Episode Book

EPISODE #6501

Splitting Frontier AI Across Thousands of GPUs

PUBLISHED	8 November 2026 at 14:43 UTC
DURATION	30:44
LISTEN	myweirdprompts.com/episode/distributed-training-gpu-parallelism-58306/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	23 cited by the research (listed at the end)
BOOK GENERATED	9 November 2026 at 08:09 UTC

SYNOPSIS

Training frontier AI models is fundamentally a distributed plumbing challenge across thousands of accelerators. When weights, optimizer states, and activations vastly exceed single-chip memory, engineers must split workloads across data, tensor, pipeline, and expert parallelism. Balancing these techniques reveals hard trade-offs between network latency and memory bandwidth, showing why out-of-the-box software often hits only ten percent of peak theoretical efficiency while trying to prevent catastrophic cluster crashes.

TRANSCRIPT



Corn

Sixteen thousand GPUs, and the job still stops every three hours. That's the number that reframes everything, because training a frontier model isn't really a machine-learning problem at that point — it's a plumbing problem. A very expensive, very flaky plumbing problem.



Herman

And that's exactly the territory Daniel's prompt lives in this week. He's pointing at the fact that a frontier model doesn't fit on one GPU, or even one server, so training means slicing the work across thousands of accelerators that all have to march in lockstep. He names the whole menu — data parallelism, tensor, pipeline, sharded optimizers like ZeRO and FSDP, expert parallelism for mixture-of-experts — and he wants to know how each one actually splits the problem, how you stack them, where the network becomes the wall, and what goes wrong at that scale. Loss spikes, hardware dying mid-run, checkpointing and restarting a job that costs millions.



Corn

That's the part nobody draws on the architecture diagram.



Herman

It never makes the diagram. The plan, then: start with the arithmetic that makes all of this necessary, walk the axes of splitting one at a time, then spend the back half on what breaks and what it costs. Because the axes are the fun part, but the breaking is the story.



Corn

So let's start with the arithmetic. A dense four-hundred-five-billion-parameter model in bf16 needs about eight hundred ten gigabytes just to hold the weights. An H100 has eighty.



Herman

Eighty gigabytes of high-bandwidth memory, and this is just the weights — before gradients, before optimizer state, before the activations you need for the current batch. Weights alone put you more than ten times over the ceiling. And the other three pieces



Herman (continued)

are worse than people expect. The Adam family of optimizers keeps two extra numbers per parameter — a first moment and a second moment, essentially running averages of the gradient and its square — so optimizer state alone is typically several times the weight footprint. The model remembers its own momentum twice over.



Corn

So the weights are the small part.



Herman

On a per-device basis, often yes! The weights are the part you can shard most easily. Activations are the part that shows up uninvited every single step. There's a nice framing from a paper this January — Placement Semantics — that says every strategy is really a decision about where four training states live: parameters, optimizer state, gradients, activations. Replicated, sharded, offloaded, and so on. Pick your placements and the memory and communication costs fall out.



Corn

And every one of those placements buys memory by paying in communication.



Herman

That's the entire design space. There is no split that's free. You can move the bill around — from memory to network, from one collective pattern to another — but the bill exists. So the five axes, by name: data, tensor, pipeline, sharded optimizer, expert. There's a sixth, sequence or context parallelism, that's become important with long context. And the honest way to describe all of it is that this is a distributed-systems problem wearing a machine-learning costume. The math fits on a napkin. The collectives, the failure detection, the determinism — that's the hard part.



Corn

Which is my kind of problem, honestly. Slow, boring, and entirely about things not dying.



Herman

You would say that. Data parallelism first, then, because it's the one everyone meets first and the one that sets the baseline. You copy the whole model onto every worker — full weights, full optimizer state, the lot — and you feed each worker a different slice of the batch. Each one computes gradients on its own data, and then every step, all the workers all-reduce their gradients so everyone agrees on the update before moving on.



Corn

Notice what that bought us. Nothing, memory-wise.



Herman

Nothing! That's the misconception worth killing right there — people hear "we scaled to a thousand GPUs with data parallelism" and assume the model got bigger. It didn't. Data parallelism splits the data, not the model. Memory per device is unchanged. What you got was a bigger effective batch and more samples per second. Every step still pays a full gradient all-reduce across the whole fleet, which at sixteen thousand GPUs is its own engineering achievement, but a four-hundred-five-billion model does not become an eighty-gigabyte model because you added workers. You need something that splits the model itself.



Corn

Tensor parallelism is where that starts, and it's the most aggressive of the bunch.



Herman

You take an individual weight matrix and cut it up. Split the attention heads across devices, split the feed-forward layers column by column, so a single layer's multiplication is distributed — several chips each holding a slice of the same matrix, passing partial results back and forth mid-layer. It works beautifully. It's also the most demanding axis on the network, because that communication happens inside every layer, every step, latency-critical, with no way to batch it up nicely. So in practice, tensor parallelism lives inside a node, on the NVLink fabric, where bandwidth is enormous and latency is tiny. You almost never run tensor parallelism across nodes — the network can't keep up with how often it needs to talk.



Corn

So the placement rule falls out immediately. Tensor parallelism wants the fastest links in the building.



Herman

Right — TP stays intra-node, glued to the NVLink domain. Pipeline parallelism is the opposite temperament. Instead of cutting a layer into pieces, you cut the model into chunks of layers: stages. Device one holds layers one through eight, device two holds nine through sixteen, and so on, and you stream micro-batches through the stages like an assembly line. While stage three is chewing on micro-batch five, stage one is already starting micro-batch six.



Corn

The communication there is small. Just activations crossing the stage boundaries.



Herman

Small and infrequent, which is why pipeline parallelism tolerates slower inter-node links. But it has its own tax — the bubble. While the pipeline fills at the start and drains at the end, some stages are idle, and the deeper the pipeline, the bigger that idle fraction. So you tune the number of micro-batches to amortize the bubble away. It's a scheduling problem more than a networking problem, which makes it the friendliest of the axes and, I'd argue, the one where the intuition transfers from factory floors rather than supercomputing.



Corn

Then the sharded optimizers. ZeRO and FSDP.



Herman

This is the elegant one, and I say that knowing how you feel about the word elegant.



Corn

I feel that anything described as elegant in distributed systems costs you a weekend.



Herman

The weekend is available. ZeRO's insight is that data parallelism was wasting memory by replicating everything. Each rank holds the full optimizer state, the full parameters, even though all-reduce already proves they can cooperate. So instead you shard the optimizer state, gradients and parameters across the data-parallel ranks, and each rank gathers the parameters for one layer just-in-time, uses them, computes, scatters the gradients back, and throws the layer away. Layer by layer, the full model never lives anywhere at once on any single device. The January paper reproduces the original numbers exactly: ZeRO-3 uses eight times less memory than plain data parallelism, at one and a half times the communication cost.



Corn

Eight times the memory for fifty percent more talking. That's a trade I'd take every day of the week and twice on Sunday.



Herman

Most people would! And FSDP — fully sharded data parallelism — is the same idea wearing a PyTorch wrapper. You wrap your module, and the sharding, gathering and scattering happens under the hood. That's how it escaped the research ghetto: it became an API one line long, and everything that becomes an API one line long gets adopted by people who never read the paper.



Corn

Expert parallelism is the odd one out, because it only exists for mixture-of-experts models.



Herman

In a mixture-of-experts model, each token only visits a fraction of the parameters — the router picks a few experts out of the pool. Expert parallelism puts different experts on different devices, and the token travels to wherever its experts live. The dispatch and the combine are both all-to-all operations: every GPU potentially sending tokens to every other GPU. That's what makes trillion-parameter models tractable at all — you only



Herman (continued)

activate a slice — and it's also the most network-sensitive pattern of the five.



Corn

And then context parallelism.



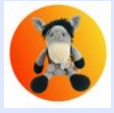
Herman

Splitting the sequence dimension. Matters more every quarter, because attention compute grows quadratically with sequence length, so a million-token context isn't a hundred times harder than ten thousand — it's the square of that relationship. The four-D-parallelism work treats data, tensor, pipeline and context as the four axes and gives memory formulas for each, so you can estimate your footprint before you commit a cluster to a config. Context parallelism splits the query dimension so each rank attends over its slice.



Corn

Each of those is a clean idea on its own. The real engineering is stacking them.



Herman

And that's where it gets ugly. Nobody runs one axis. A realistic frontier recipe is something like: tensor parallelism inside the node, because it needs NVLink; pipeline parallelism across nodes, because it barely talks; sharded data parallelism on top to widen the batch; expert parallelism wedged in if it's a mixture model; context parallelism for the long-context phases. Four or five simultaneous grids overlaid on the same physical cluster, each with its own communication rhythm, each constraining where the others can sit. The SuperMUC-NG recipe paper from May combines tensor, pipeline and sharded data parallelism to reach one hundred seventy-five billion parameters on Intel Max 1550 hardware — and its striking number is that the out-of-the-box software stack reaches about ten percent of theoretical peak bf16 FLOPs per tile. The phrase they use is that the complex interplay among these techniques determines GPU efficiency. Which is polite.



Corn

Ten percent of peak. The other ninety percent is scheduling, communication, and waiting for someone's bubble to drain.



Herman

A lot of it is. There's a lovely empirical anchor from the four-D-parallelism study: they ran four hundred fifty-four experiments on A100s and H100s, and found that when estimated memory usage stays below eighty percent of available GPU memory, training never hits an out-of-memory error. Which sounds obvious until you realize it means there's a usable safety margin you can actually target — eighty percent is the line where the config is safe and the last twenty percent is headroom for the jitters of real runs.



Corn

So the practical question for anyone configuring a cluster is how to choose the stack. And the answer, historically, is...



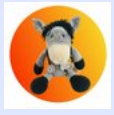
Herman

Trial and error, mostly. The Placement Semantics author says it outright — practitioners select parallelism strategies through trial and error because no unified systematic framework predicts their behavior. What that paper tries to do is give composition rules: proofs that two conditions, gradient integrity and state consistency, are necessary and sufficient for a distributed run to match what a single device would have computed. Which is quietly a big claim — it means you can check whether your combination is sound before you burn a week of cluster time finding out.



Corn

So the weights alone need eight hundred and ten gigabytes, and a single H100 has eighty.
Franklin: Sorry, this is going to sound thick, but I've been nodding along for months now and I've never actually asked. When you say the model doesn't fit on one GPU, is a GPU like the graphics card in my laptop, and do you just buy a bigger one, or plug in a few extra like a hard drive?



Herman

Same kind of chip, Franklin, much bigger sibling — and no, you can't buy a bigger one, because this is already the bigger one. Think of it like a library where the books are too many for any single shelf, so instead of a bigger shelf, you split the collection across a whole building of shelves and pay runners to carry pages between them constantly.



Corn

And the runners cost more than the shelves.



Corn

Which leads us to the network. The model is split. Now the network has to carry the split — and that's where the runtime goes.



Herman

This is the number I'd want every listener to carry around: the Occult work from ICML 2025 measured that in large-scale mixture-of-experts training, all-to-all communication consumes over forty percent of total runtime. Not ten percent, not a rounding error — forty. More than two-fifths of your sixteen-thousand-GPU cluster is, at any moment, waiting for tokens to arrive at their experts.



Corn

Forty percent of a supercomputer is an interconnect.



Herman

And if that's true, the frontier of model design quietly stopped being FLOPs. The interesting constraint is the shape of the network. Look at what the architecture papers are actually doing. FarSkip-Collective changes the model architecture — adds skip connections — specifically so that communication can be overlapped with computation, hidden behind useful work. They convert models from sixteen billion to a hundred and nine billion parameters that way, and the Llama 4 Scout variant stays within one percent of its instruction-tuned release after self-distillation. Nobody signed up for skip connections because they loved the loss curve. They signed up because the skip makes the network disappear.



Corn

So when people argue about attention variants as if it's a math debate, it's partly a networking debate in disguise.



Herman

Largely, at scale. And the wall isn't just volume — it's imbalance. With dynamic routing, a handful of experts get popular and overloaded, and those hot experts set the pace for the whole step. LAER-MoE calls it out directly: a few overloaded experts hinder overall iteration. Then RailWave, just this October, showed that even with routing and placement fixed, uneven utilization of the parallel network rails and incist — incast, sorry, many senders hitting one receiver — limits performance. They report up to five point eight four times speedup on H800s over the native approach, on GLM-4.5-Air-derived workloads.



Corn

Five point eight times. From traffic shaping.



Herman

From shaping traffic below the routing layer, no model change at all. Which tells you how much slack was sitting there. But here's the counterweight I actually want to spend a minute on, because it's rare: TerraceMoE, from August. They build a cost model for hierarchical two-hop expert dispatch — send to a local aggregator, then onward — and the two-hop route only wins if the fast fabric is roughly four times the bandwidth of the slow one. Their measured breakeven hierarchy ratio is three point nine eight. Then, in the paper, they disable their own step-level extrapolation, because a validation gate they set for themselves failed. They write that neither machine they measured reaches the hierarchical regime the model describes. These are ratio-only sensitivity results, not deployment predictions.



Corn

A paper that switched off its own headline claim because its own test failed it.



Herman

I want to frame that and hang it in the lab. In a field drowning in five-point-something-x speedup tables, somebody publishing "our extrapolation gate failed, here are the ratios anyway, do not treat them as predictions" is a public service. The whole speedup framing survives on benchmarks nobody gets to inspect, and this one paper demonstrating what self-gating looks like is worth a dozen of them.



Corn

Okay. Network's the wall, imbalance is the tax, papers exaggerate. Even if the network were free, though, the job still has to survive months of hardware.



Herman

And this is where Meta's Llama 3 paper is such a gift, because it's the most detailed public account of a frontier run's failure statistics. Fifty-four days of pre-training on sixteen thousand three hundred eighty-four H100s. Four hundred sixty-six job interruptions, of which four hundred nineteen were unexpected. Faulty GPUs caused a hundred forty-eight of them; HBM3 memory, seventy-two. GPU issues overall were nearly sixty percent of unexpected interruptions, and about seventy-eight percent were confirmed or suspected hardware. Run the arithmetic and you get a failure every three hours.



Corn

Every three hours. And the run still finished.



Herman

With over ninety percent effective training time, and significant manual intervention only three times in the whole run. Sit with that pair for a second — something breaks every three hours, and the cluster is still doing useful work ninety percent of the time. That gap is the restart path. It has to be fast, automated, and boring. And here's the honest part: nobody publishes how long a restart actually takes. Not the checkpoint size, not the write bandwidth, not the interval, not the reload time. The Llama 3 paper describes detection — the communication library times out on stalls, and NCCLX traces kernel and network activity and snapshots the failing collective's internal state, every rank, every pending transfer. But the restart itself is a black box. Four hundred nineteen unexpected interruptions in fifty-four days tells you the restart must be minutes, not hours — anything else and the goodput number couldn't hold. It's inference, though, not a published figure.



Corn

What's your guess, then? Honestly.



Herman

Minutes. A small multiple of minutes for a coordinated rollback across sixteen thousand ranks. But the word guess is doing the work in that sentence.



Corn

Then there's the failure mode I find unsettling. Stragglers.



Herman

Yes — because they don't fail. 's phrasing is that even a single straggler can slow down thousands of other GPUs, often appearing as functioning but slow communications. A chip that's degraded but alive looks exactly like a network problem. Every step, the whole fleet waits for the slowest rank, because lockstep means lockstep. 's approach was to rank suspect process groups — by investigating just a few top suspects, they could usually find the straggler. But notice what that assumes: that the slowdown is findable at all, that it shows up in your telemetry rather than hiding in the noise. My clinical instincts bristle at that — a system reporting itself fine is not evidence it's fine.



Corn

And Google got the scarier version of that problem.



Herman

Silent data corruption. The Gemini report describes it occurring at Ultra scale roughly once every one to two weeks — a bit flips somewhere in memory, network transmission, even CPU computation, and nothing crashes. The training continues, confidently, on wrong numbers. Their answer is the part I find profound: end-to-end determinism, plus proactive scanners running on idle machines and hot spares. If the same input bit-for-bit reproduces the same output, a corrupted result stops being invisible — recompute it, compare, and the disagreement is the corruption. Determinism turned silent wrongness into a findable event, and goodput went from eighty-five percent to ninety-seven percent.



Corn

Redundancy wouldn't have caught that. Two copies of a silent lie agree with each other.



Herman

Redundancy catches failures that announce themselves. Determinism catches the ones that don't. And at that scale you need it, because at Ultra-class cluster sizes the once-a-week bit flip is basically a scheduled event, not bad luck.



Corn

So the detection and restart machinery is carrying the run. Which takes us to checkpointing — the thing Daniel specifically flagged, and the thing I suspect people imagine as a quick save-file.



Herman

It's a global barrier. Every rank stops, the model's state is written out, and then everything resumes — and when tens of thousands of GPUs all pause at the same instant, the power draw swings. Says the instantaneous fluctuation across the data center can be on the order of tens of megawatts, stretching the limits of the power grid. Their words: this is an ongoing challenge as they scale to even larger models. A save-file that the electrical grid has opinions about.



Corn

The save button is a grid event.



Herman

So the interval is a genuine optimization problem, not a detail: checkpoint too rarely and every failure costs hours of lost work, checkpoint too often and you're paying the barrier plus the power spike plus the wear constantly. And I'll stay honest — the actual frontier numbers here are unpublished. Nobody outside these labs knows the size of a Llama-class checkpoint, the write bandwidth, the interval, the reload time. What we get is the goodput number and the failure count, and the difference between them is all the checkpoint engineering, hidden.



Corn

There's a small irony, though, that the checkpoints do double duty at the end of training.



Herman

They do! Llama 3 anneals the learning rate down to zero over the final forty million tokens, and during that annealing they compute the average of the checkpoints — Polyak averaging — and the average is the final pre-trained model. So the safety mechanism and the finishing technique are the same artifact. The thing you write every few minutes to survive a dead HBM module is also the thing you blend to produce the model the world actually downloads. I find that structurally lovely.



Corn

You'd find a checkpoint lovely.



Herman

It's a model averaging technique with a downtime justification! Everything at this scale is two things at once. Before we get to what else goes wrong — Corn, you've been circling the loss spike question since Daniel's prompt. Do you buy the framing that spikes are a central obstacle, or a solved annoyance?



Corn

I don't know, and that's the interesting tension. You've got SPAM measuring gradient spikes up to a thousand times larger than typical gradients, across multiple architectures and datasets. A thousand times. That's not jitter, that's a lightning strike in your optimizer.



Herman

And AdaGC's diagnosis is that the causes are a confluence — data outliers, hardware or transient computational faults, numerical precision, hyperparameters — and the damage mechanism is contamination: abnormal gradients poison both Adam moments, so even after the spike passes, the optimizer's momentum and variance estimates stay wrong until they're cleaned or reset. The spike is an event; the damage is a state.



Corn

Which explains why the fixes are so surgical. ZClip watches gradient norms with z-score anomaly detection and adapts the clipping threshold before the spike lands, rather than clipping everything to a fixed ceiling all the time.



Herman

SPAM resets momentum after a spike — throw out the contaminated state, keep training. And AdaGC clips per-tensor relative to a running average of that tensor's own historical clipped norms, reporting spike scores of zero on Llama-2 7B, Mixtral 8x1B and ERNIE 10B-A1.4B, with downstream accuracy actually improving over global clipping — by 1.32, 1.27 and 2.48 percent respectively. There's even a communication bonus: global norm clipping needs an extra all-reduce across all ranks to compute the global norm, so per-tensor clipping removes a collective from every step in hybrid-parallel setups.



Corn

Now put that whole literature next to 's own report on the 405B run — and I'm quoting — "we found this training recipe to be very stable: we observed few loss spikes and did not require interventions to correct for model training divergence."



Herman

A dozen papers building spike-aware optimizers, and the largest openly documented dense run says spikes barely happened.



Corn

Either the spike people are solving a problem the frontier labs already solved, or the frontier labs are leaving something out.



Herman

My read — and it's a read — is the batch schedule. Started that run at a batch of four million tokens and doubled it twice: to eight million after two hundred fifty-two million tokens seen, then to sixteen million after two point eight-seven trillion. That's a stability measure as much as a throughput one — huge batches from step one are exactly where the spike literature says trouble lives, and ramping the batch is a known smoother. Add data quality and their hardware teams catching faults early, and I think the honest reconciliation is that the mitigation literature is real and the recipe discipline is also real, and both get written up separately so they look like a contradiction. But it's unresolved,. Nobody has published a per-run spike frequency to settle it.



Corn

Unresolved, and nobody's spending a frontier run to settle it on purpose.



Herman

Not when a failed experiment costs more than the question.



Hilbert

Forty percent of runtime on the wires, and one slow chip dragging thousands — that matches my experience exactly. Corn, you said "no two chips are the same" and I've never heard anything truer said on this program. I ran a machine room for six years and I kept a handwritten log of which slot in which rack was the problem child, because I didn't trust the monitoring to tell me. The machine that ran hottest ran slowest, every time. The log was indexed by the sound the fans made, not by any sensor reading. I could walk the aisle and tell you which node was about to hold up the batch, by ear. I still keep a notebook of suspicious noises for this building. Never once been wrong about the lift.



Corn

You index a data center by ear.



Hilbert

The sensors lie. Fans don't. Fans only know one song and they sing it louder when something's wrong.



Herman

The fan curve as a straggler detector — that's better instrumentation than the process-group ranking describes, honestly. Though I notice the monitoring you didn't trust was presumably on the machines, and your notebook was on you.



Hilbert

The notebook never went down for maintenance. Took it home after they let the room go, kept the habit. The lift in this building has been making a two-note click since March and I have that on page one.



Corn

Page one.



Hilbert

Front of the book. Important things at the front. Learned that in the machine room and it's the only thing I took from it.



Herman

Needed a whole distributed tracing system to find what you found with a walk down an aisle.



Hilbert

Their aisle cost more. Mine smelled worse and I knew it better.



Corn

That gets us back to the thing nobody publishes — what a restart actually costs. That's where I'd leave the open question, because it's the number holding the whole story together. A failure every three hours and ninety-plus percent goodput — the difference is restart minutes we've never seen.



Herman

The goodput number is the only way we can even see the shape of it from outside. If you take one thing from today, take this: the frontier of model training isn't compute, it's the network and the failure path. When skip connections get added to a model so that communication can hide behind them, architecture itself has become a networking decision. The math is the easy part.



Corn

The hard part is that the job has to keep running through a piece of hardware dying every three hours, silently, with nobody noticing. That's the actual product.



Herman

As clusters grow, the interesting engineering moves from how do we split the model to how do we make the split survive without anyone watching.



Corn

On that — If this was your kind of episode, go back for episode eleven, How Does Fine Tuning Work Anyway; episode eighteen, Beyond the GPU; and episode three, Safetensors or something else. this has been My Weird Prompts, with me and Herman, produced as ever by Hilbert Flumingtop, who has never been wrong about the lift. If you got anything out of today's episode, leave us a review, it's the cheapest way to help the show. Find everything else at my weird prompts dot com. See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **Meta AI, The Llama 3 Herd of Models, arXiv:2407.21783v3, 2024-07-23 [primary]**
   (retrieved 2026-11-08)
   https://arxiv.org/pdf/2407.21783v3

2. **DataCenterDynamics, Meta report details hundreds of GPU and HBM3 related interruptions to Llama 3 training run, 2024-07-29**
   (retrieved 2026-11-08)
   https://www.datacenterdynamics.com/en/news/meta-report-details-hundreds-of-gpu-and-hbm3-related-interruptions-to-llama-3-training-run/

3. **Tom's Hardware, Faulty Nvidia H100 GPUs and HBM3 memory caused half of failures during Llama 3 training, 2024**
   (retrieved 2026-11-08)
   https://www.tomshardware.com/tech-industry/artificial-intelligence/faulty-nvidia-h100-gpus-and-hbm3-memory-caused-half-of-the-failures-during-llama-3-training-one-failure-every-three-hours-for-metas-16384-gpu-training-cluster

4. **DataCenterDynamics, Training Google's Gemini: TPUs, multiple data centers, and risks of cosmic rays, 2023-12-14**
   (retrieved 2026-11-08)
   https://www.datacenterdynamics.com/en/news/training-gemini-tpus-multiple-data-centers-and-risks-of-cosmic-rays/

5. **HackerNoon, Gemini**
   A Family of Highly Capable Multimodal Models: Training Infrastructure, 2023-12-24, (retrieved 2026-11-08)
   https://hackernoon.com/gemini-a-family-of-highly-capable-multimodal-models-training-infrastructure

6. **Glenn Klockwood, silent data corruption, 2025-04-04**
(retrieved 2026-11-08)
https://www.glennklockwood.com/garden/silent-data-corruption

7. **Mehta, Placement Semantics for Distributed Deep Learning, arXiv:2601.02311v1, 2026-01-05**
https://arxiv.org/abs/2601.02311v1

8. **Fujii, Watanabe, Yokota, Accelerating Large Language Model Training with 4D Parallelism and Memory Consumption Estimator, arXiv:2411.06465v1, 2024-11-10**
https://arxiv.org/abs/2411.06465v1

9. **Rajgopal, Solmsdorf, A Scalable Recipe on SuperMUC-NG Phase 2, arXiv:2605.07726v1, 2026-05-08**
https://arxiv.org/abs/2605.07726v1

10. **Luo et al., Occult: Optimizing Collaborative Communication across Experts, arXiv:2505.13345v1, ICML 2025**
https://arxiv.org/abs/2505.13345v1

11. **Liu et al., LAER-MoE: Load-Adaptive Expert Re-layout, arXiv:2602.11686v1, ASPLOS 2026**
https://arxiv.org/abs/2602.11686v1

12. **Goldman et al., NCCL EP: Towards a Unified Expert Parallel Communication API for NCCL, arXiv:2603.13606v3, 2026-04-02**
https://arxiv.org/abs/2603.13606v3

13. **Wang et al., RailWave: Adaptive Spatial and Temporal Scheduling for Expert-Parallel Communication, arXiv:2610.03415v1, 2026-10-02**
https://arxiv.org/abs/2610.03415v1

14. **Zeng et al., DisagMoE: Computation-Communication overlapped MoE Training, arXiv:2605.11005v1, 2026-05-10**
https://arxiv.org/abs/2605.11005v1

15. **Sun, Waleffe, Training Communication-Efficient Mixture-of-Experts Language Models with Layer Re-Configuration, arXiv:2608.28511v1, 2026-08-28**
https://arxiv.org/abs/2608.28511v1

16. **Xue et al., TerraceMoE: A Cost Model for Hierarchical MoE All-to-All Communication, arXiv:2608.27874v1, 2026-08-28**
https://arxiv.org/abs/2608.27874v1

17. **Dukler et al., FarSkip-Collective: Unhobbling Blocking Communication in Mixture of Experts Models, arXiv:2511.11505v3, MLSys'26**
https://arxiv.org/abs/2511.11505v3

18. **Wang et al., AdaGC: Enhancing LLM Pretraining Stability via Adaptive Gradient Clipping, arXiv:2502.11034v4, ICML 2026**
https://arxiv.org/abs/2502.11034v4

19. **Huang et al., SPAM: Spike-Aware Adam with Momentum Reset for Stable LLM Training, arXiv:2501.06842v2**
https://arxiv.org/abs/2501.06842v2

20. **Kumar et al., ZClip: Adaptive Spike Mitigation for LLM Pre-Training, arXiv:2504.02507v1, 2025-04-03**
https://arxiv.org/abs/2504.02507v1

21. **Ramasinghe et al., Taming Curvature: Architecture Warm-Up for Stable Transformer Training, arXiv:2606.16768v1, 2026-06-15**
https://arxiv.org/abs/2606.16768v1

22. **Khona et al., SOAP, Muon, and Beyond: Pushing LLM Pretraining Scales, arXiv:2607.20548v1, 2026-07-13**
https://arxiv.org/abs/2607.20548v1

23. **Xiao et al., Memory-Efficient LLM Training with Dynamic Sparsity, arXiv:2606.00888v1, ICML 2026**
https://arxiv.org/abs/2606.00888v1