READ MODE · ANSWER FIRST

DeepSeek-V4: Million-Token Context

An interactive, visual explanation of DeepSeek-V4: Million-Token Context, from setup through the complete system.

CHEAT SHEET · 7 ESSENTIAL IDEAS

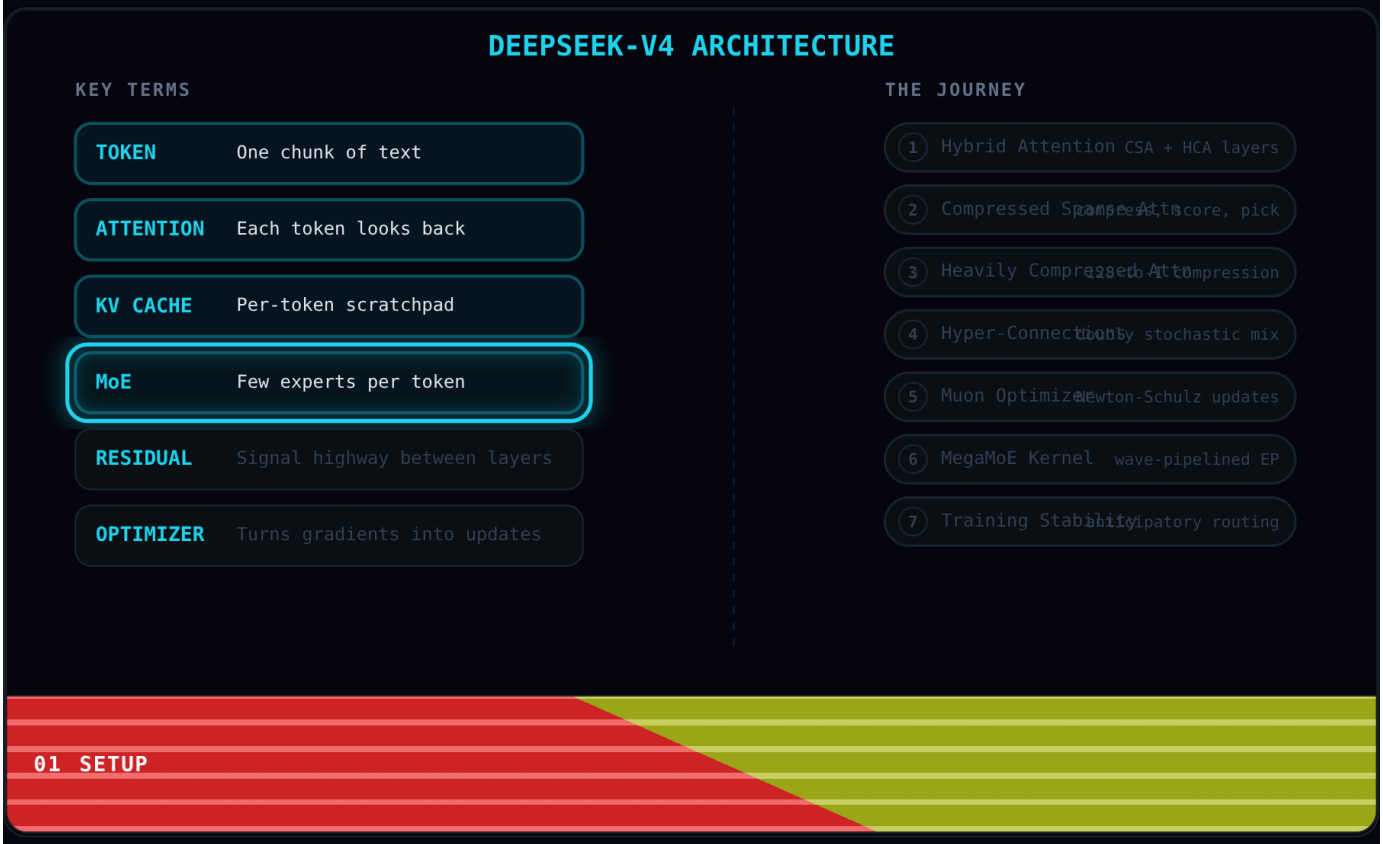
The whole story in 7 lines

DeepSeek-V4: Million-Token Context becomes easier to reason about when every stage is connected as one system.

1. DeepSeek-V4 interleaves two attention layer types. CSA compresses and then sparsely selects.
2. CSA compresses every m tokens into one KV entry, then a lightning indexer scores each compressed block against the query, and a top-k...
3. HCA groups every 128 tokens into one compressed entry, shrinking a million-token sequence into roughly eight thousand entries.
4. mHC expands the single residual stream into four parallel channels, then uses a mixing matrix that is mathematically constrained to be...
5. Muon orthogonalizes the gradient matrix before applying it. It uses a hybrid Newton-Schulz iteration to approximate that...
6. MegaMoE splits experts into small waves and pipelines dispatch, linear-1, activation, linear-2, and combine across waves.
7. Anticipatory routing breaks the feedback loop between router and backbone that causes loss spikes.

STAGE 1 OF 9

Setup



Welcome. DeepSeek-V4 is a large language model built to handle a context of one million tokens. That is roughly a thousand pages of text in a single prompt. To make that feasible, the team replaced many pieces of the usual transformer recipe. Our job in this explainer is to understand those pieces.

First, a token is one chunk of text, roughly one word. A model reads and writes tokens one at a time. When we say "one million tokens", we mean the model can keep a million of these chunks in view at once.

Attention is the math that lets each token look at all the previous tokens to figure out what it means in context. It is powerful, but the cost grows with the square of the sequence length. At a million tokens, naive attention is completely out of reach.

A KV cache is a scratchpad the model writes during attention. For every token, it stores a Key and Value vector. Later tokens read from this scratchpad instead of recomputing. Large contexts mean a huge KV cache, so shrinking it is a big deal.

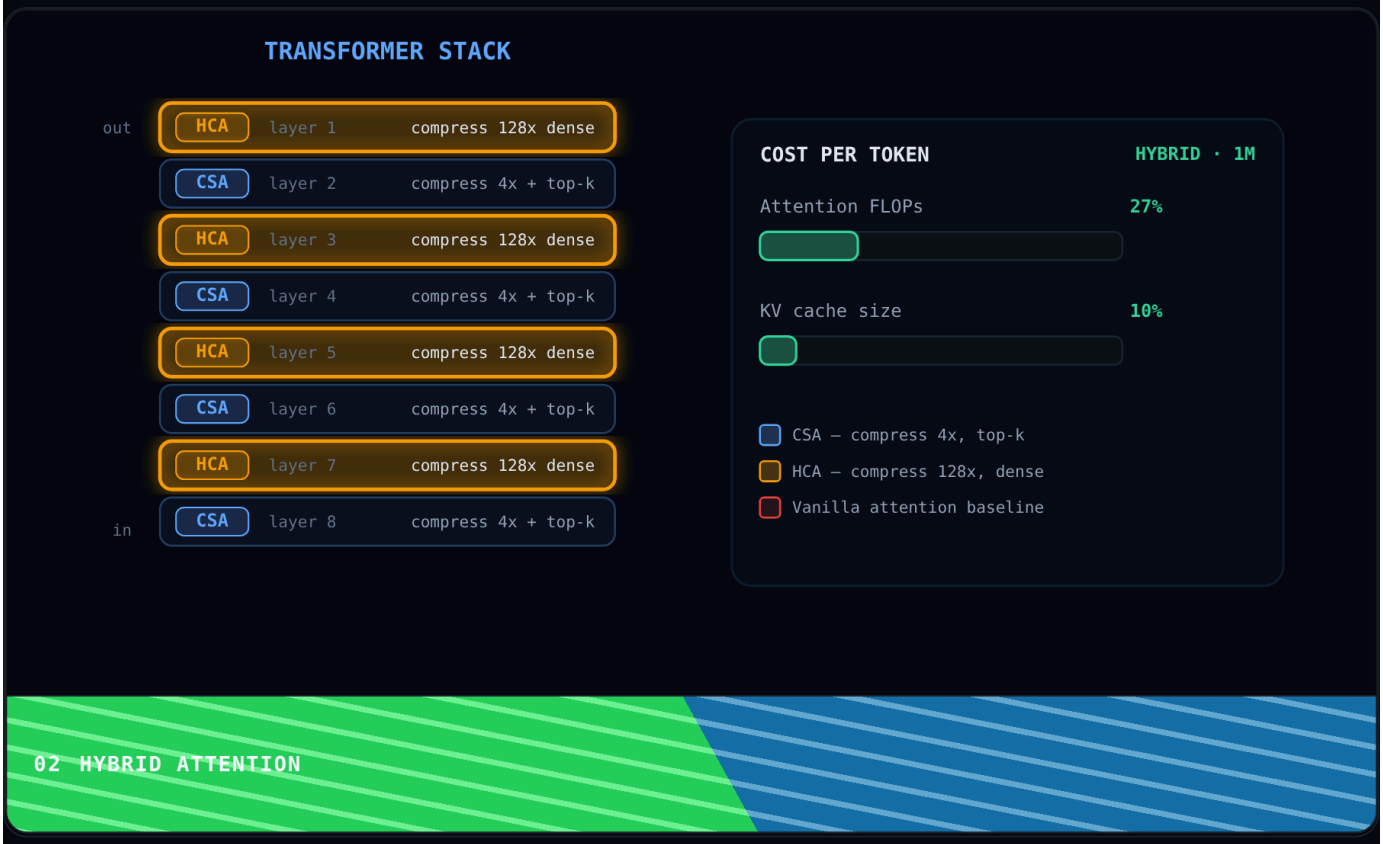
Mixture of Experts, or MoE, means the model has many small specialist networks called experts. For each token, only a few experts are activated. This lets the model have trillions of total parameters while only using a tiny slice per token.

A residual stream is the running signal that flows through all the transformer blocks, carrying information forward. An optimizer is the algorithm that decides how to nudge the weights during training. Both get upgrades in DeepSeek-V4.

Over the next seven stages we will see how hybrid attention crushes the KV cache, how a new residual connection keeps deep stacks stable, how the Muon optimizer converges faster, how a fused mega-kernel keeps GPUs busy, and what tricks stopped the training loss from blowing up. Let us begin with the most dramatic change: the attention rewrite.

STAGE 2 OF 9

Hybrid Attention



02 HYBRID ATTENTION

Here is the core problem this stage answers: a regular transformer has to compare every token against every other token. Double the context, and the attention cost quadruples. At one million tokens you are looking at a trillion comparisons per layer. Nobody can afford that.

DeepSeek-V4 does not use one attention design. It uses two, stacked in alternating layers. Use the Mode control in the sidebar to flip between vanilla attention and the hybrid layout. The difference at 1M context is night and day.

The odd layers run CSA, Compressed Sparse Attention. CSA groups every 4 tokens into one compressed entry, then uses a sparse selector to attend only to the most relevant entries. This preserves fine detail where needed.

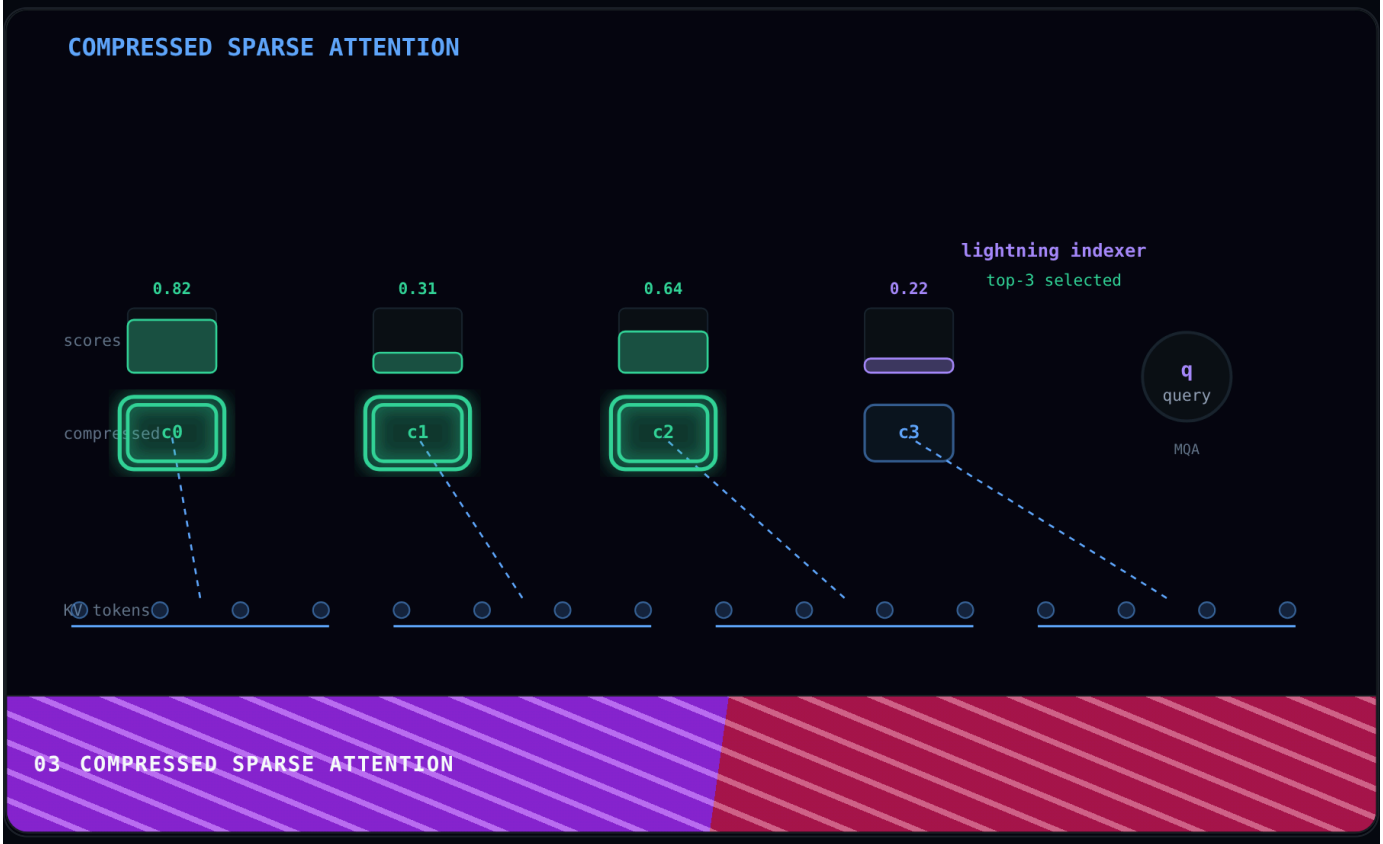
The even layers run HCA, Heavily Compressed Attention. HCA squashes every 128 tokens into one entry and then does dense attention over the shrunk sequence. It loses detail but is extremely cheap at long contexts.

By interleaving CSA and HCA, the model pays for detail only where detail matters. Try the Context control, too. At 4K tokens the two paths cost about the same. At 1M the hybrid uses only 27% of the FLOPs and 10% of the KV cache of the previous generation.

That is the big picture. The rest of the attention story is how CSA and HCA each work inside. Next we will zoom into CSA and watch the compressor plus the sparse top-k selector in action.

STAGE 3 OF 9

Compressed Sparse Attention



We just learned CSA preserves detail by being selective. How does that work in practice? This stage walks through the three pieces: compressor, indexer, and selector. Watch the flow from left to right.

On the left we have the raw KV tokens. The compressor takes every m=4 of them and produces a single compressed entry, using a learned softmax weighting over those four. This already cuts the sequence by 4x.

Now the interesting part. The lightning indexer is a small attention head that scores every compressed block against the current query. You can see scores lighting up above each block. Higher scores mean that block looks relevant.

The top-k selector takes those scores and keeps only the best few. Try the Top-k control in the sidebar. With k=2 the attention sees very few blocks, with k=5 it sees many. More blocks give more context but more compute.

Finally the core attention runs only over the selected compressed entries. Each query head attends to the small picked set plus a sliding window of the very recent tokens. That combination gives the model both far-context recall and local precision.

So CSA is compress then cherry-pick. Compression cuts the length by 4x, sparse selection cuts it again by picking the top k blocks. Next we will look at the other half of hybrid attention, HCA, which takes a very different approach: compress much more and keep everything.

STAGE 4 OF 9

Heavily Compressed Attention

Remember how CSA compressed by 4x and then selected? HCA compresses by 128x and does not select. It is the cheap workhorse layer. You will see right away how much smaller the compressed band becomes compared with CSA.

Raw KV tokens stream in on the left. HCA slices them into non-overlapping windows of m'=128 tokens. Each window becomes one compressed entry. A million-token context becomes roughly eight thousand entries.

Now try the compression control. A smaller m' keeps more detail but grows the compressed sequence. 128 is the sweet spot the paper uses. The tradeoff: less detail per block, but far fewer blocks to attend to.

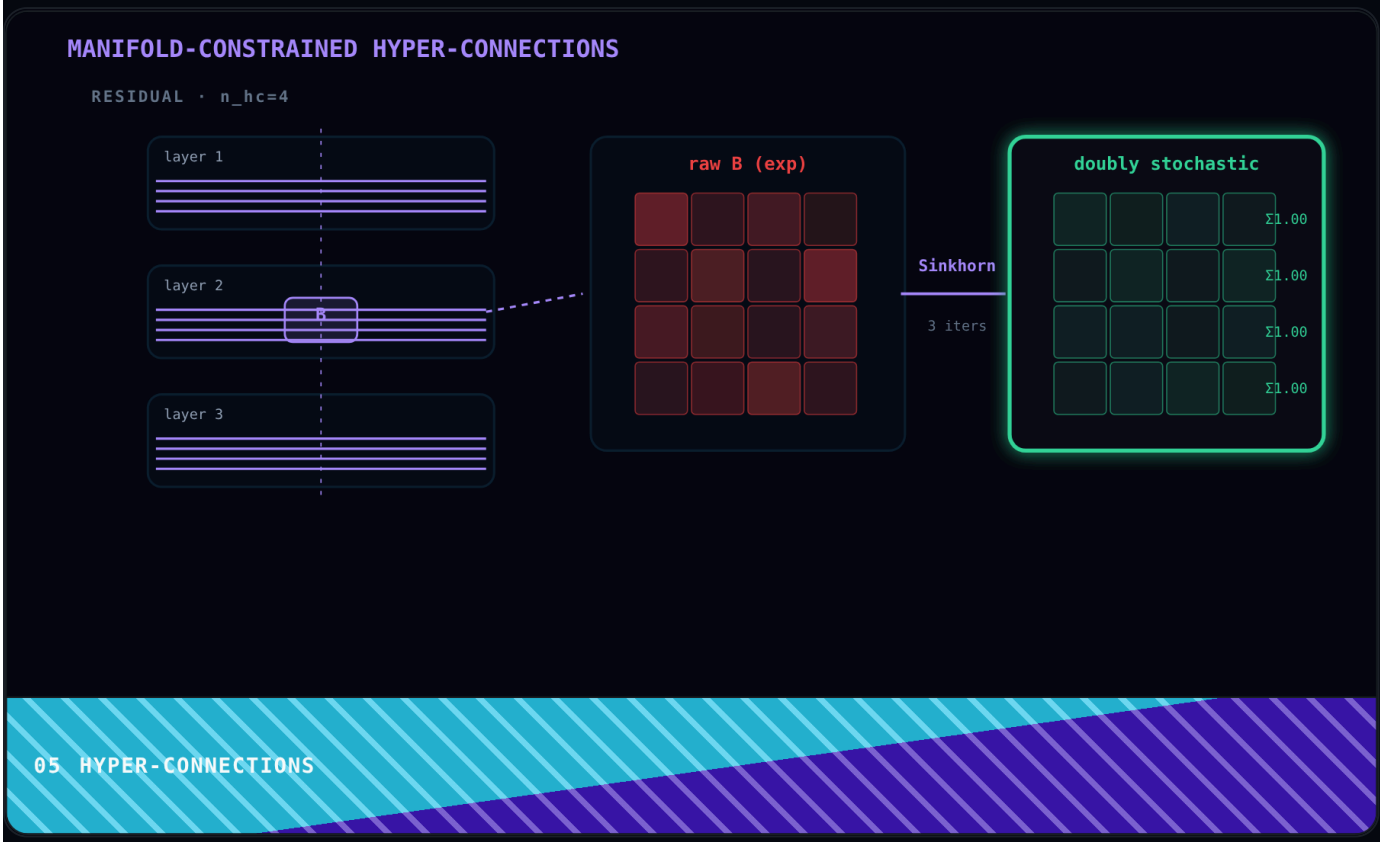
Because the compressed sequence is so short, dense attention is cheap. Every query simply attends to all compressed entries. This is the same vanilla multi-query attention math, just on a much shorter sequence.

HCA also keeps a small sliding window of the most recent uncompressed tokens. That way the model does not lose the fine-grained stuff that the aggressive compression washed out. Think of it as an HD strip for the here-and-now.

So HCA trades detail for cheapness. CSA trades compute for detail. Interleaving them lets each layer specialize. Next we leave attention behind and look at a subtle but surprisingly important change to how signals travel between layers: manifold-constrained hyper-connections.

STAGE 5 OF 9

Hyper-Connections



05 HYPER-CONNECTIONS

A regular transformer has one residual stream: a single vector that flows through every layer, adding contributions. For very deep networks, that stream can explode or vanish. DeepSeek-V4 replaces it with multiple parallel streams plus a careful mixing rule.

Here we show n_hc=4 parallel channels flowing down the stack. Each layer mixes them using a 4x4 matrix. That mixing is what lets the channels exchange information. The mixing matrix itself is the innovation.

On the right we see the raw B matrix. It is generated on the fly from the input, so every layer and every token gets its own B. At this point the numbers are arbitrary, and if used directly they would amplify the signal chaotically.

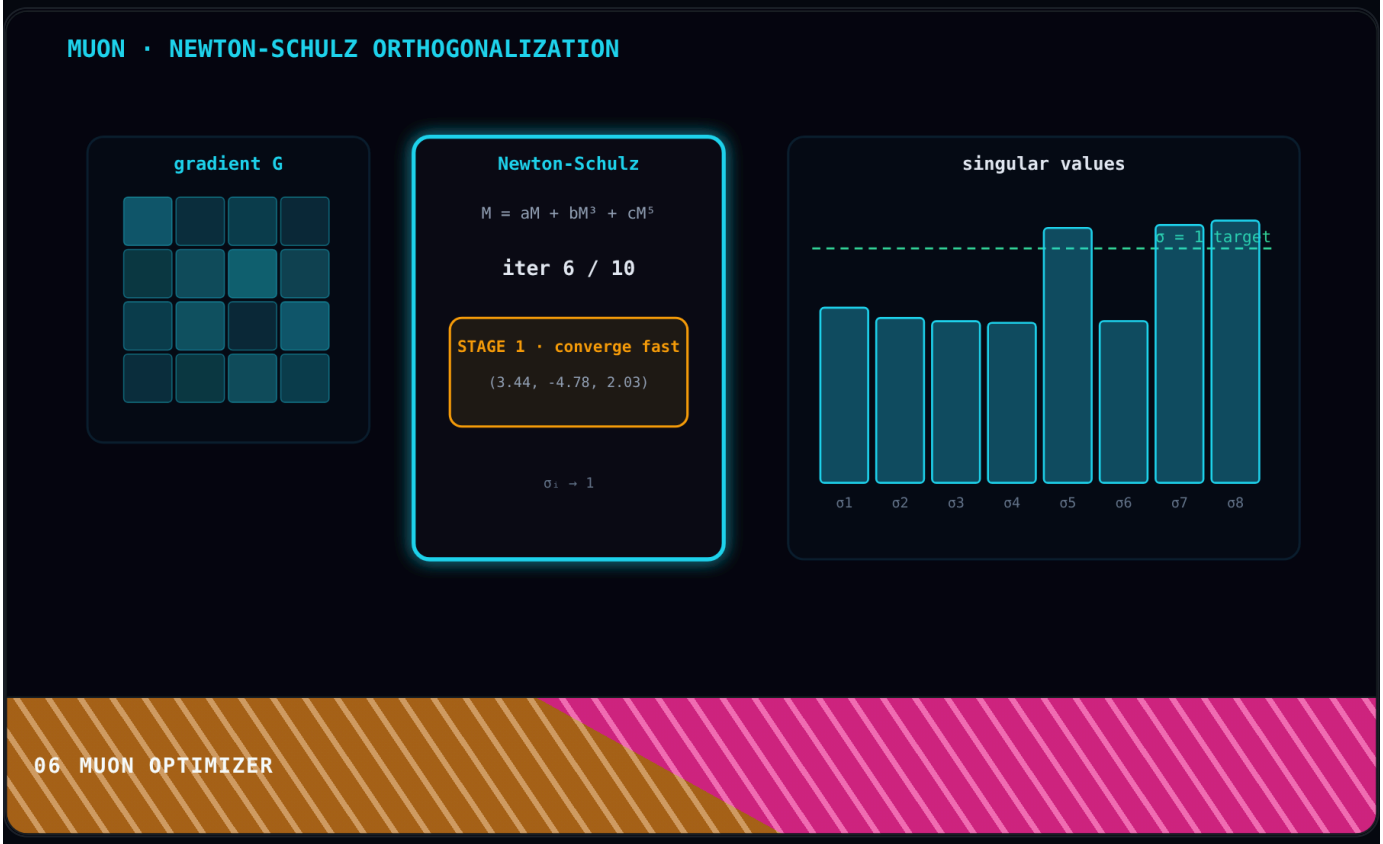
Now watch the Sinkhorn-Knopp algorithm do its job. It exponentiates the raw values to make them positive, then alternates normalizing rows and columns to sum to one. After a few passes you get a doubly stochastic matrix: a clean probability-mixing operator.

Why this shape? A doubly stochastic matrix has spectral norm bounded by 1. That means applying it cannot blow up the signal. Stack twenty of these, stack a hundred, the signal stays bounded. That is what made stable training of ultra-deep models possible.

Try the Sinkhorn iters control. With 1 pass the matrix is still lopsided, with 3 or more it converges to well-balanced rows and columns. The paper runs 20 iterations. The upshot: a tiny extra cost between layers buys a huge amount of training stability. Next we look at another ingredient for stable training, the Muon optimizer.

STAGE 6 OF 9

Muon Optimizer



06 MUON OPTIMIZER

An optimizer is the rule that decides how each gradient becomes a weight update. AdamW, the usual choice, rescales each weight independently. Muon does something more elegant: it reshapes the whole gradient matrix before applying it.

On the left is the raw gradient matrix G. Its singular values, shown as that histogram, are uneven: some big, some tiny. Applied directly, the big ones dominate the update and the small ones get ignored.

The goal is an orthogonalized matrix where every singular value is 1. Normally that requires a singular value decomposition, which is expensive on a GPU. Muon uses a polynomial trick instead: the Newton-Schulz iteration.

Each Newton-Schulz pass applies a cubic polynomial in the matrix. One pass nudges the singular values toward 1. Several passes drive them right up to 1. Try the Iterations control and watch the histogram converge.

DeepSeek-V4 uses a hybrid schedule: 8 iterations with one set of coefficients for rapid convergence, then 2 more with a different set to lock the singular values precisely at 1. This gives both speed and accuracy.

Finally the orthogonalized matrix is rescaled and applied as the update. Muon tends to converge faster than AdamW on weight matrices, and it avoids some of the logit explosions that plague large-model training. Next, we leave training math behind and look at the engineering: a single fused GPU kernel that makes MoE fast.

STAGE 7 OF 9

MegaMoE Kernel

07 MEGAMOE KERNEL

In an MoE model, each token is routed to a handful of experts that may live on different GPUs. Sending tokens across GPUs takes time. If you do it naively, the GPUs sit idle while the network shuffles data. Try the Naive scheme option on the sidebar.

Previous work like Comet overlapped dispatch with the first linear layer, and combine with the second linear layer. That is a chunky two-stage overlap. Better than serial, but there is still dead time.

DeepSeek-V4 splits the experts into waves and pipelines five stages across them: dispatch, linear-1, activation, linear-2, combine. Switch the Scheme control to MegaMoE and watch the colored bars fill every tick.

While wave 1 is in linear-2, wave 2 is in activation, and wave 3 is being dispatched over the network. Every hardware unit, compute and interconnect, is busy almost all the time.

The theoretical speedup over the previous best is around 1.9x. In practice the open-sourced MegaMoE kernel delivers up to 1.96x on latency-sensitive workloads like RL rollouts. That is the difference between training a model in six months and nine.

MegaMoE is one kernel that fuses computation, communication, and memory access. No Python orchestration, no wasted ticks. Next, we look at what happened when this enormous model actually tried to train, and the two small tricks that kept it from falling over.

STAGE 8 OF 9

Training Stability

08 TRAINING STABILITY

At this scale, training is a fragile process. The team saw loss spikes that did not go away even after rolling back. The pattern was consistent: spikes appeared right after outlier activations in MoE layers, and the routing mechanism made them worse.

Flip the Stability control to Off to see what a spike looks like. A single outlier activation pushes the router to select bad experts, which produce worse activations, which push the router harder. It is a feedback loop that runs away.

The first fix is anticipatory routing. The idea is beautifully simple: compute the routing decision using slightly stale router weights from a few steps ago. The backbone updates freely at step t, but the routing indices are computed as if it were step t minus delta t.

That tiny decoupling breaks the feedback loop. The router cannot chase its own outliers because it is looking at older parameters. The team triggers this only when a spike is detected, then reverts to normal. The overhead is about 20% during the brief fix window, effectively zero overall.

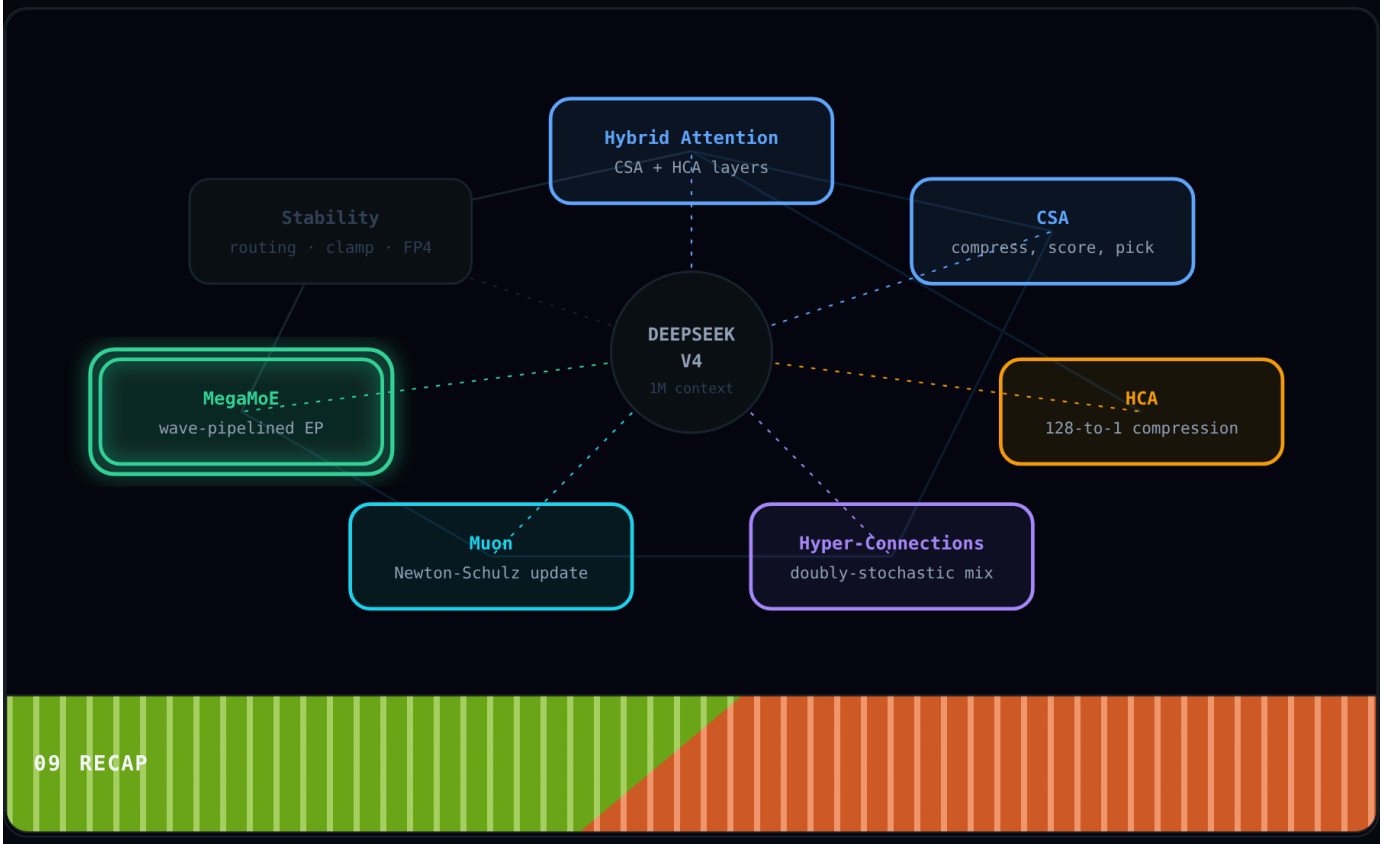
The second fix is a hard clamp on SwiGLU activations. The linear component is clamped to the range minus-ten to ten, and the gate is capped at ten. Outliers cannot exceed the clamp, so they cannot corrupt the downstream MoE layer.

The third trick is about deployment, not stability: FP4 quantization-aware training. During training the expert weights are quantized to 4 bits and dequantized on the fly. The model learns to tolerate the precision loss. At inference time the experts ship as real FP4, cutting memory by 2x.

Turn Stability back on and the loss curve stabilizes. Three small ideas, individually cheap, collectively the difference between a successful run and a failed one. Now let us zoom out and see how all seven pieces fit together.

STAGE 9 OF 9

Recap



09 RECAP

We started with the attention rewrite. Plain attention is quadratic, so a million-token context is impossible. CSA compresses by 4x and selects sparsely. HCA compresses by 128x and stays dense. Alternating them keeps cost manageable.

We zoomed into CSA: compress tokens 4-to-1, score them with a lightning indexer, keep only the top k. This is the detail-preserving path, with a sliding window for fresh local context.

Then HCA: compress tokens 128-to-1 and do plain dense attention on the tiny result. Cheap, blunt, effective for long-range signals that do not need fine detail.

Manifold-constrained hyper-connections fixed the residual stream. Four parallel channels mixed by a doubly stochastic matrix. Spectral norm bounded at 1. Deep stacks stop exploding.

The Muon optimizer replaced AdamW almost everywhere. Newton-Schulz iterations orthogonalize the gradient without an SVD. Convergence is faster and more stable.

MegaMoE was the systems breakthrough. Expert computation and cross-GPU communication interleaved in a single fused kernel, pipelined across waves. Close to 2x speedup over the previous best.

And three small tricks kept the whole thing stable: anticipatory routing broke the MoE feedback loop, SwiGLU clamping capped outliers, and FP4 quantization-aware training made deployment cheap.

Put it all together and you get a 1.6 trillion parameter MoE model that takes one million tokens in a single breath, trained successfully on 33 trillion tokens. None of these ideas is individually revolutionary. The revolution is that they fit together.