Episode Book

EPISODE #6024

What's Actually Inside the KV Cache?

PUBLISHED	16 October 2026 at 10:52 UTC
DURATION	28:21
LISTEN	myweirdprompts.com/episode/kv-cache-inference-optimization-60443/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.5
BOOK GENERATED	19 October 2026 at 17:42 UTC

SYNOPSIS

Every token an LLM generates has to attend to every token before it, and the KV cache is what keeps that from being recomputed from scratch — while quietly becoming the thing that caps your context length, your batch size, and your latency. This episode walks through what's actually stored (keys and values, never queries), why prefill is compute-bound and decode is memory-bandwidth-bound, and the formula that turns a single Llama 2 conversation into two gigabytes of cache. Then it maps the whole lineage of fixes onto three multiplicative dimensions — entry size, sequence length, and layers — from Multi-Query and Grouped-Query Attention through DeepSeek's Multi-head Latent Attention, DeepSeek Sparse Attention, cross-layer sharing, and the inference-only tricks like PagedAttention and KV quantization that need no retraining at all.

TRANSCRIPT



Corn

What if I told you that the single most important number in AI right now isn't a parameter count, or a benchmark score, or a training cost... it's eight hundred and ninety bytes.



Herman

Eight hundred and ninety bytes per token.



Corn

Per token. And that number is the reason Daniel sent us this prompt. He wants to know about the KV cache. What's actually stored in it, how keys and values get generated from tokens in the first place, why any of it can be reused, and what inference would even look like without it. Then he wants the whole lineage of innovations since attention got popular. Multi-Query Attention, Grouped-Query Attention, DeepSeek's latent attention, quantization, sparse attention, cross-layer reuse, offloading across GPU memory and system RAM and SSDs. And then the case study he's actually circling: DeepSeek-V4.1-Flash and its Compressed Sparse Attention 2.



Herman

And the distinction he wants drawn, which I think is the sharpest thing in the whole prompt, is between architectural innovations that require training and inference optimizations you can bolt onto a model that already exists.



Corn

Right. Plus what all of it means for the economics of running agents. Daniel's framing is that this is a mundane caching mechanism that somehow became a frontier. I think he's right, and I think the reason it became a frontier is worth sitting with for a second.



Herman

It's worth sitting with because the cache is where the bill lives. Every token you generate has to attend to every token that came before it. The cache is what keeps you from recomputing all of that from scratch, but the cache is also the thing that has to sit in memory, get read every single decode step, and eventually get paid for.



Corn

So start at the mechanism. What is actually being stored?



Herman

Keys and values. That's it. For every attention layer, for every token already in the sequence, you keep the key tensor and the value tensor that layer produced. Nothing else.



Corn

And the reason it's keys and values and not queries is the part people get wrong.



Herman

Queries are not cached. A query only exists for the token whose attention output you're computing right now. It's used once and thrown away. Keys and values are the opposite. A key and a value produced by token forty-two stays relevant for token forty-three, forty-four, forty-five, all the way to the end of the sequence, because every future token has to attend back over it. So you keep those.



Corn

Walk through the two phases. Prefill and decode.



Herman

Prefill is when you feed the prompt in. All those prompt tokens get processed in parallel, and at every cache-producing layer, the model stores their keys and values. Then decode starts. Each new token computes only its own query, key, and value. It attends to everything in the cache plus its own new pair. Then its key and value get appended to the cache, and the next token does the same thing.



Corn

So the cache grows by one entry per layer per token, forever, until you stop.



Herman

Until you stop or until you hit the context limit. And the size of it is a formula, which is the one thing from this episode I'd actually tattoo on something. Two, times the number of layers, times the number of attention heads, times the dimension per head, times the bytes per number. That's per token.



Corn

Two because there are two tensors. Keys and values.



Herman

Two because there are two tensors. So put real numbers on it. Llama 2, seven billion parameters, half precision, batch size of one, four thousand ninety-six tokens of context. That cache is about two gigabytes.



Corn

Two gigabytes for a single conversation with a small model at a modest context length.



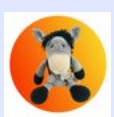
Herman

And that's before you batch anything. You want to serve thirty users at once, that's thirty copies of that, unless you're clever about sharing.



Corn

Here's the thing I want to flag early, because it changes how you hear everything after it. The cache removes the repeated prefix projections. It does not remove the attention work. It does not remove the memory traffic over the prefix you kept.



Herman

It removes the redundancy of recomputation. It does not make long context free. Every decode step still has to read the whole cache out of memory and do the attention math over it.



Corn

So the cache is what makes this practical, and the cache is also what makes long context expensive. Both true at once.



Herman

And which one dominates depends entirely on whether you're in prefill or decode. That distinction is the hinge for the entire rest of the episode.



Corn

Explain it.



Herman

Prefill is a matrix-matrix operation. You've got thousands of tokens going through at once, and the hardware loves that. It's compute-bound. You're saturating the arithmetic units. The GPU is doing actual math the whole time.



Corn

And decode.



Herman

Decode is a matrix-vector operation. You're generating one token. One. There's barely any arithmetic to do, and what you're actually waiting on is moving the weights, and moving the keys and values, from memory into the compute units. So decode is memory-bandwidth-bound. Your latency is a function of how fast you can move bytes, not how fast you can multiply.



Corn

Which means a bigger cache doesn't just cost you storage. It costs you latency on every single token.



Herman

And it caps how many users you can serve at once, and it caps how long a context you can afford. Three constraints, one mechanism.



Corn

Alright. So the field has spent three years attacking that. And the way DeepSeek frames it in their own paper is that there are three multiplicative dimensions you can squeeze.



Herman

Entry size, sequence length, and layers. Those are the three. And I love this framing because it makes the whole zoo of techniques legible. Every trick you've heard of is attacking one of those three, and because they're multiplicative, you can stack them.



Corn

Entry size first.



Herman

Entry size is how big each individual cached item is. Multi-Query Attention is the blunt version. Standard multi-head attention gives every query head its own key and value head. MQA throws that away and shares one single key and value head across all the query heads.



Corn

One head to serve them all.



Herman

One head. Massive reduction. The cost is quality, and you have to train for it. Grouped-Query Attention is the compromise that won. Instead of one shared head, you group the query heads and give each group its own key and value head. Llama 2 seventy billion uses it. So does basically everything since.



Corn

And both of those require training.



Herman

Both require training or uptraining. NVIDIA's number is about five percent of the original training volume to uptrain a model into GQA. It's not free, but it's not starting over either.



Corn

Then DeepSeek's move on entry size is Multi-head Latent Attention.



Herman

MLA, from DeepSeek-V2. Instead of caching full key and value vectors per head, you project everything down into a small shared latent vector and cache that. You reconstruct what you need at attention time. That cut the cache by ninety-three point three percent, and it took max generation throughput up by a factor of five point seven six versus their previous sixty-seven billion model.



Corn

Ninety-three percent. So the cache is roughly a fifteenth of what it was.



Herman

Roughly. And the interpretability work since is interesting, because a group looked at what survives in that latent bottleneck. Entity identity survives at about ninety-eight percent retention. Position information is what gets thrown away. So the compressed latent keeps the what and discards the where.



Corn

That's a strange thing for a cache to decide.



Herman

It's a very strange thing for a cache to decide, and I don't think anyone fully predicted it.



Corn

Now the sequence dimension.



Herman

The sequence dimension is about not keeping one entry per token. You compress every m tokens into a single entry, or you decide which tokens to read at all. Sparse attention lives here. DeepSeek Sparse Attention, from V3.2, adds what they call a lightning indexer. It scores the prefix and keeps a learned top-k subset. In the released code that k is two thousand forty-eight.



Corn

So you've stopped reading the whole prefix.



Herman

And that's a completely different move from compression. MLA compresses what you store. DSA changes which stored positions you read.



Corn

Two different levers on the same dimension.



Herman

And they compose. You can store compressed entries and then only read a subset of them.



Corn

Now the third one. Layers.



Herman

Layers is the newest, and it's the one where there's still real room. The question is: why does every layer need its own cache? Adjacent layers in a transformer are computing fairly similar things. What if they shared?



Corn

Cross-Layer Attention.



Herman

CLA, from a paper last year. It shares key and value heads between adjacent layers. That's another factor of two. Then MLKV extended the idea to share across more layers. They report up to six times reduction versus MQA.



Corn

So you're amortizing one cache across several layers.



Herman

The cost is that you're forcing those layers to agree about what to attend to, and that's the thing that has to be learned.



Corn

And this is all before we get to the part that requires no retraining at all.



Herman

Which is the other half of Daniel's question, and it's the half that matters most to people running models they didn't train.



Corn

Give me the taxonomy cleanly.



Herman

Architectural changes need training or uptraining. MQA, GQA, MLA, cross-layer attention, FP4 quantization-aware training, all of it. Inference-only optimizations apply to a frozen model. PagedAttention, FlashAttention, KV quantization after the fact, eviction policies, offloading. Those you can turn on tomorrow.



Corn

PagedAttention being the vLLM one.



Herman

PagedAttention, from the vLLM paper at SOSP in twenty twenty-three. The insight is that the naive way to allocate cache memory wastes enormous amounts of it, because you reserve a contiguous block for the maximum possible sequence length and most sequences never get there. So you over-provision, and you fragment. PagedAttention manages the cache in fixed-size non-contiguous blocks, like pages in an operating system. The waste disappears, and because you're not wasting memory, you can run bigger batches.



Corn

Operating system paging for attention.



Herman

Down to the name.



Corn

So that's the infrastructure layer. What about just making the entries smaller without retraining?



Herman

KV quantization. You store the cache at lower precision. FP8 is the common default now. And moving down from there, NVFP4 compressed cache is about forty-five percent smaller than FP8.



Corn

And there's a middle path between fully architectural and fully inference-only, right?



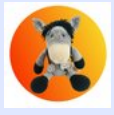
Herman

There is, and it's my favorite result in this whole space. MHA2MLA. Data-efficient fine-tuning that converts a multi-head attention model into an MLA model using only zero point three to zero point six percent of the training data.



Corn

That's a rounding error of a training run.



Herman

It cut Llama 2 seven billion's cache by ninety-two point one nine percent with a half a percent drop on LongBench. Half a percent.



Corn

So if you have a model you like and you have a weekend and a little data, you can get most of the way to the MLA cache reduction without pretraining from scratch.



Herman

That's the pitch, yes.

L

Larry

A public service announcement. Every day, millions of conversations are held without adequate memory hygiene. Are you leaving context on the table? Are you forcing your models to remember things they should have let go? Talk to your inference provider about proactive context retention management. Symptoms of poor cache hygiene include repeated tokens, elevated memory pressure, and conversations that slow down the longer they go. Ask whether structured forgetting is right for you. This message was brought to you by the Council for Responsible Sequence Management.



Corn

Did we book that?



Herman

No. And I own one.



Corn

You own a what? One what?



Herman

One of their memory hygiene kits. It was a subscription. I've had it for about four months and what it actually did was archive my old conversations into a folder I can't open. I've lost three recipes.



Corn

Noted. Back to layers.



Herman

Back to layers, and to DeepSeek, because that's where all three dimensions come together.



Corn

So V4.1-Flash. Give me the shape of the thing first.



Herman

Five hundred fifty-two billion parameter multimodal mixture of experts. Forty-layer architecture, but not a standard one. It's a Causal Encoder-Decoder, twenty encoder layers followed by twenty decoder layers. During prefill it activates eight billion parameters per token. During decode, sixteen billion.



Corn

And the encoder-decoder split isn't cosmetic.



Herman

It isn't. In a normal decoder stack, each decoder layer projects its own global keys and values from its own hidden state. In CED, the decoder's global key and value come from the final encoder hidden state instead. So you compute that projection once instead of twenty times.



Corn

Which nearly halves prefill computation.



Herman

Nearly halves it.



Corn

Then Compressed Sparse Attention 2. This is the centerpiece.



Herman

And the idea is that layers don't all need to do the same amount of work. CSA2 assigns each layer one of three static modes. Full mode computes the main keys and values, computes the indexer keys, and produces fresh top-k indices. Reindex mode reuses the main keys and values and the indexer keys from elsewhere, but re-scores with its own indexer query to get fresh top-k. And Reuse mode just takes the main keys and values and the top-k indices wholesale. No indexer work at all.



Corn

So some layers are paying full price, some are paying a partial price, and some are paying nothing.



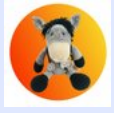
Herman

And the modes are static. Decided at design time, not dynamically per input. That's what makes it schedulable.



Corn

There's a hierarchical indexer too.



Herman

The first Full-mode decoder layer builds a candidate pool. Something like two thousand forty-eight blocks times eight positions, so sixteen thousand three hundred eighty-four candidates. Later indexers search that pool instead of the entire prefix. Which means the cost of the indexers deeper in the stack is bounded by the pool size, not by your context length.



Corn

So the deep layers stop caring how long the conversation is.



Herman

That's the trick. The pool is a fixed budget no matter how big the context gets.



Corn

Now the cache itself. This is the eight hundred ninety byte number.



Herman

FP4 main key and value caching. MXFP4 format, E2M1 with one E4M3 scale per sixteen channels. It's NVFP4 style minus the second-level global scale. And crucially, it's applied through quantization-aware training, so the model learns to live with it rather than having it bolted on afterward. Dequantization happens before attention, so there's no native FP4 matrix multiply needed anywhere.



Corn

And then SWA Bounded Replay, which is the weird one.



Herman

And honestly the most philosophically interesting thing in the paper. Sliding window attention normally means you only keep the most recent n tokens of key and value for the layers that use it. That's a real cache for those layers.



Corn

And they don't keep it.



Herman

They replay it. To reconstruct the sliding window cache for a decoder layer, instead of storing the window for every layer, they re-run the most recent one hundred twenty-eight tokens through the encoder and reconstruct it. So they trade compute for storage.



Corn

How much storage?



Herman

The naive cost of persisting sliding window caches over L layers is L times n_win entries. Bounded replay is n_win. So it's L times less, and L here is twenty.



Corn

That's a twentyfold reduction on that piece of the cache for the cost of replaying a hundred twenty-eight tokens.



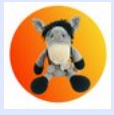
Herman

Which on modern hardware is basically free.



Corn

Give me the aggregate numbers.



Herman

Eight hundred ninety bytes per token for global key and value. That's about a quarter of V4-Flash. Persistent cache is around an eighth. And from DeepSeek-V1 to V4.1-Flash, per-token global key and value is down by a factor of four hundred thirty-seven. And compared to V4-Flash, roughly four times better.



Corn

One more thing on the architecture, because I think it's the freshest part. V4.1-Flash drops the hybrid entirely. Pure CSA2.



Herman

That's right, V4 had a CSA and HCA hybrid, and V4.1 goes pure CSA2. And the paper makes a claim I find a little bold. They argue that prior layer-reuse work, things like IndexCache and YOIO and HySparse, only covers all three multiplicative dimensions when you do what CSA2 does. Sharing the main key and value across layers, and sharing the indexer keys, and reusing the top-k selections across layers. All three at once.



Corn

So the layer dimension isn't just the newest. It's where you can still find multiple factors stacked.



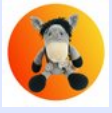
Herman

Which is why I think the entry-size and sequence-dimension techniques are starting to look like settled ground.



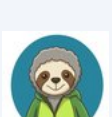
Corn

Now the systems side, because that's where the three-week-later number comes from.



Herman

vLLM put up a blog post on October seventh. Three weeks after V4.1-Flash shipped. Five point three times throughput improvement, one point nine times speedup at low concurrency.



Corn

Five times in three weeks, on a fixed model.



Herman

No retraining. All of it from SWA bounded replay integrated with CUDA graphs and kernel fusion. And then on top, thirty to forty percent cut in prefill computation, and around seventy percent drop in time to first token at roughly a hundred thousand tokens of throughput.



Corn

That's the case for the systems side of this being its own frontier.



Herman

DeepSeek co-designed their kernels with the model. MegaAttention, Mega-mHC, DeepSelect. Those are built for the architecture. vLLM's five times is pure systems work on a frozen architecture it didn't design. Both directions are producing large gains, and they're no longer separable in practice.



Corn

There's a serving wrinkle I want to hear about, because I think it's the most underrated consequence of all this. Something about how you have to parallelize these models differently.



Herman

Yes. This is the DP versus TP thing. Normally to serve a big model you split it across GPUs, and one of the standard ways is tensor parallelism, where individual tensors get split across devices. For a model with standard attention, that works fine.



Corn

But MLA-style models have a shared latent.



Herman

They have a shared latent across heads, which means if you split the heads across GPUs with tensor parallelism, each GPU ends up needing its own copy of that shared latent cache. You're duplicating the very thing you spent all this effort compressing.



Corn

So you pay the compression and then hand the savings straight back to duplication.



Herman

That's exactly what happens. So vLLM uses data parallel attention for V4.1-Flash instead. Split the attention work across devices, share the cache. It's a serving decision that falls directly out of an architecture decision, and it's not obvious until you hit it in production.



Corn

Let me go back to something you said earlier and pull it apart. Approximation.



Herman

Go ahead.



Herman

SWA Bounded Replay is not bit-exact. It reconstructs an approximation of the cache, and it does that on purpose. The whole thing trades exactness for storage. And vLLM checked it on GSM8K and GPQA and found no meaningful accuracy difference, with gaps well within a standard error and a half.



Corn

Which is a real shift from where this field started.



Herman

It's a real shift. FlashAttention's entire selling point was that it was exact. It was faster and it gave you the same answer, to the bit. That's why it slid into existing models so easily. And now the framing has moved to: this is approximately the same, and the approximation is worth it.



Corn

Where's the line, though?



Herman

I don't know where the line is. That's my honest answer, and it worries me a little. If you're having a casual conversation with a chatbot, an approximation that loses a bit in the tail is fine. If you're doing retrieval over a legal document where the answer hinges on a clause on page four hundred, and the approximation quietly dropped that key and value, you'll never know. The output will look just as confident.



Corn

That's a hard thing to test for, because you'd have to know what the model would have said if it had seen everything.



Herman

And by definition you can't. So you're betting that the reconstruction is close enough, based on benchmark gaps that are inside the noise. It's not that they're wrong. It's that the confidence has to come from somewhere, and it's coming from aggregate benchmark scores.



Corn

And their published numbers are self-reported.



Herman

There's no independent, widely-cited benchmark that isolates bytes of cache per token as a cross-vendor metric. DeepSeek's own figure is the closest thing we have. Which doesn't make it wrong, but it means it's not apples-to-apples with anyone else.



Corn

Now the economics, because that's where Daniel's question ends.



Herman

The paper is very direct about this. Large key and value caches strain both HBM and SSD capacity, plus the bandwidth to move data between them. And they say those compute, storage, and bandwidth demands together are the primary bottleneck to lowering deployment costs further.



Corn

And their announcement is even more direct, I think.



Herman

Their announcement ties it straight to agents. The line is that cache-hit charges often account for a large share of agent costs, and compressing the cache cuts those costs significantly.



Corn

That's the sentence that explains why any of this got funded.



Herman

It explains everything. If you're running an agent that makes fifty tool calls against a fifty thousand token context, every single one of those calls re-reads that context. And if the provider is charging you for the cache, or the provider is paying for the memory to hold it, the cache is the line item. Compress the cache and you've compressed the agent's operating cost, without touching the model's weights or changing a single one of its



Herman (continued)

answers.



Corn

Nine hundred ninety bytes per token versus what a naive cache would cost at a million tokens of context.



Herman

At a million tokens, the difference between eight hundred ninety bytes per token and a few kilobytes per token is the difference between a context that fits on one machine and a context that needs a cluster.



Corn

That's the thing that strikes me. The context window isn't limited by the model's ability to understand a million tokens. It's limited by the memory cost of remembering them.



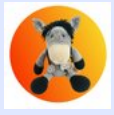
Herman

It's limited by memory, almost entirely, in serving. The model was trained on million-token sequences, but training is a different problem from serving a million-token conversation for thirty concurrent users.



Corn

One more caveat, and it's important.



Herman

Offloading. The whole story I've been telling you assumes you can move cache between GPU memory and system RAM and SSDs and get your capacity back. And the honest position is that vLLM didn't need to offload at all during their benchmark on V4.1-Flash. They expect it to only matter at higher concurrency with prefill-decode disaggregation. So the SSD tier of this story is real in principle and not yet demonstrated at scale in their runs.



Corn

Which is a refreshing thing for them to write, honestly. The benefit of the architecture is measured, and the benefit of the offloading stack is projected.



Hilbert

The number's wrong.



Herman

Which number?



Hilbert

Eight hundred ninety. You're comparing it to the wrong side.



Corn

The wrong side of what?



Hilbert

Of the ratio. Everyone treats a cache like a locker. Put the thing in, take the thing out, the locker's full. That's not what it is. I spent four years of my life being the man who explained to people that the size of a thing and the cost of a thing are different numbers. That was the job. I did it at a company you've never heard of, in an office above a printer repair shop, and I got very good at it.



Herman

What were you caching?



Hilbert

Documents. Same problem. Everyone wanted to keep everything, because keeping is cheap per item and forgetting is expensive per decision. And I'd tell them, we don't store it because we might need it. We store it because we've worked out what we can throw away. The number that matters is not bytes per token. It's bytes per token *that you'll still need*. And nobody measures that. Nobody can. You'd have to run the same conversation twice and diff the answers.



Corn

That's actually a real problem.



Hilbert

It's a real problem and there's a man in Waltham who still calls me about it. Every year. To tell me they never got the number. I told him, you'll never get the number. He didn't believe me. I think he still doesn't.



Herman

Wait. Did you say Waltham?



Hilbert

I said Waltham. The best cache is a filed document you never have to open, because it means you didn't need it. If you need to open it, the cache failed. You spent bytes on something you used. That's all of it.



Corn

That's a confusing slogan.



Hilbert

It's not a slogan. It's arithmetic. If you're reading from cache, the caching was wasted.



Herman

That's not how attention works. The whole point is that you read from cache.



Hilbert

Then you're doing it wrong, and you've been doing it wrong since before I could have told you.



Corn

Hilbert, if you never read the cache, what exactly did you store it for?



Hilbert

You stored it so that the person asking would stop asking. That's what a document's for. It's not for reading. It's for being there. There's a difference. There's a name for the difference. I don't remember the name. It was in the documentation.



Herman

Which documentation?



Hilbert

The one my brother-in-law had laminated. Sits on my desk right now. The whole stack.



Corn

The whole stack of documentation is laminated on your desk.



Hilbert

It holds the monitor up. That's not what I'm saying.



Herman

You're saying the number's wrong.



Hilbert

I'm saying the number's being read the wrong way. Eight hundred ninety bytes per token. Everyone hears "small." Nobody hears "small and being read from on every single token, forever, by every user you have." That's a hundred users times a hundred tokens a second times eight hundred ninety bytes. It's still the biggest line in the invoice. Keep telling me it's small.



Corn

He's not wrong about that part. That's essentially the whole HBM argument.



Hilbert

I'm aware it's the argument. I've been makin' it since before it was called anything.



Herman

Okay, that's fair. The per-token figure is small, and the aggregate bandwidth is what actually bites. That's a real correction.



Hilbert

Good.



Corn

Hilbert, what do you want them to measure?



Hilbert

I want them to measure the answer that changes when you throw something away. That's the only number that matters. Everything else is just what you wrote down.