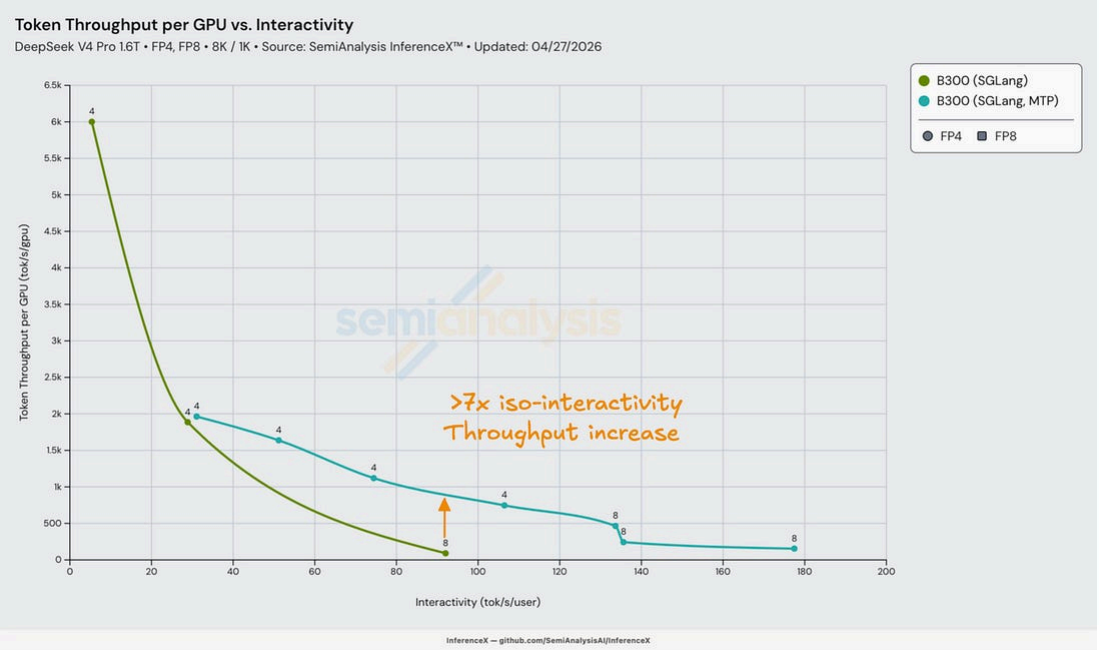
Source: InferenceX

vLLM and Nvidia were very fast in delivering their GB200 distributed inference Dynamo vLLM recipe in srt-slurm. Disaggregated inferencing and wide expert parallelism (WideEP) are inference optimization techniques that can considerably improve performance per dollar - readers can learn more about these techniques in our InferenceX V2 article. The recipe itself was rudimentary: eager on prefill, using NIXL for KV cache transfer. We independently replicated the recipe achieving up to 5x better results than for a B200 run using lower interactivity configs.

This is a great example of the CUDA moat at work: With CUDA, distributed inferencing tends to be supported near Day 0 for the latest open models.

Day 3 Multi-Token Prediction (MTP) Speculative Decoding

QQ:1413221435
QQ:1413221435



Source: InferenceX

QQ:1413221435

The first MTP support delivered for DeepSeek v4 came on Day 3 from SGLang. Using MTP led to a substantial improvement in throughput at higher interactivities. An explanation of MTP and how it benefits memory-bound small batch size decode can be found in our InferenceX V2 article.

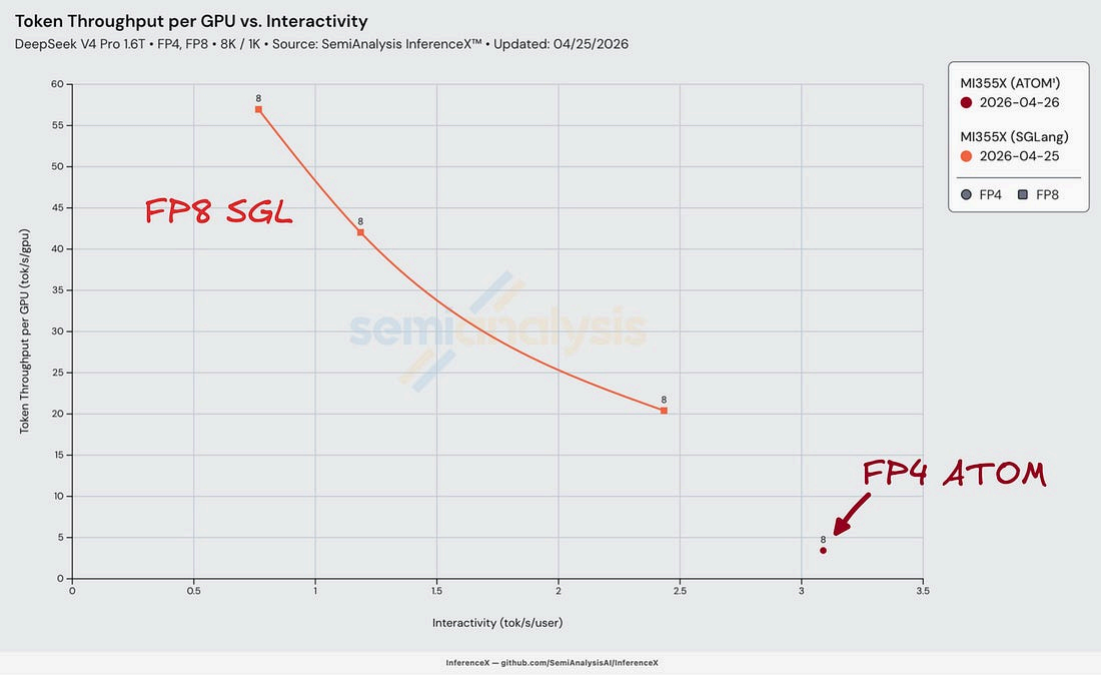
Subscribed

Day 0 ROCm AMD MI355X Disappointment

Turning to ROCm on AMD MI355X, our Day 0 results for DeepSeek v4 were confusing. Most AMD users in the open-source ecosystem were also mired in confusion. The MI355x could only run FP8 on Day 0, and delivered results on the bottom left of the chart below in the overall Day 0 plot. Inference was technically working, but it was an unusable experience given extremely low interactivity levels of only 1-2 tokens per user per second, far below average user reading speeds.

We used the Day 0 WIP recipe provided by an SGLang PR, courtesy of HaiShaw et al at AMD. This was the only working recipe we could find on Day 0. Unfortunately, its performance was disappointing and native FP4+FP8 checkpointing didn't work - this was likely due to the less mature ROCm ecosystem. However, as we will talk about later in the article, HaiShaw's team eventually came through, doing an amazing job by improving performance by over 100x from Day 0 through Day 26 through some classic first principles driven engineering work.

QQ:1413221435



Source: InferenceX

Subscribed

AMD ATOM Inference Engine Disappointment

ATOM did somewhat better on interactivity, but it still falls short for a concurrency of greater than 1. We used ATOM #650 during the early days of DeepSeek v4 hardcoded `kv_cache[:1,...]`, and this meant that the KV cache was pinned to a single sequence slot. With only one slot available, a second concurrent request has nowhere to store its KV state. This was the case because the infrastructure that enables batching possible was not yet in place, so we could only run batch size of one user.

ATOM also ran with almost every hot path on a fallback: the FP4 MoE was forced onto Triton because AITER's `fused_moe` was broken on GFX950, and the mHC pre-projection patched to Torch because AITER's kernel crashed, forcing eager execution.

NVIDIA TensorRT-LLM Bugs and Lack of Day 0 DeepSeekV4 Pro Support

TensorRT did not support DeepSeek v4 out of the box because `mhcFusedHcKernel.cu` had a single hardcoded `FHC_HIDDEN = 4096` constant. This is a problem because SHAPE_K, residual/x TMA descriptors, and the

QQ:1413221435

MMA kernel template instantiations were all tied to that hidden size. All previous DeepSeek models and DeepSeek v4 flash have a hidden size of 4096, so this worked for the meantime. But attempting to run inference for DeepSeek v4 Pro resulted in a `“mhcFusedHcLaunch: hidden_size=7168 not supported (only 4096)”` guard error.

Nvidia engineers also encountered this guard error and instead of adding code to support DeepSeek v4 Pro’s 7168 hidden size, they simply removed the guard. To nobody’s surprise, the error then disappeared.

SemiAnalysis is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

Subscribed

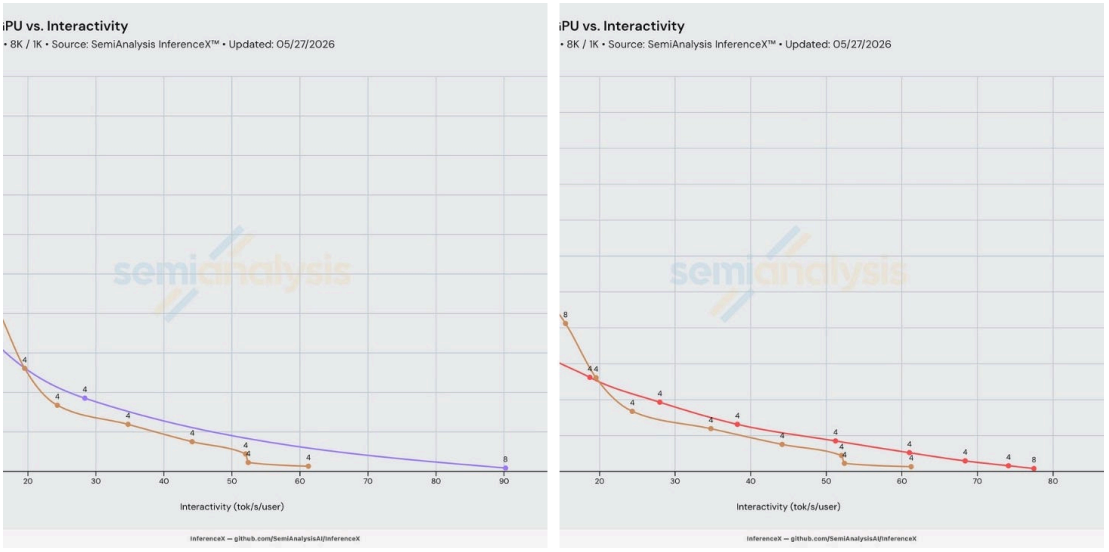
QQ:1413221435

Because of this “fix”, there was a period of over a week where, unless the env var `TRTLLM_MHC_ENABLE_FUSED_HC=0` was used, the kernel would compile exclusively for 4096, with nothing rejecting a 7,168 call. Using default settings (fused HC on by default; B300 = SM10x → MMA path), a stock trtllm-serve of DeepSeek v4 Pro feeds 7,168 tensors into the 4,096-wired kernel. Running inference with these settings results in an inference run without triggering an immediate crash, but there are hidden consequences: the engine ends up corrupting hidden states and producing invalid generations. This was fixed in a PR authored by us, and it was surprising that such a simple problem took a week to be noticed and that it took another few days for the PR to be approved.

In the time it took to diagnose the issue and narrow the problem down to the fused HC hidden size mismatch, we had already reached Day 9 post DeekSeep v4 Pro launch. This episode is a good case study that proves the strength of the open native SGLang and native vLLM engine ecosystems. Thanks to these robust ecosystems, Day 0 support will always come to native SGLang and native vLLM first before it comes to TensorRT-LLM or AMD’s ATOM engine (ATOM, by the way, currently has zero production customers).

In the graphs below, we can see how as of today, TRT-LLM’s performance is superior at higher batch sizes, but it tends to fall behind at higher interactivity levels.

QQ:1413221435



Source: InferenceX

Section 1.5: Performance Over Time

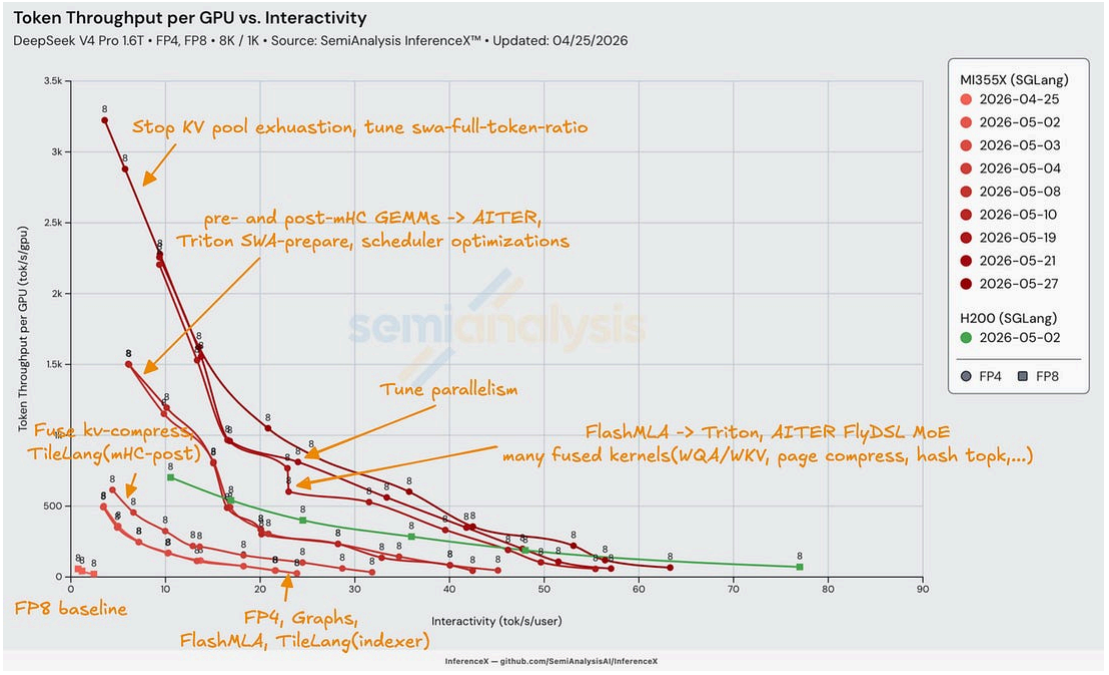
As mentioned earlier in the article, we capture a snapshot of Day 0 performance across the inference engines and recipes as it acts as the baseline against which improvements in performance can be measured over time. With this baseline performance - we are able to measure and present the following data analyzing performance improvements over time.

DeepSeek v4 Pro on MI355X - 100x Improvement in less than 1 Month

On Day 0, DeepSeek v4 Pro was technically working on the MI355X, but it was clearly not deployable into any production workflow. However, the improvement since then have been phenomenal - with the AMD team led by HaiShaw delivering an over 100x improvement in throughput in less than a month post DeepSeek v4 release.

QQ:1413221435
QQ:1413221435





Source: [InferenceX]

The chart above shows how the throughput pareto-optimal frontier climbed when comparing results from the FP8 build on Day 0 released on 25th April to results from the FP4 build released on the 27th of May. The gain came almost entirely from AMD replacing PyTorch-native fallback paths with real AITER, Triton, TileLang, and FlyDSL kernels.

There are two steps that drive the lion’s share of the gains. The largest percentage improvement actually came from the first commit after the baseline Day 0 submission - the team managed to mop up a ton of low hanging fruit and significantly the first iteration from the FP8 baseline. The next largest improvement came a few days later as the AMD team finally got FP4 weight MoE working, allowing us to switch MoE experts from FP8 to native FP4 (MXFP4), improving expert-weight

QQ:1413221435

bandwidth. This also moved FlashMLA and the sparse-attention indexer off the torch fallback onto TileLang kernels and enabled HIP graphs.

The next big improvement we saw came from the introduction of AITER mHC kernels, which are used at every layer. This improvement boosted performance such that we were able to see MI355X exceed H200 performance for DeepSeek v4 Pro at lower interactivity levels for the first time.

Before the windowed-attention kernel can run, it needs to know which KV-cache slots each query's window covers. This is done by SWA-prepare, and its implementation in Triton also helped with the improvement.

The next big jump came on May 19th as the team retired the remaining fallbacks: FlashMLA moved from TileLang to Triton, and as the AITER FlyDSL FP4 MoE kernel landed. The team also enabled fused hash-topk, DSv4 radix attention, fused store-cache, fused WQA/WKV projection, and fused paged-compress, further boosting performance. The concurrency sweep was also increased to 1024, drawing the high-throughput, low-interactivity end of the frontier that did not exist before.

QQ:1413221435



ATOM grows from one point in bottom left to full frontier. H200 for reference. Source: InferenceX

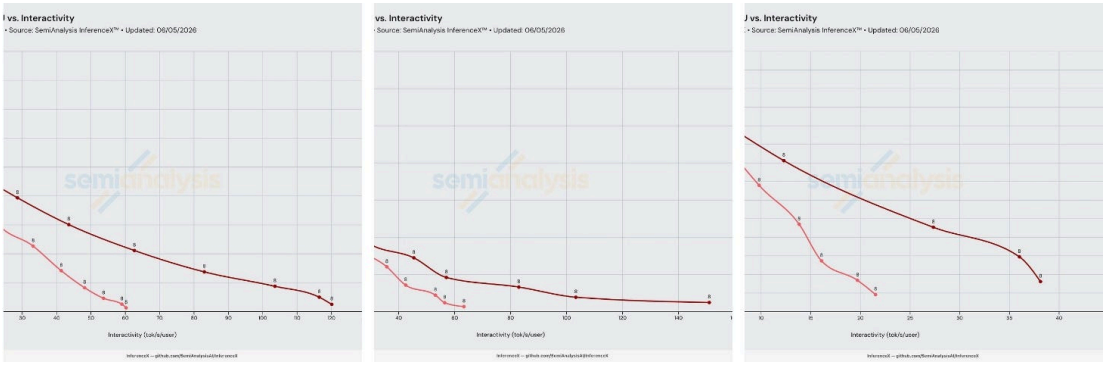
ATOM also improved drastically, expanding from a single conc=1 point to delivering decent throughput along the whole pareto frontier, with some points beating H200. The first gain came from AITER fix #2916, which corrected the device-allocation bug behind the mHC crash and let ATOM restore that AITER kernel. Next, FP4 experts moved onto AITER's fused MoE kernel (Triton override removed), the

QQ:1413221435

sparse-attention OOM was cleared so eager mode and the single-sequence caps could be dropped. Batching support was also implemented, expanding the sweep from conc=1 to conc 1–512 with much better performance.

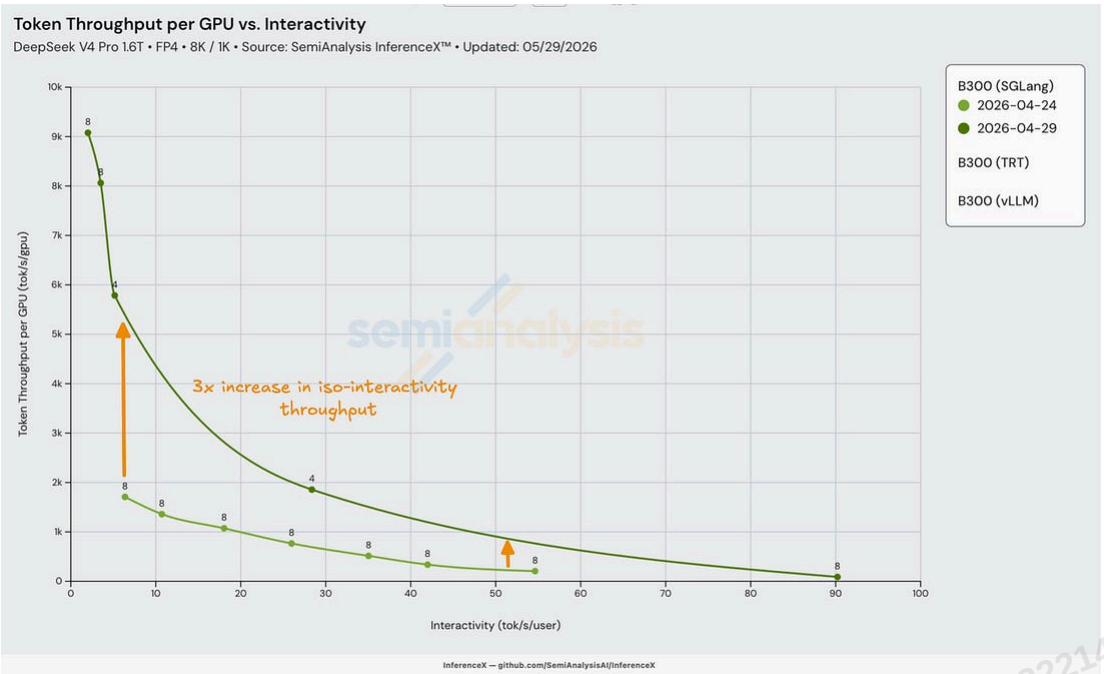
MI355X MTP

By Week 4, MTP was working for AMD on all frameworks, delivering multiple-fold is-interactivity throughput improvement. One consistent characteristic we noticed however, is that MTP tends to deliver worse results at higher throughput. This is because MTP exploits the compute slack in memory-bound decode, so compute-bound large batch size decode jobs have an MTP cost that outweighs the benefit that draft tokens would provide.



Sources: InferenceX

B300

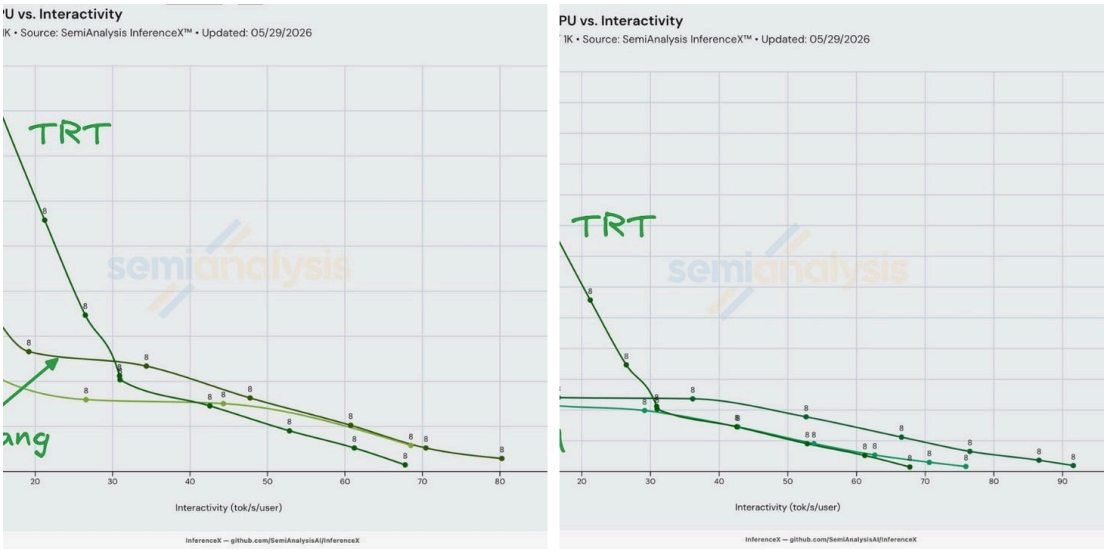


Source: [InferenceX]

QQ:1413221435

For B300 on SGLang, DeepGEMM MegaMoE, our results showed a 3x improvement over less than a week thanks to the use of a grouped FP4 MoE GEMM that keeps experts resident and does one mega-dispatch instead of per-expert kernels, as well as from tuning to use EP4 instead of EP8.

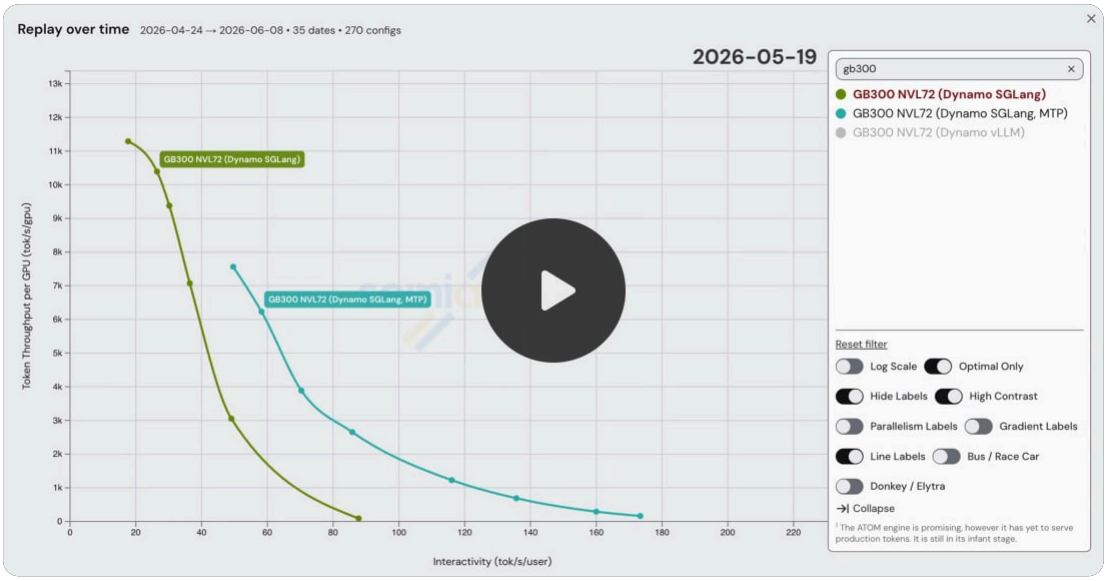
B200



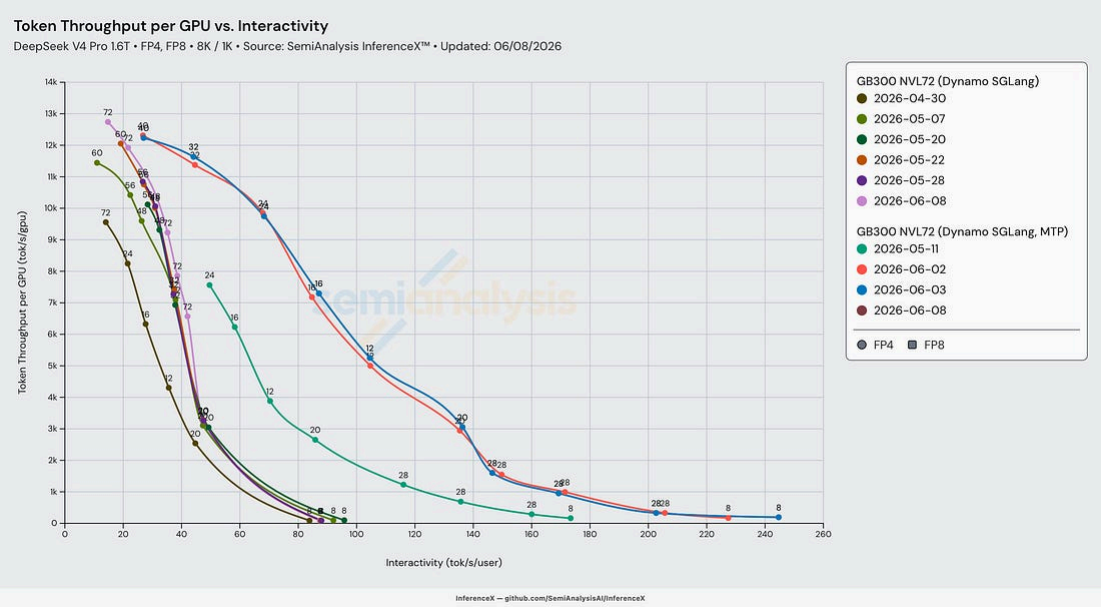
Source: InferenceX

Performance for the B200 was relatively similar to that of the B300, with TRT being superior at lower interactivity for the B200. But TRTLLM does not work out of the box, compared to CUDA vLLM & SGLang vLLM which works out of the box.

GB300 NVL72



QQ:1413221435
QQ:1413221435



Source: InferenceX

The most dramatic improvements for GB300 SGLang MTP came on June 2nd from the implementation of W4A4 (MXFP4) MegaMoE. Compared to the non-MTP implementation that was in use on May 7th, the main improvements in the June 2nd version came entirely from a rewrite of the GB300 decode topology rather than touching kernels or precision. While the Day 0 recipe ran at most points with narrow EP=8 fed by one or two prefill workers and capped concurrency at 16,384; the May 20th run widened decode to EP=16, scaled prefill to four–twelve workers per decode worker, and pushed concurrency to 21,504.

Based on the graph and analysis above, we can see that, as expected for a larger world size inference system, Wide EP is the main lever for GB300's excellent performance which is delivered by amortized weight loading across more GPUs. Read the InferenceX V2 article to find out more about Wide EP.

These results for GB300 were only possible due to CoreWeave's support.

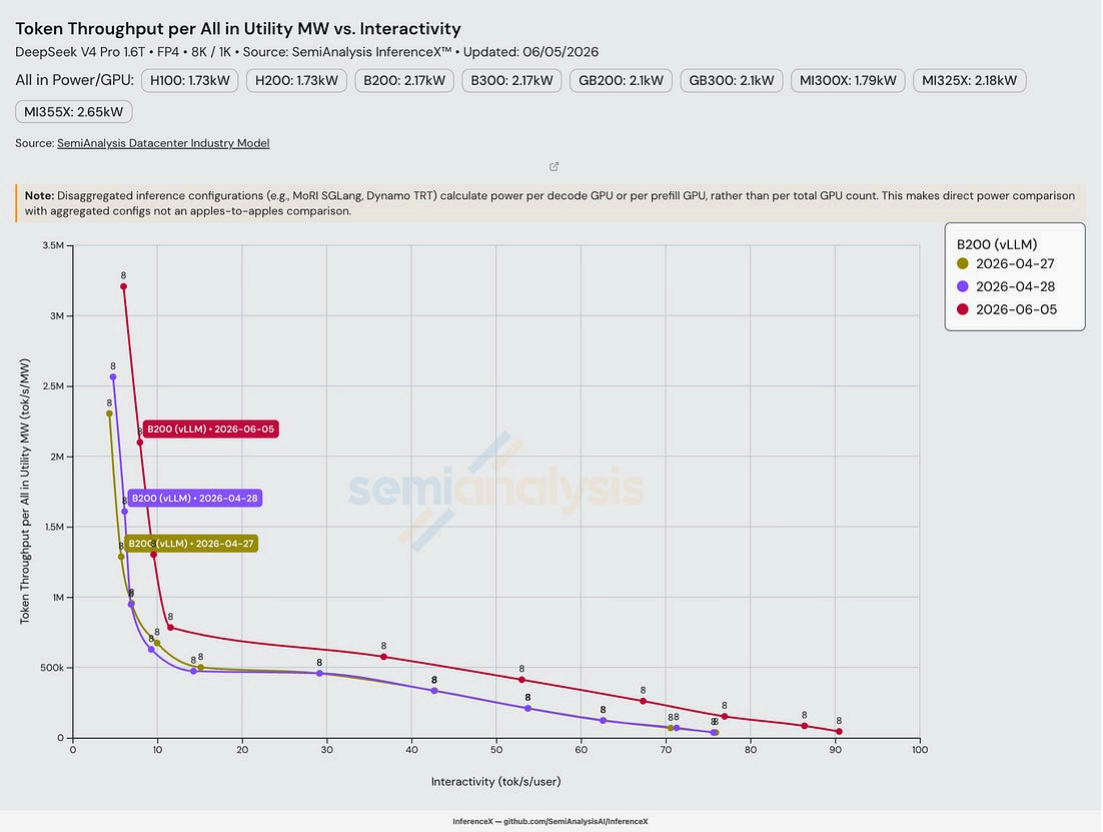
Thanks for reading SemiAnalysis! This post is public so feel free to share it.

Share

B200 Tokens Per Megawatt (MW) Improvements

QQ:1413221435

For the B200 with the vLLM engine, token throughput per all-in provisioned utility megawatt reached 300,000 tokens per second per MW for 50 tok/s/user interactivity on Day 0, improving to nearly 500,000 tokens per second per MW by June 5th.



Source: InferenceX

Tokens per all-in provisioned-utility MW is the best figure of merit for considering return on investment at the fleet scale: It adds more information to raw per-GPU token throughput because it reflects PUE and datacenter overhead. Because a B200's all-in utility power envelope is fixed near 2.17 kW/GPU, that ~1.7x jump from ~300k to ~500k tok/s/MW reflects a pure software gain.

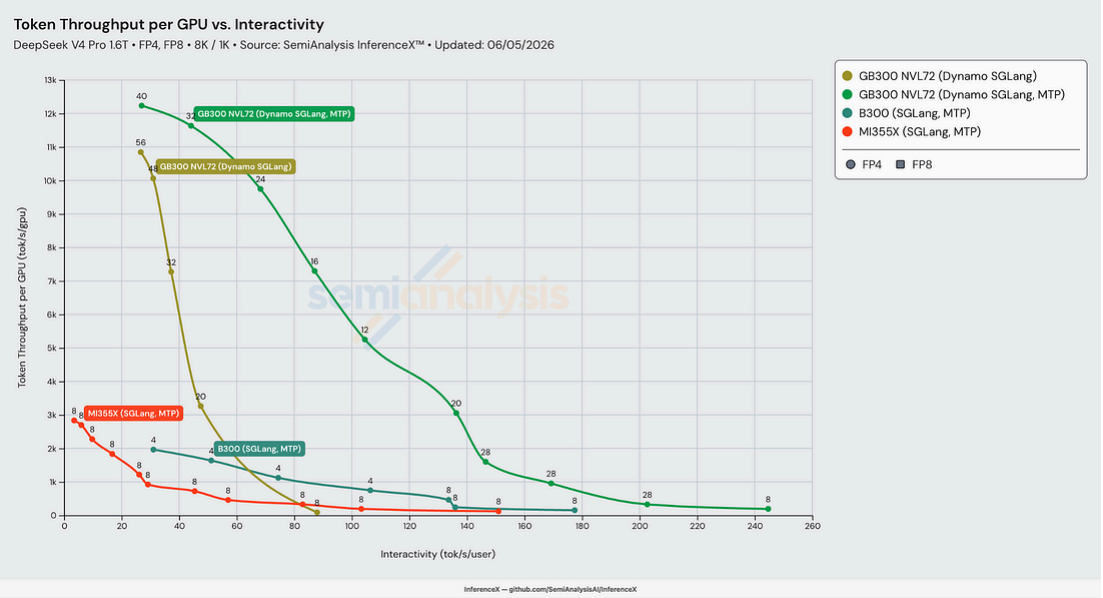
The same class of optimizations that pushed the throughput frontier (MegaMoE grouped-FP4 GEMMs, wider EP, the FP4 weight path, scheduler tuning) drop straight through to power efficiency because the all-in utility power in MW is unchanged.

Many organizations approach inference fleets from a perspective of maximizing a scare quantity of utility power. The question is how one can convert provisioned MW into as many billed tokens as possible at a given utilization and price. This analysis is best informed by metrics like revenue per MW, tokens per all-in utility power, capex per MW among others. This is a business case that our Tokenomics model is built to address.

QQ:1413221435

Current Performance As of June 6, 2026

Let’s round out this section on performance improvements by quickly reviewing the best performance across systems and inference engines. When using SGLang, the GB300 continues to dominates and mogs all other inference systems, demonstrating the advantage of the GB300 NVL72’s rack-scale world size.



Source: InferenceX

When we switch on MTP, serving with the GB300 is unbeatable across all interactivity levels we analyzed. The cost per million output tokens for the GB300 reaches $0.156 at 50 tok/s/user assuming 8k tokens input, 1k tokens output. Find out more about how we calculate Total Cost of Ownership (TCO) in our TCO model.

QQ:1413221435
QQ:1413221435



Source: [InferenceX TCO calculator]

The rack-scale advantage is fundamentally a scale-up domain story. NVL72 puts 72 GPUs in a single NVLink domain, which lets the serving stack run expert parallelism wide enough to keep DeepSeek V4's MoE dispatch/combine all-to-all entirely on NVLink instead of spilling onto the slower scale-out fabric, all while amortizing expert-weight loads across far more ranks.

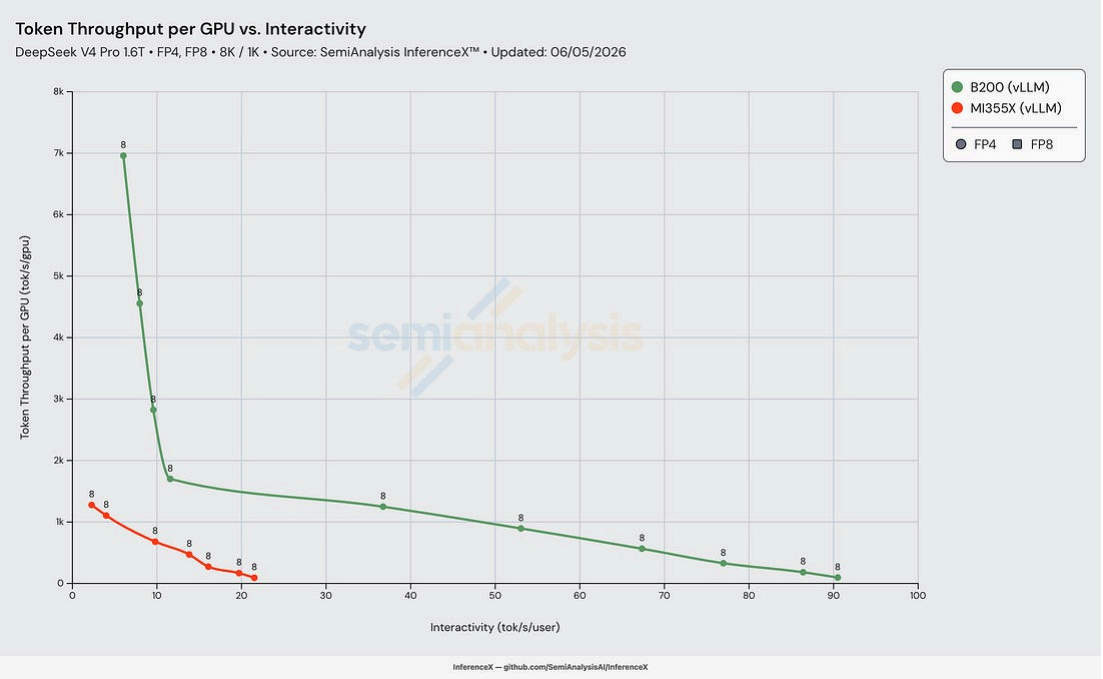
B200 and B300 in 8-GPU NVLink islands scaled out over InfiniBand hits that wall much earlier, and the MI355X sits further back on both scale-up domain size and collective-stack maturity. Turning these per-rack throughput wins into deployed serving capacity is a separate question that comes down to how many of each SKU are actually online: shipments and ASPs by SKU, plus installed base and effective FLOPS by customer, quarter by quarter, which we track in our Accelerator & HBM model.

ROCm vLLM DeepSeek v4 Pro Disappointment

When it comes to ROCm, progress has been much slower for native vLLM vs native SGLang. Performance of ROCm vLLM is lagging far behind its CUDA vLLM counterpart. Part of the issue is that AMD is re-focusing on ATOM (an inference engine that serves 0 production tokens) instead of focusing on native vLLM (an inference engine that lots of their major customers uses). We will talk more about in our upcoming State of AMD 2026 article (covering the good, the bad, and the ugly when it comes to AMD inference). One of the positive developments we will be

QQ:1413221435

covering is that the functional enablement of distributed inferencing on open source out of the box upstream AMD vLLM has finally happened for non-DeepSeekv4 models. It took many months to get there, and there is still much ground for the AMD vLLM team to cover.



Source: InferenceX

What’s next for DeepSeek v4 vLLM

vLLM’s plan is tracked in the DeepSeek V4 roadmap issue (#40902), with landed code in implementation PR #40860 and a description of benchmarking against SemiAnalysis’s InferenceX dashboard. The FP4 Indexer and initial MegaMoE support has already been implemented, and Hopper is now supported. The remaining work for vLLM for DeepSeek v4 spans five areas:

- **Core model support**: continued MegaMoE work (PR #40833) and NVFP4 support.
- **Runtime and parallelism**: Model Runner V2 integration, MTP optimizations, prefill/decode (PD) optimizations, and pipeline parallelism support.
- **Kernel integration**: a paged prefill kernel, a fast top-k kernel, more horizontal fusion, DeepEP V2, and integration with DeepSeek’s own TileKernels.
- **KV cache**: KV cache offloading, covering PD + CPU offloading (PR #39654) and distributed KV offloading.

QQ:1413221435
QQ:1413221435

- **Hardware support**: beyond the completed Hopper support, SM120 and AMD support remains as key to-do items.

The focal themes here are on the surrounding systems: a new model runner, pipeline parallelism, KV offloading, and wider hardware coverage.

Upcoming updates to InferenceX, SemiAnalysis's open source & public's EcosystemX dashboard will visualize the software evolution & CI coverage & queue times across all major ML open libraries across all major AI chips: NVIDIA, AMD, TPU, Trainium, Huawei, and more.

SGLang

SGLang's plan lives in performance optimization tracker (#23666) that Nvidia structured around DeepSeek v4's network diagram, walking it block by block; some items may already be partially covered by the initial support PR (#23600), and community contributions are welcomed.

There are three high-level goals reflected here: CUDA graph support for decode, piecewise CUDA graph support for prefill, and no at-runtime weight processing. In addition, weight prep should happen once rather than per step. Within these three high level goals, the checklist is grouped by V4's components:

- **mHC**:trying TF32/BF16 for the `fc_hc_fn` GEMM (which has a small N dimension and may need a special kernel), a 1/RMS + multiply fusion, single-kernel `hc_split_sinkhorn` and `hc_post`, and fusing MulSum + RMSNorm (+ FP8/MXFP8 quant) in the attention and MoE blocks.
- **HCA (with Compressor)**: horizontal fusion of `fc_qa` + `fc_kv` into one FP8 GEMM, q-norm/k-norm and RMSNorm+RoPE fusions, a non-sparse MQA path that drops `topk_idx`, MQA reading directly from the compressed and SWA KV-caches with no copy/concat, single-kernel InvRoPE, a fused Compressor state update (kv-update + ape-Add + score-update), and making HCA, especially the Compressor, CUDA-graph-compatible for decode.
- **CSA (Indexer + Compressor)**: similar direct cache reads for the sparse path, optional (P1) fusions of fc_compressor + fc_idx_compressor and fc_qb + fc_idx_qb, a (RoPE +) Hadamard + MXFP4-quant fusion, an efficient MXFP4 BMM+ReLU kernel (potentially fused with MulSum and even Top-1024), and CUDA-graph compatibility for the Indexer and Compressor.
- **MoE**: checking TF32/BF16 for the router GEMM, collapsing the routing path (softplus + sqrt + bias-add + Top-6 + gather + norm + multiply) into as few

QQ:1413221435
QQ:1413221435

kernels as possible, fusing block-wise FP8 and MXFP8 activation quant, ensuring both shared-expert and routed-expert FC13 are single kernels, and auditing the tiny sorting kernels before the routed experts.

SGLang’s focus is replacing chains of small ops with single fused kernels, making the new attention variants read caches in place, and pulling the decode path fully into CUDA graphs.

Section 2: Huawei 950DT Day 0 DeepSeek v4 Analysis

DeepSeek v4 was the first major open model with first class Day 0 support on the Huawei Ascend, and indeed, part of the DeepSeek official API has been served on Huawei since Day 0. We have Huawei performance numbers on DeepSeek v4 and plan to release a follow up article deep diving into an apples to apples comparison of inference on Huawei vs on the H200 and B200, measuring comparative performance using the same benchmark harness.

Our upcoming public open source SemiAnalysis EcosystemX dashboard will visualize the software evolution and CI coverage across all major ML open libraries, for all major AI chips, including the Ascend stack.

CANN

CANN (Compute Architecture for Neural Networks) is Huawei’s software toolkit for running AI workloads on its own Ascend chips. Starting August 2025, they’ve open-sourced CANN to attract more developers and to “chip” away at Nvidia’s dominance, especially within China given that the US government heavily restricts CUDA chip shipments into China.

QQ:1413221435
QQ:1413221435



Source: CANN slides

QQ:1413221435

On Day 0, CANN released an optimization guide and benchmark figures for Ascend chips. Through it, we can see Huawei's CANN strategy: To make Ascend competitive through full-stack inference optimization that is aimed at Chinese domestic model releases. Huawei is trying to show the Chinese ecosystem that if DeepSeek releases a new architecture, CANN can ship the kernels, graph path, quantization, serving integration, and deployment recipe.

One interesting methodology of the CANN team that we observed while benchmarking MTP and can't help but mention was how they dealt with MTP draft token AR(acceptance rate) or AL(acceptance length). Benchmarking MTP is not trivial as the AR/AL of the benchmark may differ from the user's use case. For example, a benchmark may on average accept two draft token out of three but with deployed use cases being extremely varied, it may end up may only accepting 1.5 tokens out of three on average.

This means that the user may see lower performance compared to the benchmark, and thus wrongly conclude their setup is flawed. We addressed this in our InferenceX v2 article by comparing ARs with MTBench. Future iterations of our benchmark will comprehensively address this gap by using real traces.

To address this quirk, Huawei instead times the full decode step to coincide with the last MTP module, thus recording time per decode step instead of time per token. The final benchmark result published then requires the user to multiply by their usecase's MTP AL to derive comparable performance metrics, which is a very elegant way to compare performance.

QQ:1413221435

Hey NVIDIA Goliath, There’s a New David in Town - The Ascend 950

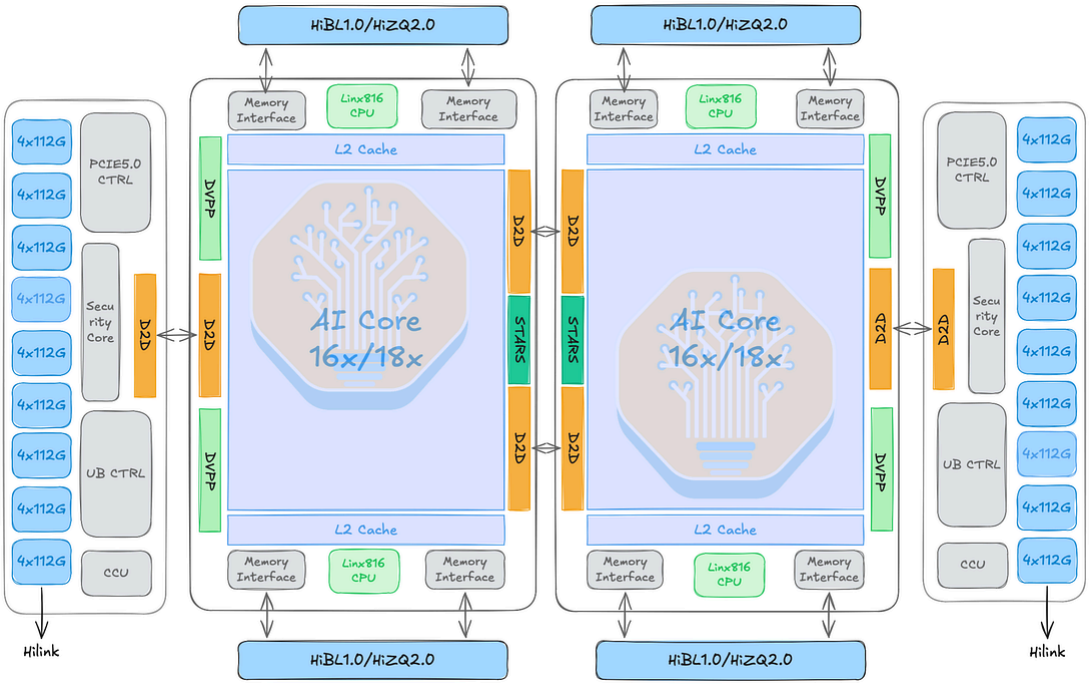
Huawei’s internal codename for Ascend 950 chips is “David”, and this codename is referenced multiple times in the CANN codebase. No doubt it is because they believe they are the David to Nvidia’s Goliath.



QQ:1413221435

Source: SemiAnalysis

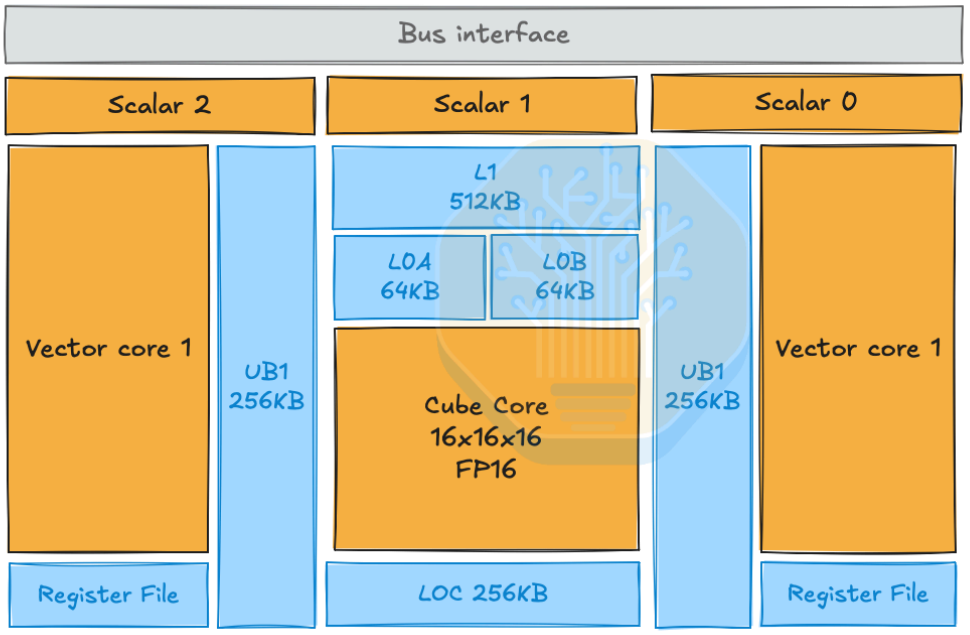
The SIMT/SIMD 950 chips come in two flavors: 950PR, and 950DT. PR stands for Prefill and Recommendation and are lower cost chips with better cost-performance. DT stands for Decode and Training, and this variant features higher memory bandwidth and higher performance. Both are based on the same Ascend 950 Die, which uses a dual-die UMA architecture, but they are each packaged with different memory. The Huawei roadmap estimates and volumes per quarter of each Huawei chip is available in the SemiAnalysis Accelerator model.



Source: CANN

Two major components of the chip architecture that are important to discuss are the AIC(AI Cube) and AIV(AI Vector). AIC is the **matrix/tensor core** side of Ascend's AI Core. It is used for dense matrix math: GEMM, matmul, convolution-like tensor ops, attention projections, FFN linear layers, etc. Huawei documentation describes AIC as the **matrix-compute** core in split AI Core architectures. AIV is the **vector core** side. It handles elementwise/vector work: activation functions, normalization pieces, masking, reductions, type conversion, layout transforms, post-processing around matmuls, etc.

QQ:1413221435
QQ:1413221435



Source: CANN

QQ:1413221435

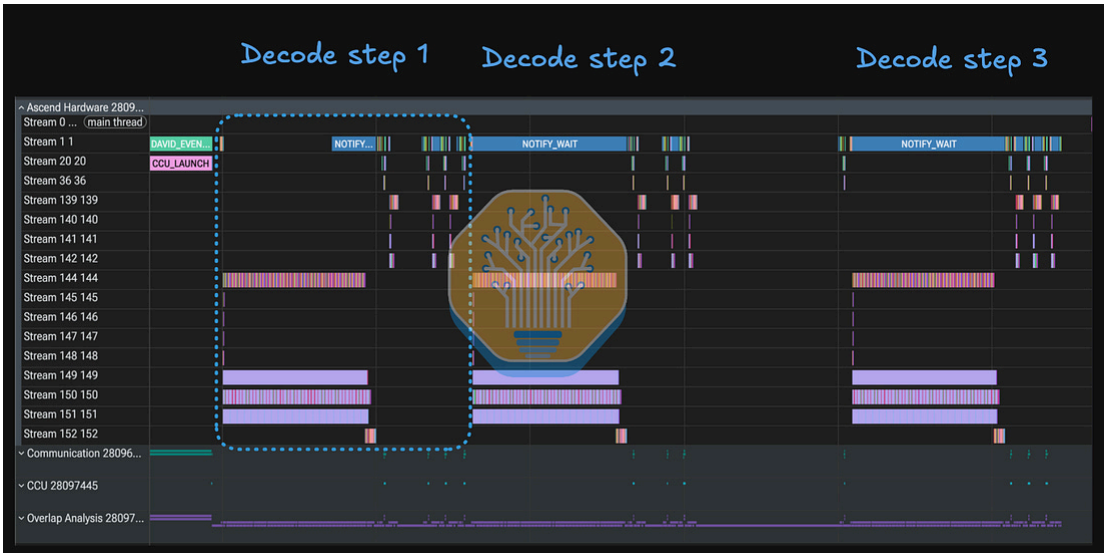
This is similar to TPU's MXU. However, Ascend exposes the split between the two functions more directly as separated independent cores, with each able to load its own code segment, and also feature a "dual-master mode," where AIC and AIV independently run code rather than having the AIV driving the AIC through messages.

The AI CPU is a device-side ARM64 execution unit with direct access to device memory. It is used as a complement to AI Core for the work that maps poorly onto the SIMD/SIMT cores: branch-heavy control flow, scalar logic, dynamic-shape handling, and the value-dependent scheduling/tiling metadata that kernels need before they run. Because the AI CPU lives on the device, Ascend can keep this irregular control-style work local instead of round-tripping work to the host CPU, which is a primary source of latency and pipeline bubbles. The AI CPU is also the unit that historically sat on the older AICore → AICPU → SDMA path for communication orchestration, before the dedicated CCU offloaded that work.

Like the TPU and Trainium, Ascend 950 adds a dedicated CCU communication engine. This engine sits alongside the compute die and handles collective-communication work without consuming AI Core compute capacity by supporting remote-read + reduce + local-write, and local-read + remote-write. The benefit is in lower communication latency, less HBM traffic, fewer user-buffer copies, and freeing compute cores from comm orchestration, avoiding the older AICore -> AICPU -> SDMA path.

QQ:1413221435

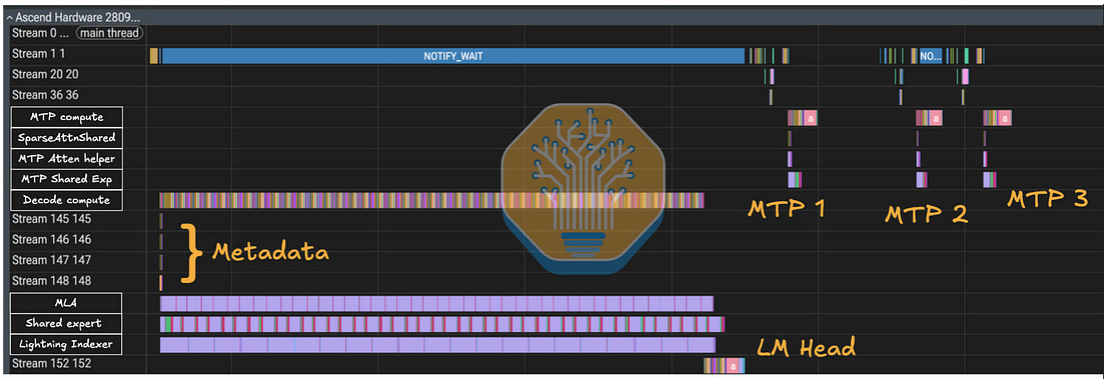
Huawei DeepSeekV4 Pro 950DT Profile



Source: SemiAnalysis, Huawei

The above shows a three step profile of DeepSeek flash v4 on Ascend 950DT, running using a config with a 16-rank DP/EP deployment. It shows 16-rank collective participation plus active MoE dispatch/combine traffic.

As is now standard across most stacks, CANN also uses independent compute and communication operators that can run on multiple streams - performance improves by controlling Cube and Vector core allocation to avoid resource contention. Operations like Prolog, Compressor, and LightningIndexer can be overlapped, C4A Compressor can be completely hidden, and shared expert computation can be hidden under routed expert execution without degrading routed expert performance.



Source: SemiAnalysis, Huawei

Zooming in into a given decode step, we can see how different components are split into streams. Operations on different streams may run concurrently when the device has free, suitable resources. Models use multiple streams because a layer may not be

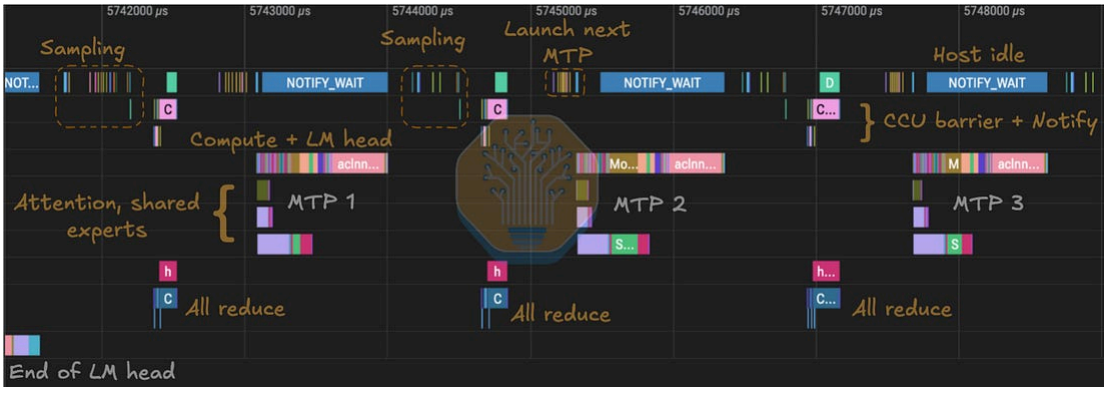
QQ:1413221435

a single serial chain, and instead can contain branches that only need to synchronize when their results are combined, for instance a shared-expert compute overlapping 100% with routed-expert compute.

Streams 145-148 in the above diagram correspond to metadata streams. These operators fire once per decode pass and precompute value-dependent scheduler/tiling metadata that later kernels reuse. They are the only AI CPU ops in the decode step, the make up a tiny fraction of total time, and are fully overlapped with AI Core compute. The impact is likely larger on longer-context benchmarks, where there is more sequence-length and mask-dependent partitioning to resolve up front.

In DeepSeek v4, Huawei moves the value-dependent scheduler stage for sparse attention and the LightningIndexer onto the AI CPU rather than bouncing it back to the host. These metadata ops build reusable per-core partitioning tensors from runtime sequence-length, mask, and paged-KV information; `SparseAttnSharedkv` and `QuantLightningIndexer` then consume them to decide which Batch/Head/Q-block/K-block work each cube core handles, along with the corresponding vector-core reduction tasks. Conceptually, this mirrors FlashInfer's planning phase on host for paged attention: a cheap, dynamic-shape-aware setup step that runs once and is thus amortized over layers, with the only caveat being that Huawei pushes that same planning work onto the on-device AI CPU instead of the host.

Stream 152 in the daigram above contains the LM head, the last layer, and the second last layer's `o_proj` and MoE. This is a decision of the `npugraph_ex` graph compiler, likely to allow the `npugraph_ex` runtime to consider the main graph "complete" on stream 144 while the tail work continues asynchronously.



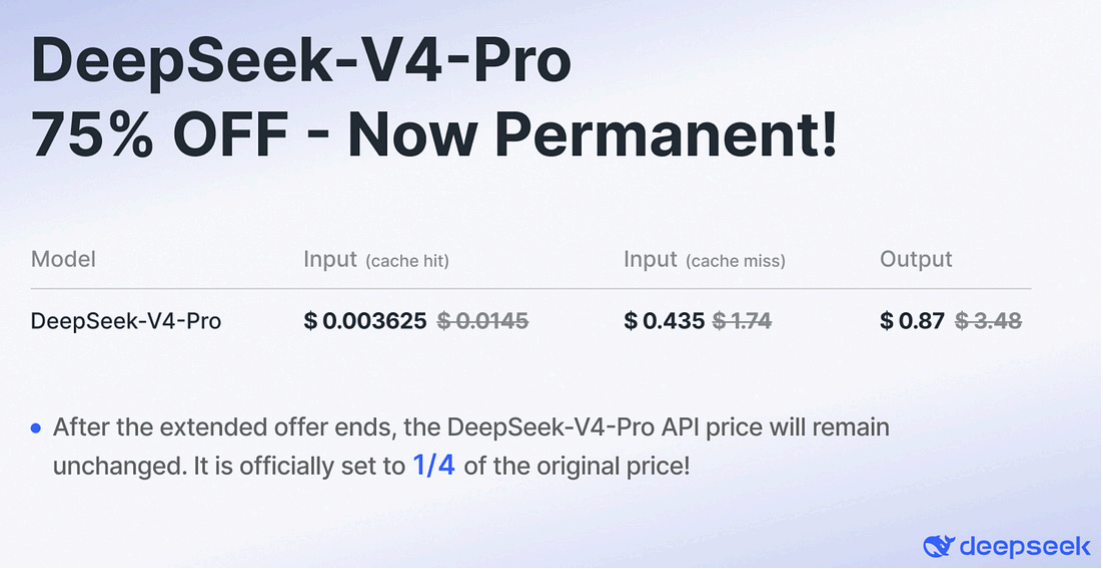
Source: SemiAnalysis, Huawei

CANN also introduced MC² (merged compute-communication) back in 2024. This is a class of fused operators that are not ordinary kernels nor HCCL collectives. They embed communication and compute into one kernel. In DeepSeek v4 decode, we can

QQ:1413221435

see `MoeDistributeDispatchV2` and `MoeDistributeCombineV2` MC² EP operators being used.

The main takeaway here is that Ascend delivers working, optimized inference infrastructure for DeepSeek v4 on Day 0. The Huawei CANN stack is one of only two stacks with Day 0 Support for DeepSeekV4, the other being Nvidia's CUDA. As we explained earlier in the article, AMD's stack unfortunately did not work well on Day 0. This is in stark contrast to last year when DeepSeek v3/R1 released. Back then, only one stack worked on Day 0: the Nvidia CUDA stack.

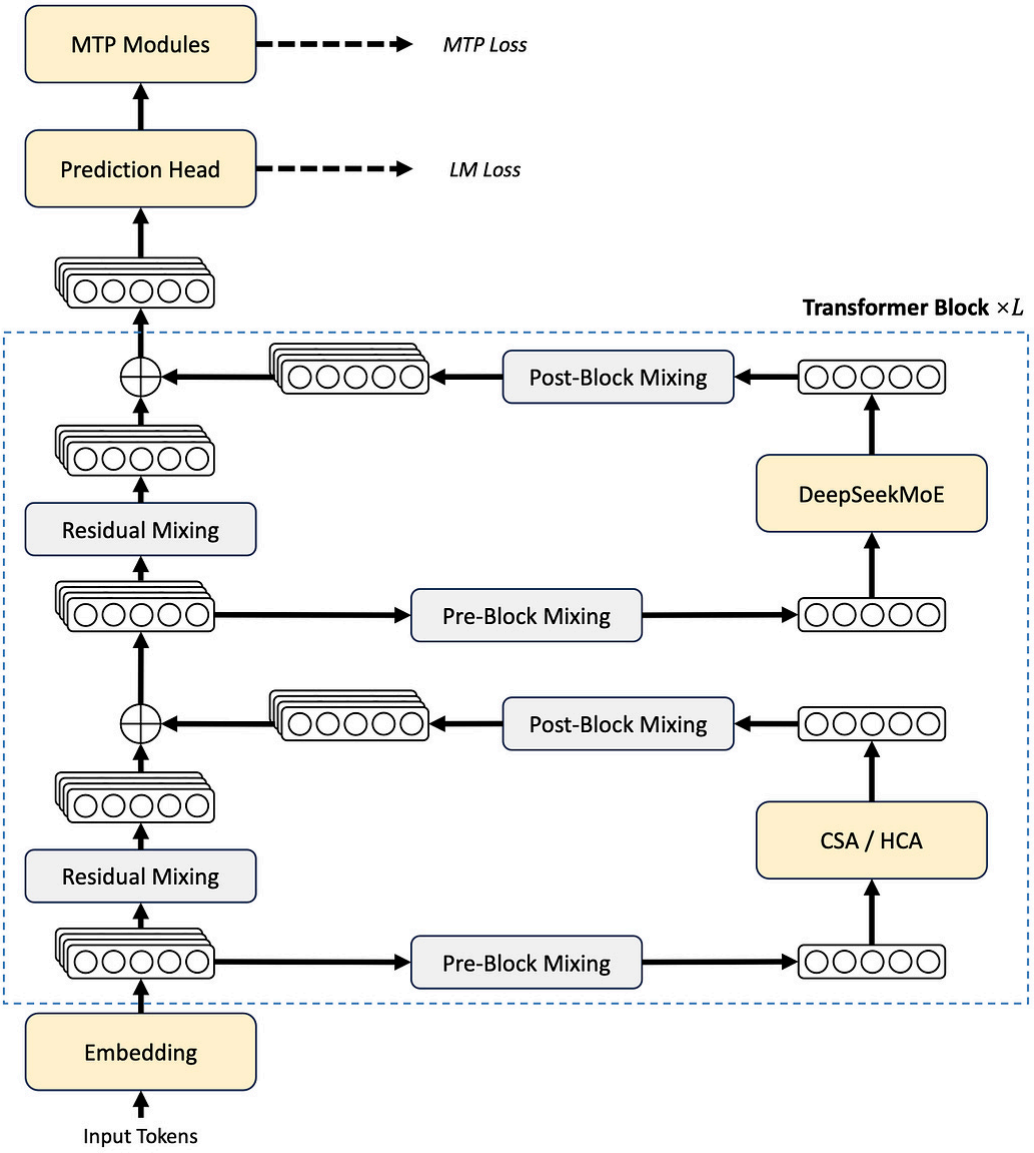


Source: DeepSeek

The biblical story that gave the Ascend 950 its internal codename ends with the giant face-down. But the Goliath in that story stood still and let David sling stones at him, whereas Nvidia's Goliath is constantly in motion, ships a new architecture every year and improves existing architectures. Huawei has proven it can sling a stone on Day 0; whether it can fell a moving giant is yet to be seen.

DeepSeek V4 Architecture Deep Dive and Co-Design

QQ:1413221435
QQ:1413221435



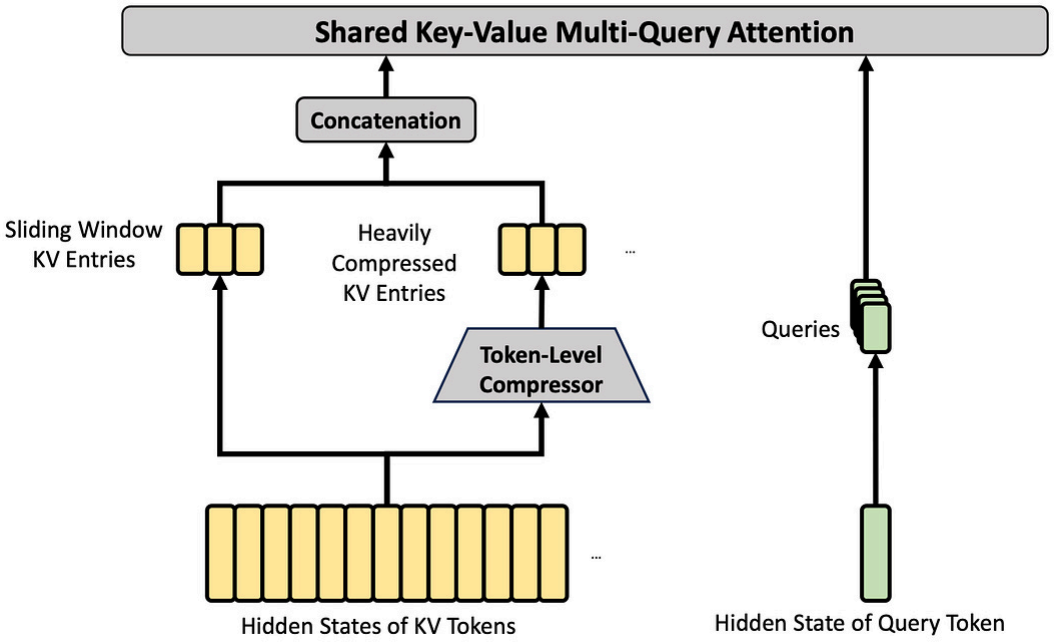
Source: DSv4 Tech report

Inference Optimizations for 1M Context Length

DeepSeek v4 features Compressed Sparse Attention (CSA) and Heavily Compressed Attention (HCA), walking away from Multi-head Latent Attention (MLA). The design is heavily motivated by KV cache size reduction.

In essence, HCA's KV cache consists of a sliding window of the KV embeddings and a set of compressed KV entries, where each entry compresses key / value into one and across m' tokens ($m' = 128$ for DeepSeek V4 Pro).

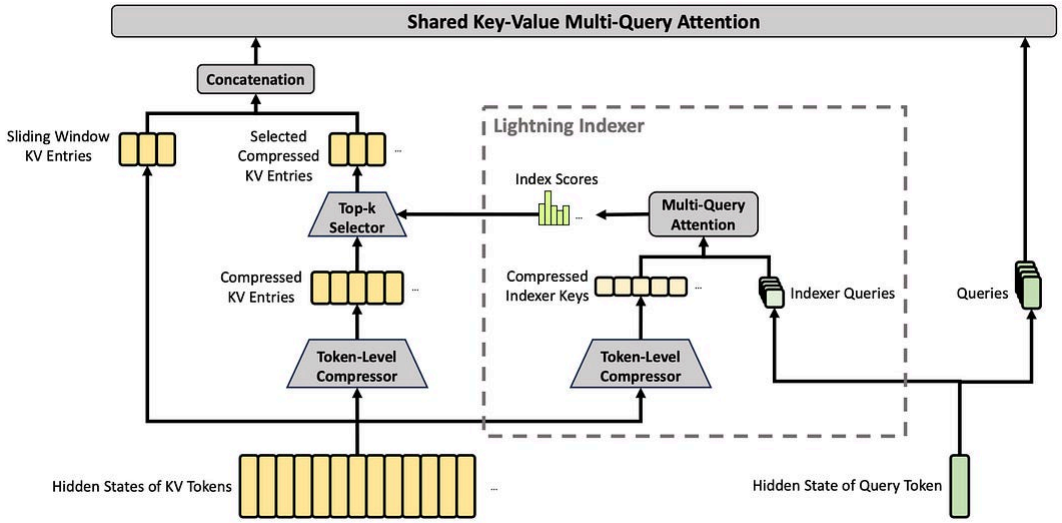
QQ:1413221435
QQ:1413221435



Source: DSv4 Tech report

QQ:1413221435

CSA uses the same KV cache compression technique as HCA, but with a lower compression rate (m=4). CSA also applies sparse attention on the compressed KV entries by using a lightning indexer to select tokens to attend to. The sparse attention inherits DeepSeek Sparse Attention in DeepSeek v3.2.



Source: DSv4 Tech report

By interleaving CSA and HCA, DeepSeek v4 aggressively compressed KV cache size, resulting in 50x KV cache reduction at 1M context length.

However, the novelty of CSA and HCA creates KV cache management challenges for serving frameworks. For example, vLLM's KV cache memory allocator implements complex strategies to ensure efficient memory loading patterns and support serving

QQ:1413221435

features like prefix caching. This includes setting a logical block size that divides the KV compression rates of both CSA and HCA, and a page size bucketing strategy to avoid memory fragmentation due to storing the KV cache, compressor states, indexer KV, each having a different size per entry.

Determinism

To ensure RL training stability, DeepSeek went all in on making computation deterministic. This effort shows when looking intoGPU kernels and their rollout infrastructure. DeepSeek wrote custom kernels for all operations to achieve batch invariance by enforcing a specific reduction order regardless of batch size. This includes batch-invariant split KV attention forward, GEMMs, and MoE backward kernels. Batch-invariant kernels come at a performance loss because using them prohibits many popular algorithmic techniques that don't guarantee deterministic reduction order. DeepSeek mitigates the performance loss by writing kernels that are tailored for their workload, for instance specializing kernels for matrix shapes. On the rollout infrastructure side, DeepSeek focused on fault tolerance to make all rollouts reproducible. DeepSeek built a token-granular write-ahead log for each generation request, so any request preempted during either prefill or decode can be resumed without recomputation.

QQ:1413221435

MegaMoE

The DeepSeek V4 release also included a new fused MoE kernel that achieves better overlap of all of the operations in the MoE layer. MoE with expert parallelism consists first of a token dispatch all-to-all followed by Linear1, Activation, Linear 2, and finally a token combine all-to-all. Linear 1 and Linear 2 are grouped GEMM operations where each expert in a given rank has its weights applied to its routed tokens. The authors mention in the DeepSeek V4 paper that other implementations overlap/interleave the token dispatch with Linear 1 and the Combine with Linear 2, but there is still a sync across all experts at the operation boundaries, between Linear 1, Activation, and Linear 2. MegaMoE instead splits experts into waves and schedules each wave separately, allowing for finer-grained overlapping of each of the operations and resulting in more of the communication latency being hidden. This is reminiscent of compute-comms fusions such as distributed GEMM, where a compute kernel and a dependent communication kernel are overlapped by breaking the workload into smaller pieces and pipelining to hide communication latency.

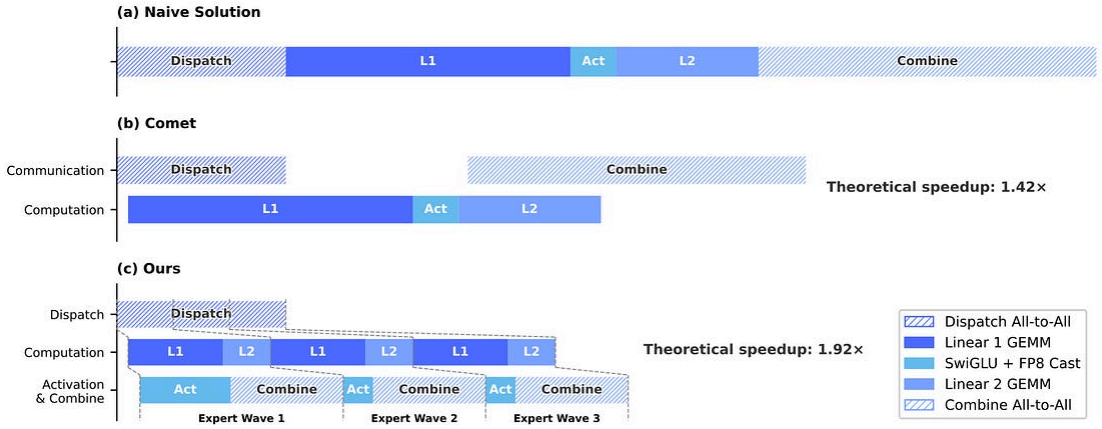
QQ:1413221435

SemiAnalysis is a reader-supported publication. To receive new posts and support my work, consider becoming a free

or paid subscriber.

Subscribed

The paper claims a theoretical speedup of 1.92x over the naive kernel in the DeepSeek v4 Flash configuration, which means that the naive kernel must spend close to 50% of its time on Dispatch and Combine communication!



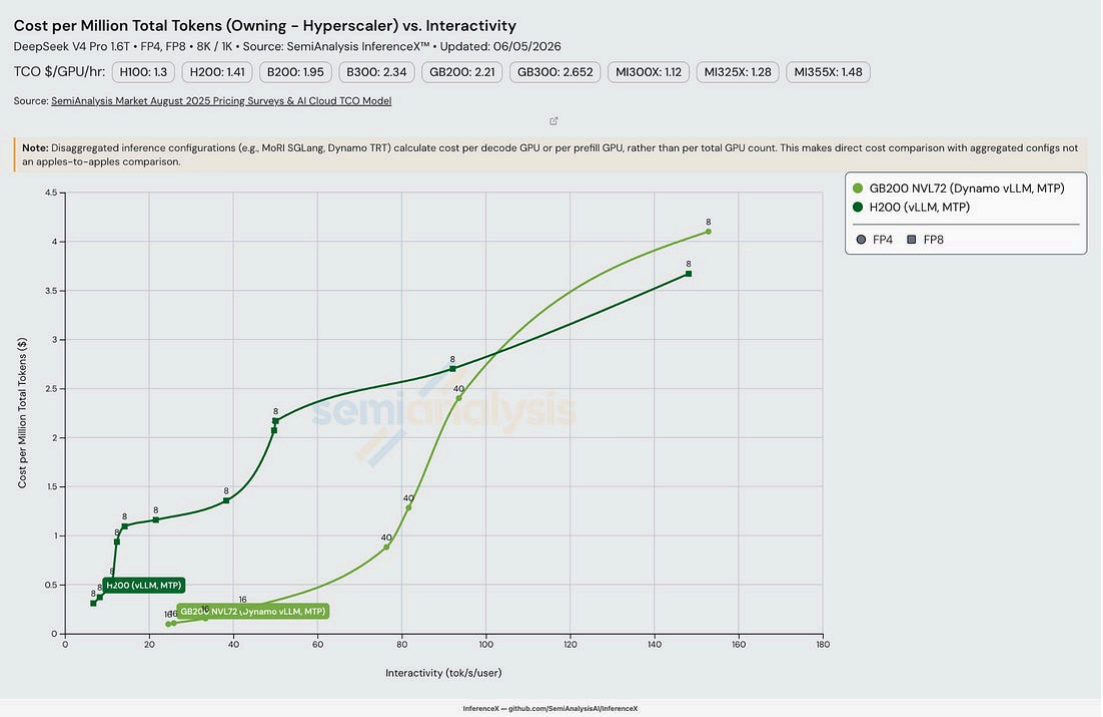
Source: DSv4 Tech report

Now that we have discussed performance benchmarks in detail, let’s discuss total cost of ownership and cost per token when running DeepSeek v4 on H200 and the GB200 NVL72.

When looking at the common serving range for DeepSeek v4 Pro around 40-60 tok/s/user interactivity, we see that thhe GB200 NVL72 is over 10x cheaper when it comes to cost per million tokens. This is because the GB200 NVL72 being able to take advantage of the NVL backplane which connects 72 GPUs together at 18x the speed of the B200’s InfiniBand, enabling the use of optimizations such as wide expert parallelism.

QQ:1413221435

QQ:1413221435



Source: [InferenceX]

QQ:1413221435

QQ:1413221435