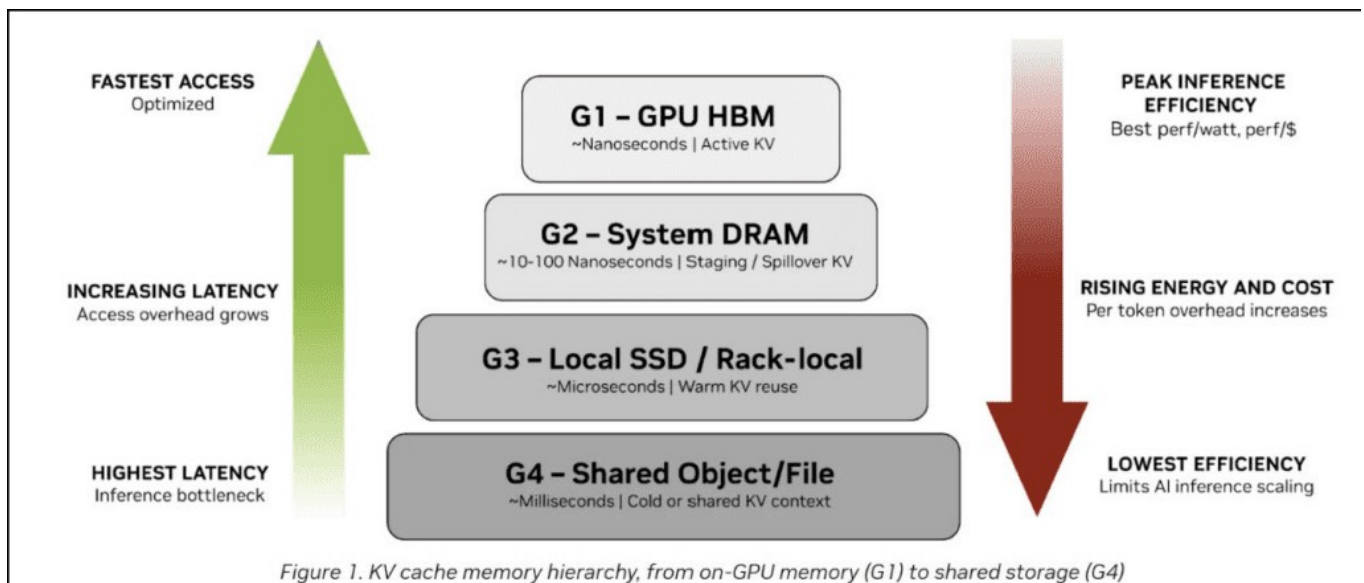


# GPUs Changed Everything. Storage Is the Bottleneck Again.

Julien Laurenceau | Published 2026-01-23 | Updated 2026-10-21

<https://pepitedata.com/gpu-ai-storage-bottleneck/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien\_Lau



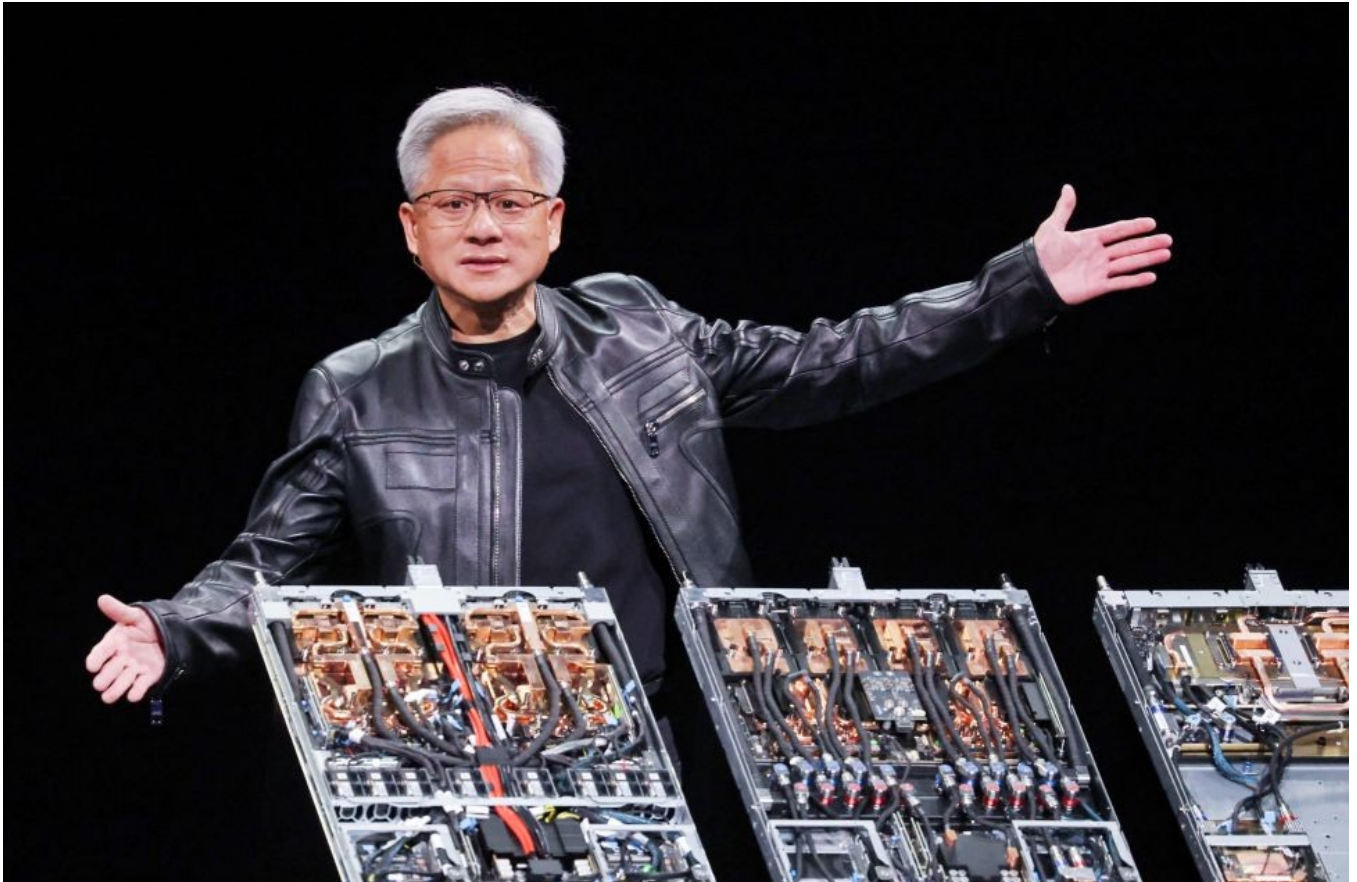
## GPUs are no longer the bottleneck. Data movement is.

The bottleneck in AI infrastructure has moved from compute to data movement. The claim was articulated in an online talk I attended recently, and it matches what I have seen for years across very different systems: raw compute stops being the constraint once accelerators are fast enough to starve.

That message, strongly emphasized by **Jensen Huang**, can be summarized in one sentence.

“The bottleneck is no longer compute.  
The bottleneck is data movement.”

This resonates deeply with my own path, from HPC to Big Data and now large-scale DevOps and infrastructure.



## A familiar lesson from HPC

The same lesson predates AI. I started my career in numerical optimization and HPC, where raw compute power alone never guaranteed performance: data locality, memory bandwidth and I/O patterns decided the real runtime. The GPU era repeats the pattern at a larger scale.

If memory, storage, and interconnects were not balanced, the system stalled.  
Adding more cores only made the imbalance more visible.

Years later, when I moved from physics into Big Data, I faced a choice.  
Focus on models and data science, or focus on distributed systems and storage.

I chose systems and storage, because that was where global performance was decided.  
With hindsight, that choice feels even more relevant today.

## GPUs changed scale, not the rules

GPUs changed the scale of what is possible, not the fundamental rules of systems engineering. Data movement, caching and storage bandwidth still decide latency and cost, and an unoptimized cache makes GPU spend rise silently while the symptom looks like a software problem.

Modern GPUs are no longer simple accelerators.  
They sit at the center of the system and pull data at extreme rates.

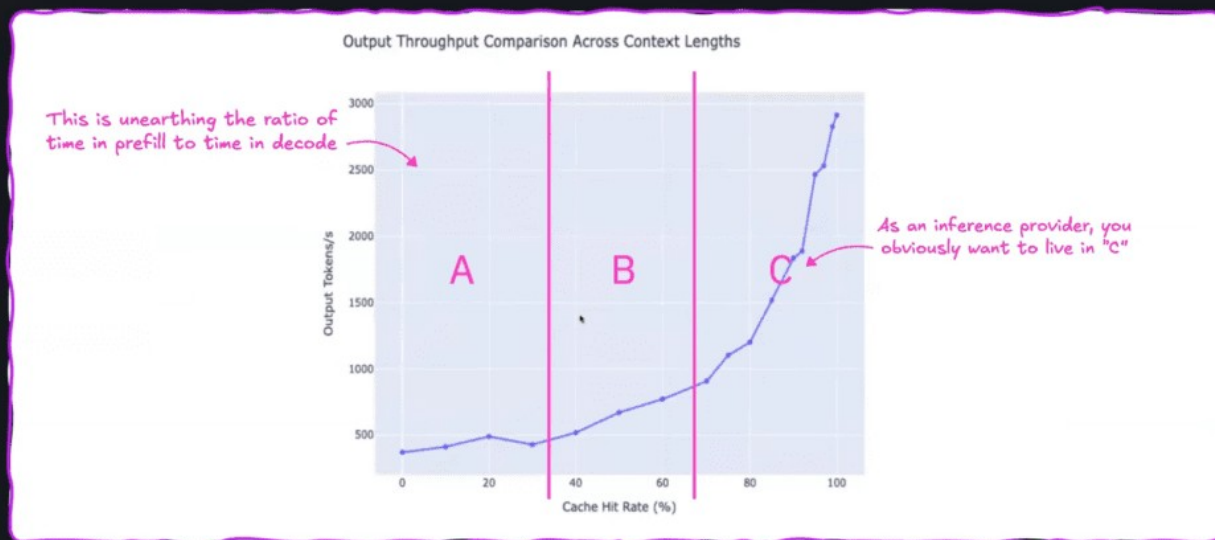
When everything works, the gains are spectacular.  
When data paths fall behind, GPUs wait.

This is exactly what we now observe in many AI platforms.

## Large context windows make the problem worse

Large context windows make the data path worse, not better. Recent analysis and benchmarks shared by **WEKA** show prefill increasingly dominating decode as context grows, which turns storage bandwidth and cache behavior and not raw compute, into the deciding factors for inference latency.

## Prefill becomes dominant at larger context lengths



As context windows grow, **prefill increasingly dominates decode**.

This means a much larger amount of data must be accessed before token generation even begins.

Two consequences follow:

- storage bandwidth becomes critical
- latency becomes extremely sensitive to cache behavior

At large context sizes, inference is no longer compute-bound. It is bound by how efficiently data can be fetched and reused.

## Cache hit rate becomes the key metric

Cache hit rate is the key metric of the new regime. The relationship WEKA measured ties hit rate directly to throughput: low or unstable rates collapse throughput and spike latency while GPUs idle, and a high stable rate unlocks a different performance level entirely.

When cache hit rates are low or unstable:

- throughput collapses
- latency spikes appear
- GPU utilization drops sharply

When cache hit rates are high and stable:

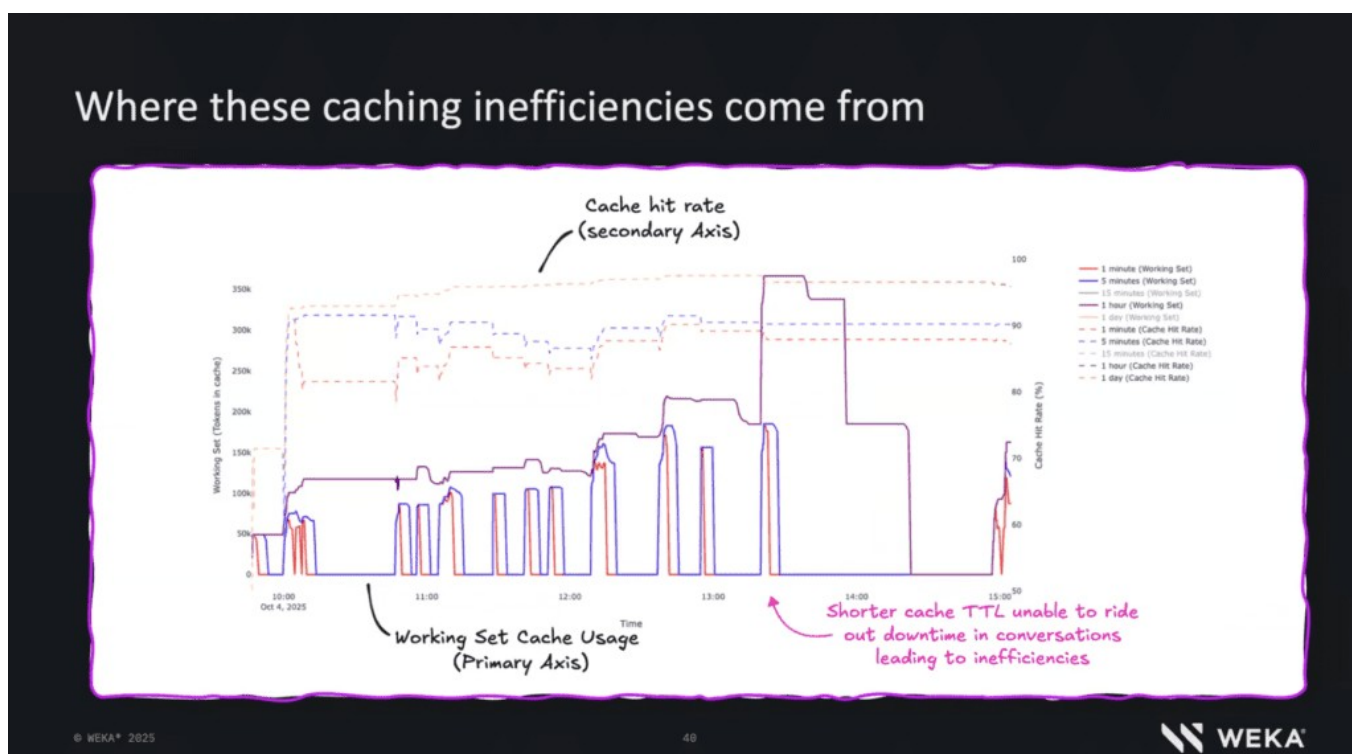
- throughput increases dramatically
- latency becomes predictable
- the system operates in a different performance regime

For an inference provider, this difference is existential.  
You want to operate in the high hit rate regime at all times.

This is not a model issue.  
It is a storage and caching issue.

## Where caching inefficiencies come from

Cache inefficiency is structural rather than accidental. Inference traffic is bursty, conversations pause, and working sets evolve as context grows. When cache time-to-live values are too short, idle periods empty the caches and hit rates fall exactly when traffic returns.



Another important point illustrated by WEKA is that cache inefficiencies are often structural.

Inference traffic is bursty.  
Conversations pause.  
Working sets evolve over time.

If cache TTLs are too short, caches empty during idle periods.  
When traffic resumes, cache hit rates drop and latency jumps.

As context windows grow, working sets grow with them.  
Short-lived caches are no longer sufficient.

This has nothing to do with GPU performance.  
It is a consequence of storage systems not being designed for large, persistent working sets with high reuse.

## The real infrastructure consequence

Larger context windows require more storage, and not merely more capacity. The real infrastructure consequence is storage designed for predictable, high-bandwidth access and efficient caching, because short-lived caches stop being enough the moment the working set grows with the context window.

If cache hit rates are not optimized:

- latency becomes unstable
- GPUs stall
- inference costs rise silently

From the outside, this often looks like a software or model issue.  
In reality, it is almost always a system design problem.

## A personal conclusion

After more than fifteen years working on large-scale systems, one conviction has only grown stronger: the biggest performance gains come from better data movement, not from better models. The bottleneck moved from compute to the data path before, and each generation of hardware relearns it.

The biggest performance gains rarely come from better models.  
They come from better data movement.

This was true in HPC.  
It was true in Big Data.  
It is now impossible to ignore in AI.

That is why Jensen Huang's message matters.  
And that is why the data shared by WEKA is so valuable. It makes the problem measurable.

## Final thought

We spend a lot of time talking about models, frameworks and GPUs, and almost no time on the plumbing that feeds them. The systems that win treat data movement as a first-class design problem, because that is where the performance and the money actually are.

In real production systems, compute is rarely the limiting factor.

Data paths almost always are.

That is where AI platforms either scale cleanly or quietly fail.

Video that triggered this reflection:  
<https://www.youtube.com/watch?v=o-etY6VLHZo>

## Credit

The cache behavior, prefill dominance, and inference insights discussed here are based on analysis and material published by **WEKA**.

See it on [LinkedIn](#)

[Discuss together](#)

*Update 2026-09-29.* This post is now part of [Performance and Cost Engineering for Data Platforms](#), the field guide to performance and cost engineering.

*Update 2026-10-21.* The file layout side of data movement, from Parquet tuned for random access to Lance, Vortex and the cloud-optimized formats, is covered in [Beyond Parquet and Avro: choosing a data file format by access pattern](#).

## Related posts

- [Performance and Cost Engineering for Data Platforms](#)

- [Beyond Parquet and Avro: choosing a data file format by access pattern](#)
- [Why your GPU infrastructure costs 40% more than it should](#)
- [Tesla's .SMOL format and why enterprise data lakes are architecturally wrong](#)
- [The hidden costs of using HDDs in on-premises MinIO deployments](#)

PDF snapshot of the public article. The online version is authoritative; links remain clickable.