PURETENSOR
INTELLIGENCE DISTILLED

A Rank-Local Host Table Beats Full Replication

Pinning 47 GiB of Huge Pages Stalled for 25 Minutes

Paper: PT-R-2026-052 v1.0

Author: PureTensor Ltd

Date: 13 September 2026

Status: Published

Abstract

DeepSeek-V4.1-Flash carries two very large static lookup tables (the publisher calls the mechanism Engram), 94.56 GiB each on disk, about 189 GiB together. Our two-node tensor-parallel pair cannot hold them on the GPUs, so they live in host RAM. This paper compares two ways of holding them there. In full replication, each node memory-maps and locks both complete table files although each GPU rank reads only its own quarter of the rows. In the rank-local layout, each rank holds only its own rows, 47.2 GiB per rank and 94.4 GiB per node, and the GPU gathers rows straight from pinned host memory with no CPU in the loop.

The memory result is clear. Full replication locked 189 GiB on the larger node and 182 GiB on the smaller one, which has about 256 GB of RAM; its available memory fell to a minimum of 24 GiB. The rank-local layout kept that minimum at 75 GiB. Boot took 384 s against 494 s (one boot per arm). Prefill throughput and median time to first token were better or equal in all 11 cells of our matrix. Decode was not so consistent: +2.8% over a ten-minute loop, between -2.3% and +13.0% across the cells, and many of those differences sit inside the spread between repetitions. Neither arm dropped a request.

The first attempt at the rank-local layout stalled for 25 minutes without an error. All four ranks logged one line about dropping the checkpoint page cache and then went silent; the launcher gave up at its 1,500 s limit. In one observation, the stall appears to lie in registering a 47 GiB transparent-huge-page region for pinning; allocating the buffer through the driver's own host allocator avoided it, and the layout then booted in 384 s. We state the memory results with confidence, the boot time as a single observation per arm, the throughput results with caution, and the root cause of the stall as unsettled.

1. A Table Too Large for the GPUs

The model is a 475 GiB checkpoint whose routed experts are stored at 4 bits. It carries two lookup tables, at layers 1 and 14, with 384,006,168 and 384,016,682 rows. Each row is 256 bytes of FP8 plus an 8-byte block scale, 264 bytes per row, which is 94.42 GiB of row data per table (the file on disk is 94.56 GiB) and about 189 GiB for the pair. A token touches 48 rows (24 per table: three n-gram orders times 8 heads), about 12.7 KB per token. The weights already take 72.6 GB per rank, so the tables cannot share a GPU with them.

We serve the model on two workstations, each with two 96 GB Blackwell GPUs on PCIe without NVLink, joined by a 200 GbE RDMA fabric. The configuration is tensor-parallel 4 and expert-parallel 4, one rank per GPU, two ranks per node, with the bundled speculative draft head enabled. One node has about 512 GB of RAM and the other about 256 GB. The smaller node is the constrained one throughout this paper, and every memory figure that matters is read from it.

Earlier work of ours moved the table from NVMe to RAM to end a hang in a host callback that read rows from disk; the cross-node setup was published on 4 September. The question here is what follows: once the table is in RAM, what is the right way to hold it? We expected the answer to concern memory, with throughput as a possible bonus.

One correction to our own notes belongs here. Our notes of 10 and 11 September put the RAM-mode table at about 95 GiB per node. That figure was wrong for the mode it described. Full replication in RAM mode locked 182 to 189 GiB per node (Section 3); about 95 GiB is the rank-local figure measured here.

2. Two Layouts, One Variable

Both arms used the same container image, the same engine arguments and the same seed. The two launch records differ in one field, the host-table mode; the driver environment also differs in the free-RAM gate (off in A, on in B), and arm B runs the patched host-table file described in Section 7.

Arm A, full replication, is the community adapter in its RAM mode. Each node memory-maps and locks up to both complete table files, about 189 GiB, although each rank reads only its quarter of the rows. Lookups run inside a CUDA host callback: the CPU copies the needed rows into a small pinned staging buffer (host memory the GPU can read directly by DMA), then a copy and a GPU gather kernel follow. The adapter has its own free-RAM safety check; our launcher does not run it, and it was bypassed in this arm as in earlier runs. We did not test the adapter under its own safety check, so this describes our launcher, not the adapter's default behaviour.

Arm B, rank-local shards, is the engine's own host-table feature in its private layout. One allocation per rank holds only that rank's rows of both tables. The GPU gathers rows directly over PCIe from pinned host memory, so there is no host callback, no staging copy and no CPU in the loop. Pinning is on by default.

Both arms ran the same battery script (the prompt manifests have different hashes; see Section 8). It had four parts: a 20-prompt greedy set at temperature 0; input lengths of 512, 8,192 and 32,768 tokens at 1, 4 and 8 concurrent streams with 1,024 output tokens each, two repetitions (18 runs per arm); 131,072-token inputs at 1 and 2 streams, one run each; and a ten-minute provoke loop of 8,192-token inputs at 8 streams, repeated back to back. The draft head was on, with at most 8 running requests and a context length of 409,600.

We took host-memory snapshots at the start, middle and end of each battery, and recorded one-minute host telemetry. Its figures below come from the comparison file generated from that telemetry, and the raw memory snapshots are retained.

3. Memory

Measure	A full replication	B rank shards
Host table per rank	whole files mapped by each rank	24,170 MiB + 24,171 MiB = 48,341 MiB (47.2 GiB)

Measure	A full replication	B rank shards
Host table per node	189 GiB locked (large node), 182 GiB (small node; the 7 GiB gap is unexplained)	94.4 GiB (two ranks)
Available memory, small node, minimum	24 GiB	75 GiB
Available memory, small node, snapshots (start, middle, end)	27.5 / 25.6 / 27.1 GiB	79.4 / 78.4 / 76.0 GiB
Available memory, large node, minimum	293 GiB	354 GiB
Swap used, small node, maximum	30.4 GiB	28.2 GiB
Swap-in rate, small node, mean	2,764 pages/s	1,865 pages/s

Table 1. Host memory in the two arms. Minima and swap come from one-minute telemetry; snapshots are the three retained meminfo captures per arm.

Full replication held roughly twice what the rank-local layout holds. On the smaller node, whose 256 GB also had to accommodate two ranks, their CUDA contexts and a 32 GiB shared-memory area, arm A left 24 to 28 GiB available. Arm B freed about 51 GiB there (minimum available 24 against 75 GiB).

Two cautions. First, the locked-memory counter reads 0 in arm B. Memory pinned through the driver's allocator is not counted as locked by the kernel, so a monitor watching only locked memory would report the better layout as using nothing. The available-memory series is the figure to trust. Second, the swap hazard was reduced, not removed. Both arms show 28 to 30 GiB of swap in use on the smaller node and non-zero swap-in throughout, which we attribute to residue and co-tenant pressure. Swap-in was about a third lower in arm B. An earlier summary of ours called the hazard removed; the data do not support that.

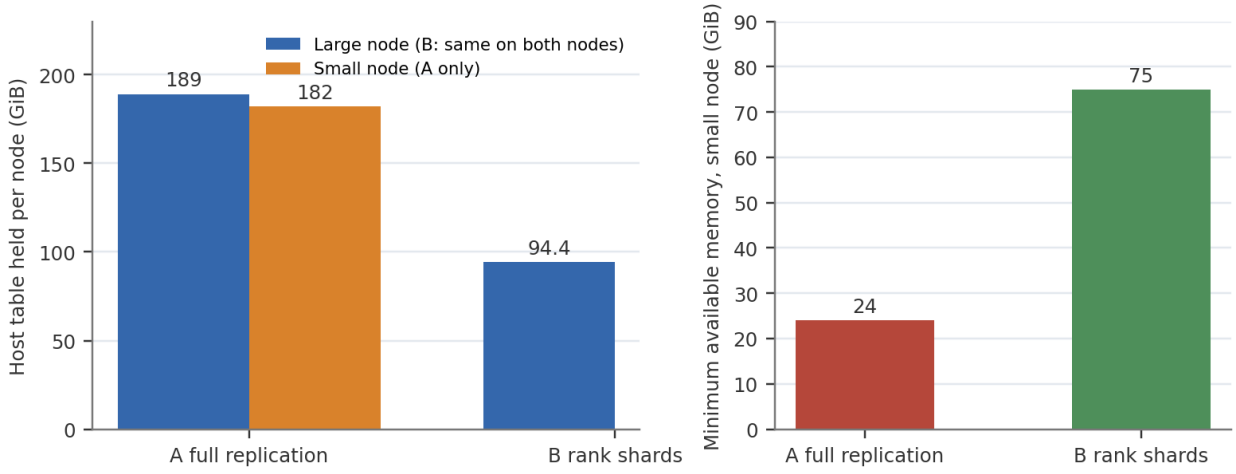


Figure 1. Left: host table held per node under each layout (B is the same on both nodes). Right: minimum available memory on the smaller node. Values are in Table 1.

4. Boot Time

Stage	A	B
Launch to healthy	494 s	384 s
Main-model weight load, large node (per rank)	209.6 s	74.6 s

Stage	A	B
Main-model weight load, small node (per rank)	335.4 to 335.9 s	220.6 to 221.1 s

Table 2. Boot stages in the two arms. One boot per arm. In arm B the weight-load stage includes copying the rank shard.

The 110 s launch-to-healthy difference matches the roughly 115 s difference in the smaller node's weight load: the slower node sets the pace. In arm B the weight-load stage includes copying the rank shard (70 s on the large node, 205 s on the small node), so the stage is not like for like. In arm B the checkpoint page cache is dropped before each rank copies only its own rows.

This is a single boot per arm, and the page-cache state before each launch was not controlled. We report the boot gap as consistent with the memory result and the load-stage timings, not as an independent finding.

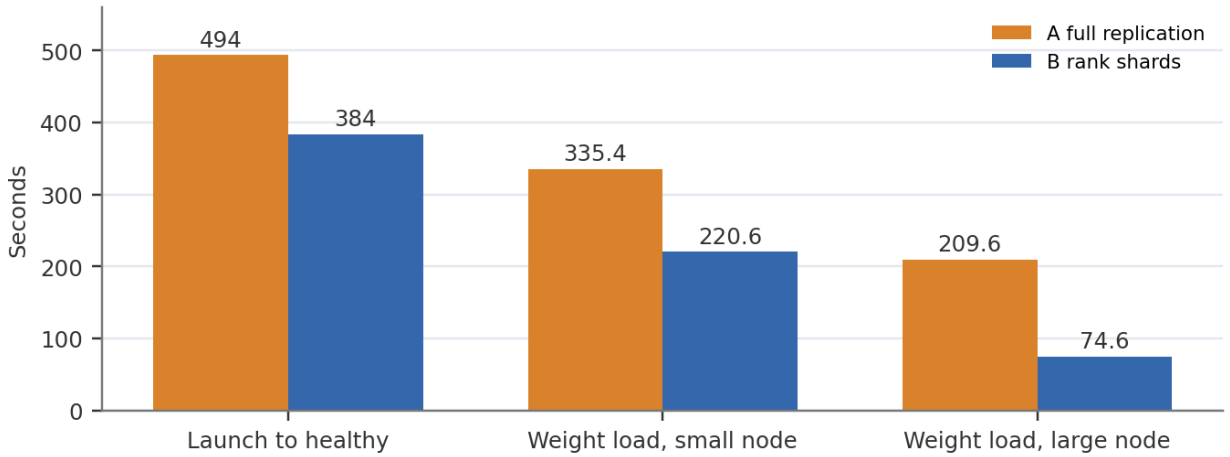


Figure 2. Launch-to-healthy time and per-rank main-model weight load on each node, one boot per arm. Values are in Table 2.

5. Throughput and Latency

Table 3 gives total decode tokens per second and the change of arm B against arm A. Each cell is a mean of two runs with 1,024 output tokens; the 131,072-token cells are one run each.

Input tokens	Streams	A decode	B decode	B vs A
512	1	149.8	152.4	+1.7%
512	4	465.8	460.9	-1.1%
512	8	663.1	658.8	-0.7%
8,192	1	151.8	156.2	+3.0%
8,192	4	447.0	478.8	+7.1%
8,192	8	656.8	671.3	+2.2%
32,768	1	152.0	148.5	-2.3%
32,768	4	458.3	468.2	+2.2%
32,768	8	653.3	682.0	+4.4%
131,072	1	133.4	150.7	+13.0%
131,072	2	272.8	272.8	0.0%

Table 3. Decode throughput (tokens per second, total across streams) by cell.

Input tokens	Streams	A prefill	B prefill	B vs A
512	1	2,996	3,245	+8.3%
512	4	5,276	5,607	+6.3%
512	8	6,036	6,329	+4.9%
8,192	1	7,166	7,459	+4.1%
8,192	4	7,408	7,628	+3.0%
8,192	8	7,327	7,626	+4.1%
32,768	1	7,414	7,636	+3.0%
32,768	4	7,424	7,648	+3.0%
32,768	8	7,479	7,677	+2.6%
131,072	1	7,061	7,236	+2.5%
131,072	2	7,161	7,316	+2.2%

Table 4. Effective prefill throughput (tokens per second) by cell.

Median time to first token was lower or equal in arm B in all 11 cells; for example 5.32 s against 5.12 s at 8,192 tokens and eight streams, and 27.63 s against 27.03 s at 131,072 tokens and two.

Prefill favours arm B everywhere, by 2.2% to 8.3%. Decode does not behave that way. Arm B decoded more than 2% faster in six of the 11 cells and within 2.3% of arm A in the other five, three of them slightly slower and one equal. The two repetitions inside a cell differ by up to about 7%: in the 8,192 by 8 cell the decode runs were 635 and 679 for arm A and 656 and 687 for arm B, and in the 32,768 by 8 cell 641 and 666 against 696 and 668. Many single-cell deltas therefore fall inside the spread, and only the pooled direction carries information. Because the prompt manifests differ between the arms, every cell delta also inherits that caveat.

The largest delta, +13.0% at 131,072 tokens and one stream, rests on one run per arm, and arm A's 133.4 sits well below its other single-stream cells (149.8 to 152.0). That is what a slow outlier in arm A would produce as much as a real effect. We do not rely on it.

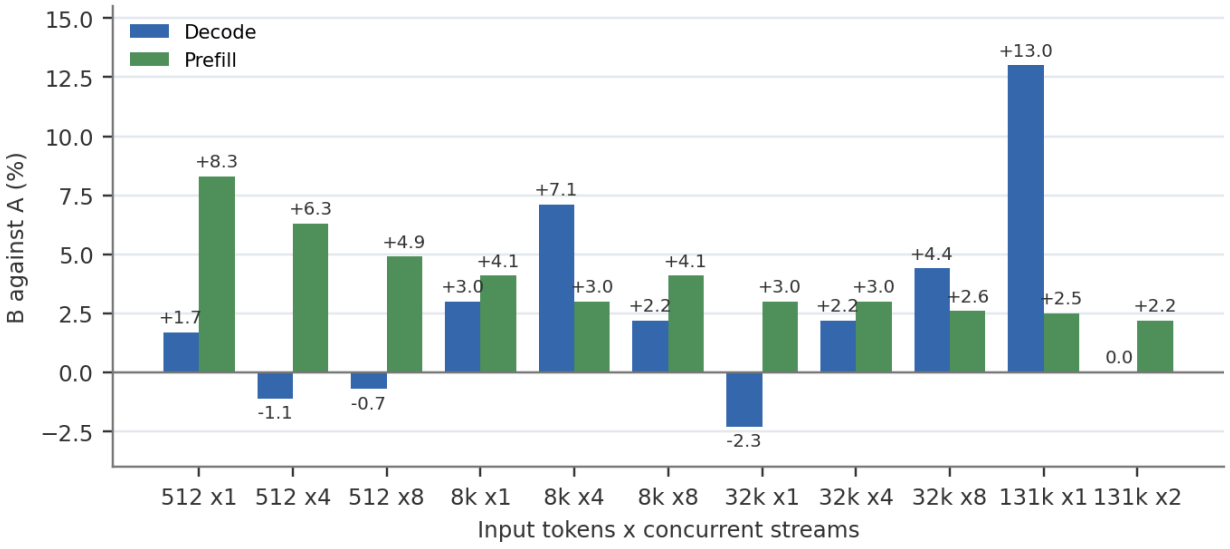


Figure 3. Change of rank shards against full replication, by cell, for decode and prefill. Values are in Tables 3 and 4.

6. The Ten-Minute Loop and Output Checks

The provoke loop repeats 8,192-token inputs at 8 streams back to back for ten minutes. We built it earlier to expose a hang.

Measure	A	B
Cases completed	28	29
Requests succeeded	224	232
Hangs or errors	0	0
Aggregate decode, mean (tokens per second)	661.7	680.1
Aggregate decode, median	657.4	679.9
Aggregate decode, minimum / maximum	618.6 / 705.0	645.0 / 706.2
Standard deviation	17.2	14.2
Prefill, mean (tokens per second)	7,354	7,509
Median TTFT, mean / maximum	5.34 / 5.94 s	5.23 / 5.46 s

Table 5. The ten-minute provoke loop at 8,192 input tokens and 8 streams.

Arm B is ahead by 2.8% on the mean. A Welch t statistic of about 4.4 on 28 against 29 cases is on the table, but the cases are consecutive within a single run per arm and so are not independent samples; we treat the statistic as indicative only.

Neither arm hung, which tells us little. The earlier NVMe-callback path hung once in 26 cases (25 clean, then a hang). If the true rate were 1 in 26, observing zero hangs in 28 cases would happen about 33% of the time, so a clean 28- or 29-case run is weak evidence either way.

On the 20-prompt greedy set at temperature 0, the two arms produced identical text on 7 of 20 prompts. The engine's agreement with itself on a rerun is about 6 of 20, because it is not deterministic across batch composition, so that difference is within the engine's own noise. Six factual-check prompts passed in both arms, and they were the same six.

7. The Failed First Attempt: A 25-Minute Stall in Registering 47 GiB

Our first attempt at arm B used the engine feature as shipped (the one stock-path attempt). The pair was relaunched at 12:30:19 UTC on 13 September. Each rank logged one line within about a minute saying that the checkpoint page cache had been dropped before pre-faulting its private shard. Then no engine progress lines (two ranks also logged a tokenizer warning). No weight loading began; in the successful run the same stage is followed within about 70 to 205 s by the weight-load lines. The launcher abandoned the launch at its 1,500 s readiness limit, 1,527 s after launch (12:55:46 UTC), a 25-minute wait. There was no error.

We then tried to reproduce the stall in isolation, in the same container, with a single registration call on a 47 GiB region of anonymous transparent-huge-page memory (the kernel's 2 MiB pages, which the pre-fault had produced), and saw the registration fail to complete in the few minutes we watched. We did not retain the output of that standalone run, only the boot logs from the real attempts, so the mechanism is a working hypothesis and we give no figures from it. The stock path was attempted once, and host memory fragmentation had not been controlled, after the preceding arm had run. What the retained logs do show is the 25-minute silence, and the success of the allocator change described below.

The fix is a small patch to the engine's host table. When pinning is on, it allocates the shard with the driver's host allocator and does not register an existing mapped region. The boot log says as much (no huge-page accounting on the driver's mapping). The first patched launch failed in 67 s with the error "Only dense CPU tensors can be pinned". The model loader builds the model under a CUDA default device, so the pinned buffer has to be created with an explicit CPU device. A second patched launch got through. In its logs the shard pages were allocated and loaded in 70 s on the larger node and 205 s on the smaller, and the boot completed in 384 s.

Whether this is an engine bug, a driver behaviour or a kernel behaviour is not settled here. The stall occurred on two different hosts, but we varied neither kernel nor driver deliberately.

8. What These Numbers Will Not Carry

Some claims survive: about half the host memory (94.4 GiB against 182 to 189 GiB per node), about 51 GiB more available on the smaller node, prefill and first-token latency better or equal in all 11 cells, and no failed requests. Boot at 384 s against 494 s survives only as a single-boot observation.

The decode claim is weak. The pooled +2.8% over the provoke loop is one run per arm over consecutive, non-independent cases. In the matrix, decode is better by more than 2% in six cells and within 2.3% in five. On 10 September, the same adapter configuration decoded 737 tokens per second as a mean over 92 cases of the same shape (diagnostics on, minimum 704, maximum 770); on 13 September it decoded 662 (minimum 619, maximum 705). That 10% day-to-day gap is larger than the gap between the layouts, and we could not explain it. The runs differ in date, prompt manifest and diagnostics setting. A decode gain of a few percent is smaller than the variation we cannot account for, so we do not claim one.

Other claims do not survive. The swap hazard is not removed, as Table 1 shows. The two arms did not demonstrably run identical prompts: the prompt-manifest hashes differ between the arms, our summaries described the batteries as identical, and we did not compare the contents. The comparison is also against the patched allocator, not against the engine's stock behaviour, because the unpinned host path was not measured. The root-cause reproduction is the weakest link, as Section 7 says; we claim only that the engine boot made no progress for 1,527 s and that, in an unretained standalone test, the registration had pinned nothing after about 6 minutes. Finally, there is one boot and one battery per arm, so there is no estimate of boot-to-boot variance.

Open questions: our internal agentic benchmark on the rank-local path; whether pinned driver memory stays compatible with a larger KV pool or longer contexts; and whether arm A would hang over many more cases.

9. What We Changed

Our production serving of this model now defaults to the rank-local host table. The full-replication adapter remains as a fallback. The pinned-allocation patch must be re-applied whenever the container image is rebuilt. The launcher gained a free-RAM gate for each mode and a check that refuses to start the upstream path when the patch is missing, because the fallback is the 25-minute silent stall. Our monitoring should not treat a zero locked-memory reading as an empty table; we read available memory instead.

10. Methods and Artefacts

The engine was SGLang, a development build from the DeepSeek-V4.1 branch, in a container pinned by digest, at tensor-parallel 4 and expert-parallel 4 with speculative decoding at block size 5 for the draft head and a seed of 0, on two nodes. The hardware is described in Section 1. The load driver was a small client that issues fixed-length prompts at a set concurrency from a hashed manifest of prompts and records total and per-request decode tokens per second, effective prefill tokens per second and median time to first token per run. Each matrix cell is a mean of two runs, or one run for 131,072 tokens. The Welch statistic in Section 6 is a descriptive figure only, given the dependence between consecutive cases. Zero requests failed in any run in either arm. Retained raw material: per-run records for every matrix and provoke case, engine logs for both nodes in each arm, launch records, the boot logs of the stalled attempt, the driver-log record of the second failed launch, the greedy outputs, the memory snapshots, the timestamped driver log and the patch. The standalone reproduction of the registration stall was not retained. The earlier hang evidence is retained as per-case logs.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-052. The measurements were taken on 13 September 2026 between 12:13 and 14:01 UTC, with the earlier hang evidence from 10 and 11 September; raw artefacts are retained.