



Decode Does Not See the Platform

Identical Blackwell GPUs on a Gen5/DDR5 and a Gen4/DDR4 host, and the upgrade we declined

Paper: PT-R-2026-033 v1.0

Author: PureTensor Ltd

Date: 3 October 2026

Status: Published

Abstract

We hold two compute nodes with the same GPUs: two RTX PRO 6000 Blackwell Max-Q cards each, at the same power cap, driver and clocks. They sit on different hosts. The newer host is a Zen 5 workstation platform with DDR5-4800 and PCIe Gen5. The older is a Zen 3 server platform with DDR4-2933 and PCIe Gen4. Moving the older node to a Gen5/DDR5 platform looked like the next natural step. The memory alone would be roughly \$16,000 of DDR5 at 2026 prices, and a new board and CPU would be needed on top. Before deciding, we ran the same serving and microbenchmark suite on both nodes at the same time, with model, checkpoint, container image, driver and tensor-parallel width held fixed.

Single-stream decode was a tie: 42.3 tok/s on the older Gen4 node against 41.7 tok/s on the Gen5 node. Under load the older node fell behind by 7 to 9 % in output throughput at 32 and 64 concurrent requests, by about 12 % in input throughput on a prefill-heavy run with 8,192-token prompts, and by 13 to 31 % in median time to first token. The steady decode step was level at every concurrency. The losses sit in the parts of the run shaped by prefill (processing the prompt), not by decode (generating tokens one at a time).

The microbenchmarks show where the platform does differ. Pinned host-to-GPU copies run at 28 against 57 GB/s, half, as the link generation predicts. Cold start from page cache takes 172 against 102 s. An eager 8 KiB NCCL all-reduce takes about 50 μ s on the older node against 8.9 μ s on the newer one, which first looked like a misconfiguration. Captured inside a CUDA graph, the older node does the same all-reduce in 12.6 μ s. The eager penalty is host-side per-call overhead. Serving runs decode inside CUDA graphs, so it never pays that penalty.

We declined the upgrade. Every serving figure is a single run, and the CPU, DRAM and PCIe generation all differ together, so this paper measures a platform rather than any one component. A planned CPU-offload stage crashed identically on both nodes because of a software defect and produced no number.

1. The Question

Our GPU fleet is three compute nodes, each with two RTX PRO 6000 Blackwell GPUs, joined by a 200 GbE RDMA fabric. We prefer three separate nodes to one large chassis. The nodes can run as three independent tensor-parallel-2 seats (a model split across two GPUs), as a two-node tensor-parallel-4 group, or all together.



Two of the three nodes carry the same GPU SKU on different hosts. The question was whether to bring the older one up to the newer platform. No Gen5 server platform accepts DDR4, so the upgrade means a new motherboard, CPU and memory. We priced only the memory: 8 x 64 GB of DDR5 RDIMM, roughly \$16,000 at 2026 prices. The board and CPU were not priced.

A smaller DDR5 build would not have worked. At the time of measurement the older node had 335 GiB of its 503 GiB in use, 286 GiB of it shared memory holding our resident model's host-RAM layer. Two DIMMs would also populate only two memory channels. Without data, the decision would have rested on cost alone. We ran the A/B to decide on measurements.

2. Two Hosts, One GPU Configuration

	Gen5 node	Gen4 node
CPU	AMD Threadripper PRO 9975WX (Zen 5)	AMD EPYC 7713 (Zen 3)
Cores / threads	32 / 64	64 / 128
L3 cache	128 MiB	256 MiB
Sockets / NUMA nodes	1 / 1	1 / 1
Memory	8 x 64 GB DDR5-4800 registered ECC (512 GB)	8 x 64 GB DDR4-2933 LRDIMM (512 GB)
GPU links	PCIe Gen5 x16, both GPUs	PCIe Gen4 x16, both GPUs
Nominal link rate, per direction	about 63.0 GB/s	about 31.5 GB/s
Nominal 8-channel DRAM	307.2 GB/s	187.7 GB/s

The nominal figures are computed from specification rates, not measured. On paper the older node has half the link bandwidth and a little over half the DRAM bandwidth. It has twice the cores, on an older microarchitecture.

The GPU side was identical. Both nodes ran 2 x RTX PRO 6000 Blackwell Max-Q at a 250 W power limit, maximum SM clock 3090 MHz, maximum memory clock 14001 MHz, NVIDIA driver 615.71.09 and x16 link width. The driver's management tool reported link generation 5 on one node and 4 on the other in each run. In both nodes the two GPUs connect through the CPU's host bridge on a single NUMA node, so the topology is the same in kind.

3. Method

Both nodes used the same container image build, with identical build timestamps. It contains PyTorch 2.13.0+cu130 (CUDA 13.0), NCCL 2.30.7 at runtime, and a development build of SGLang from source (version string 0.0.0.dev0). PyTorch reports that it was compiled against NCCL 2.29.7, while NCCL logs itself as 2.30.7+cuda13.3. The two nodes use different container runtimes: a standalone container on the Gen5 node and an orchestrated pod on the Gen4 node. They therefore report different image IDs. Both nodes' crash tracebacks (Section 9) cite the same SGLang source line numbers, which confirms the code was the same. Both containers ran privileged, with host IPC and host networking.

To free the GPUs we paused our resident model seat under a GPU reservation; the pause took 4.5 s. The paused process kept about 4.8 GB on each card on both nodes (4,884 and 4,804 MiB on the Gen5 node, 4,880 and 4,802 MiB on the Gen4 node). It also kept its pinned host-memory weight backup. Available host memory at the start was 198.0 GiB on the Gen5 node and 167.9 GiB on the Gen4 node.

The model was Qwen3.8-27B in BF16. This is a community derivative with modified weights and the unmodified Qwen3.8-27B architecture: 18 safetensors shards, 55.6 GB across 63 files, with



a manifest-verified copy on each node. The architecture is a hybrid. Gated delta-net linear-attention layers, which keep a fixed-size recurrent state instead of a growing key-value cache, are mixed with full-attention layers. There are 64 layers, hidden size 5120, 24 attention heads and 4 KV heads. Native context is 262,144 tokens; we served it at 16,384.

The SGLang server ran at tensor-parallel 2 with static memory fraction 0.80 and context length 16,384. The radix (prefix) cache was off. Chunked prefill was set to 8,192 tokens, meaning long prompts are processed in pieces of at most that size, and maximum prefill tokens to 16,384. Other settings were FCFS scheduling, FlashInfer attention and sampling, the Triton linear-attention backend, the overlap scheduler, and SGLang's custom all-reduce. CUDA graphs were on: pre-recorded GPU work sequences that are replayed without the CPU launching each kernel. These covered decode batch sizes 1 to 256 and "breakable" prefill graphs for 4 to 8,192 tokens. There was no speculative decoding and no torch.compile.

Per GPU, weights took 25.64 GB and the KV cache 24.06 GB, enough for 788,301 tokens on the Gen5 node and 789,122 on the Gen4 node. The recurrent-state cache took 21.16 GB of SSM state plus 0.41 GB of convolution state for 300 slots, which caps running requests at 300 on both.

Before each server start we read the full safetensors into the page cache. This took about 1 s on the Gen5 node and about 4 s on the Gen4 node, so the files were already cached. "Cold start" below means server process launch to first healthy response, with weights in the page cache.

Load came from SGLang's serving benchmark. Prompts were random token IDs at fixed lengths, request rate unlimited, maximum concurrency c, seed 1234, 2 warm-up requests, temperature 0, streaming. End-of-sequence was ignored so that every request generated exactly its output length. The decode sweep used 1,024 input and 256 output tokens, with 16 prompts at c=1, 32 at c=8, 128 at c=32 and 256 at c=64. The prefill-heavy run used 8,192 input and 64 output tokens, 32 prompts, at c=8. Each configuration ran once per node.

The two nodes ran the suite simultaneously on 29 September 2026. Microbenchmarks began at 20:26:43 (Gen5) and 20:26:45 (Gen4) UTC. Serving ran from 20:28:37 to 20:34:45 on the Gen5 node and from 20:30:00 to 20:36:56 on the Gen4 node. Collective-latency diagnostics followed from 20:35:31 to 20:50:31, with one run per variant.

The microbenchmarks ran as a two-rank PyTorch job inside the same image. They covered:

- pinned and pageable host-to-device (H2D) and device-to-host (D2H) copies of 1 GiB per GPU, first one GPU at a time and then both at once (pinned memory is page-locked host memory that the GPU can read by direct memory access);
- a 1 GiB peer copy between the GPUs in both directions, and 4 KiB peer-copy latency over 2,000 iterations;
- NCCL all-reduce in BF16 from 8 KiB to 256 MiB, with 500 iterations up to 1 MiB and 50 above (an all-reduce sums a buffer across GPUs and leaves the result on each);
- a crude host-memory copy: a 4 GiB float32 tensor, 16 threads, 5 passes, counting read plus write bytes;
- single-kernel launch cost over 20,000 tiny kernels, launched eagerly and then replayed from a CUDA graph;
- an 8 KiB all-reduce timed eagerly over 500 iterations and inside a captured graph (50 all-reduces per graph, 20 replays).

4. Serving Results

Run	Gen5 node	Gen4 node	Gen4 vs Gen5
c=1, output tok/s	41.68	42.28	+1.4 %



Run	Gen5 node	Gen4 node	Gen4 vs Gen5
c=8, output tok/s	258.16	255.98	-0.8 %
c=32, output tok/s	612.96	568.52	-7.3 %
c=64, output tok/s	801.26	733.23	-8.5 %
Prefill run (8,192 in, c=8), input tok/s	5,015.60	4,435.64	-11.6 %
Prefill run, output tok/s	39.18	34.65	-11.6 %

At c=1 and c=8 the two nodes are level. The older node is nominally 1.4 % faster at c=1, which we treat as run-to-run noise, although we ran no repeats to measure that noise. The gap opens at c=32 and c=64 and is largest on the prefill-heavy run.

Time to first token (TTFT) is the delay from sending a request to receiving its first output token. Inter-token latency (ITL) is the gap between successive streamed tokens. Time per output token (TPOT) is a request's decode time divided by its output tokens.

Run	Median TTFT (ms), Gen5 / Gen4	Mean TTFT (ms), Gen5 / Gen4	P99 TTFT (ms), Gen5 / Gen4
c=1	233.48 / 270.44 (+15.8 %)	231.27 / 269.22	238.81 / 275.31
c=8	1,442.78 / 1,624.14 (+12.6 %)	1,436.49 / 1,439.47 (+0.2 %)	1,684.53 / 1,776.17
c=32	3,828.49 / 5,014.75 (+31.0 %)	3,681.83 / 4,471.15 (+21.4 %)	5,665.36 / 7,105.17
c=64	6,630.55 / 7,865.02 (+18.6 %)	6,555.48 / 7,606.27 (+16.0 %)	11,178.25 / 12,923.20
Prefill run	7,386.69 / 8,539.74 (+15.6 %)	7,282.36 / 8,404.83	11,507.19 / 13,333.20

At concurrency above 1, TTFT includes waiting for the chunked prefill of the other requests admitted alongside. The older node's median TTFT is higher in every run. The c=8 row needs care. The median is 12.6 % higher but the mean only 0.2 % higher, and the TTFT standard deviation is 380.7 ms against 116.3 ms. At c=8 the older node has a wider distribution, not a slower one on average.

Run	Median ITL (ms), Gen5 / Gen4	Max ITL (ms), Gen5 / Gen4	Mean TPOT (ms), Gen5 / Gen4
c=1	23.23 / 22.81	26.77 / 24.07	23.17 / 22.68
c=8	25.21 / 24.80	148.05 / 876.42	25.45 / 25.69
c=32	30.43 / 30.45	5,177.04 / 6,037.91	37.93 / 38.92
c=64	36.72 / 37.21	10,651.50 / 12,460.63	54.43 / 57.73
Prefill run	26.70 / 26.17	9,772.19 / 11,217.73	91.68 / 100.95

This table locates the throughput loss. Median ITL, which approximates a steady decode step, agrees to within 0.1 to 1.3 % at c=32 and c=64, and is marginally lower on the older node at c=1, c=8 and in the prefill run. The maximum ITL differs by much more. A maximum ITL is a decode step that waited behind a prefill chunk. Mean TPOT folds those stalls into the per-token average, and it rises on the older node as concurrency rises. End-to-end median latency follows the same pattern: +6.6 % at c=32, +9.3 % at c=64 and +12.9 % on the prefill run.

Retokenised output counts were identical at c=1 (4,040) and c=8 (8,130), and within 0.1 % at c=32 (31,648 against 31,653) and c=64 (63,107 against 63,060). This is consistent with the two nodes computing the same greedy outputs, so the throughput comparison is like for like.



5. Cold Start

Stage	Gen5 node	Gen4 node
Total, launch to healthy	102.5 s	172.1 s (1.68x)
Launch to configuration logged (Python start-up, imports)	about 12.5 s	about 25.1 s
Distributed init	0.77 s	1.39 s
Weight load (rank 0)	8.06 s	16.73 s (2.08x)
Prefill CUDA-graph capture (58 sizes)	51.94 s	79.52 s (1.53x)
Decode CUDA-graph capture (36 batch sizes)	5.07 s	10.78 s (2.13x)
HTTP up to "ready" (server warm-up)	about 7 s	about 16 s

Stage times come from server log timestamps, at 1-second resolution except where the log reports elapsed seconds. Weight loading from page cache is a host-to-GPU copy, and it runs at about half speed on the older node, as its link does. Graph capture and Python start-up are host-CPU work, and they are slower on the Zen 3 host despite its larger core count. Prefill graph capture is the largest single stage on both nodes.

6. Microbenchmarks

Measure	Gen5 node	Gen4 node	Ratio
H2D pinned, GPU0 / GPU1 (GB/s)	57.02 / 57.03	27.94 / 27.96	2.04x
D2H pinned, GPU0 / GPU1 (GB/s)	56.50 / 56.51	28.12 / 28.29	2.0x
H2D pageable, GPU0 / GPU1 (GB/s)	44.76 / 45.11	22.41 / 22.34	2.0x
H2D pinned, both GPUs at once, each (GB/s)	57.03 / 57.03	27.94 / 27.92	2.04x
Peer copy GPU0 to GPU1 / GPU1 to GPU0 (GB/s)	52.08 / 52.04	25.79 / 25.52	2.02x
Peer copy 4 KiB latency (μs)	7.20	12.37	1.72x slower
Kernel launch, eager, per kernel (μs)	2.64	5.87	2.22x slower
Kernel inside CUDA-graph replay, per kernel (μs)	0.94	0.92	level
Crude host-memory copy, 16 threads (GB/s)	205.7	50.9	4.04x

Pinned H2D reaches about 90.5 % of the nominal per-direction link rate on the Gen5 node and 88.7 % on the Gen4 node. With both GPUs copying at once, neither node loses anything. For two GPUs, then, the root complex and host memory are not the limit; each GPU's own link is. Peer copies between the GPUs also run at half speed on the older node.

Two rows separate host CPU cost from GPU cost. An eager kernel launch costs 2.22 times as much on the Zen 3 host, while a kernel replayed from a CUDA graph costs the same on both. The host-memory copy figure is a crude stand-in, not STREAM. Its 4x ratio is far larger than the 1.64x nominal DRAM ratio, and we do not offer it as a DRAM bandwidth result.



Message	Gen5 latency (μ s)	Gen4 latency (μ s)	Gen5 bus bw (GB/s)	Gen4 bus bw (GB/s)	Gen4 slower by
8 KiB	8.9	49.1	0.92	0.17	5.53x
16 KiB	8.8	48.6	1.86	0.34	5.52x
64 KiB	8.9	56.5	7.37	1.16	6.35x
256 KiB	18.4	45.9	14.21	5.71	2.49x
1 MiB	44.8	67.8	23.42	15.46	1.51x
16 MiB	486.2	936.8	34.51	17.91	1.93x
256 MiB	7,286.9	13,574.1	36.84	19.78	1.86x

These are eager NCCL all-reduces in BF16 across two ranks, as reported by rank 0; rank 1 was within 0.4 %. For two ranks, bus bandwidth is message size divided by time. Large messages are link-bound, and the older node gets about half. Small messages on the older node sit on a flat floor near 50 μ s. The 256 KiB call (45.9 μ s) is faster than the 8 KiB call (49.1 μ s), so the floor is fixed overhead, not data movement.

7. The Small All-Reduce That Serving Never Pays

A 5.5x penalty on an 8 KiB all-reduce looked like a configuration fault. Our first suspicion was that NCCL was routing traffic through host memory. It was not. NCCL's transport trace shows peer-to-peer transfer over CUDA memory for both channels on both nodes, and peer access is reported as available on both.

By default NCCL uses its ring algorithm with the low-latency (LL) protocol at 8 KiB. We forced alternatives on the older node, one run each. The default measured 51.0 and 52.8 μ s, forced LL128 (which selected tree) 53.6 μ s, and forced Simple (also tree) 52.7 μ s. With peer-to-peer disabled, so that traffic went through shared host memory, it measured 50.2 μ s; peer-to-peer read mode gave 49.7 μ s. None moved the floor materially.

We then looked at host power management. The older node runs the acpi-cpufreq driver with the "performance" governor and boost on, and its deepest idle state, C2, has a 30 μ s exit latency. The newer node runs amd-pstate-epp with energy-performance preference "performance", and a C2 exit latency of 100 μ s. Disabling C2 on the older node did not help: 53.0 μ s with C2 disabled against 45.6 μ s with it enabled, in the same harness. We re-enabled C2 immediately afterwards. The older node also boots with the IOMMU (the device address-translation unit) in passthrough mode. More of its PCIe bridges have access control services enabled, a feature that can force peer traffic up through the root complex: 24 bridges against 11. We isolated neither setting.

The decisive test ran the same harness on both nodes, one run each.

8 KiB all-reduce	Gen5 node	Gen4 node
Eager (micro suite)	8.9 μ s	49.1 μ s
Eager (graph-test harness)	13.2 μ s	45.6 μ s (C2 on), 53.0 μ s (C2 off)
Inside a captured CUDA graph	18.2 μ s	12.6 μ s (C2 on and off)

When the all-reduce is launched from a CUDA graph, the host CPU drops out of the per-call path. The older node then does 12.6 μ s, 3.6 to 4.2 times better than its eager figures. We read the eager gap as host-side per-call overhead in PyTorch's NCCL process-group path on the Zen 3 host. That reading is consistent with eager kernel launch costing 2.22 times as much there. SGLang runs decode inside CUDA graphs, and the scheduler logs "cuda graph: True" on decode batches. The overhead therefore does not reach serving, which matches the tie at $c=1$.



The newer node's in-graph result is not explained. It measured 18.2 μ s, higher than its own eager figure of 13.2 μ s in the same harness and higher than the older node's 12.6 μ s. That process then hung at teardown and was killed after about 8 minutes, after the measurement had already printed. We changed the older node's harness to exit without the teardown call. On a single run we do not claim that graphs make the newer node slower. We claim only that it was not faster inside a graph. Across the session the older node's nine eager 8 KiB measurements ranged from 45.6 to 53.6 μ s. The newer node's two were 8.9 and 13.2 μ s.

8. Where the Prefill Gap Comes From

This section is our own arithmetic, not a measurement. With tensor-parallel 2, each layer's output is summed across the two GPUs. We assume two all-reduces per layer over 64 layers, 128 per forward pass, each carrying tokens $\times 5120 \times 2$ bytes.

A 1,024-token prompt then gives about 10.5 MB per all-reduce. At the measured 16 MiB bus bandwidths (34.51 and 17.91 GB/s) that is about 304 against 586 μ s per call, about 36 ms more per prefill on the older node. The measured difference in $c=1$ median TTFT is 36.96 ms.

An 8,192-token chunk gives about 83.9 MB per all-reduce. At the 256 MiB bus bandwidths that is about 2.28 against 4.24 ms, or about 251 ms more per chunk. The prefill run is 32 such chunks, about 8.0 s in total; the measured duration difference is 6.83 s. The two are of the same order, with the estimate high.

The estimate has known gaps. SGLang may use its own custom all-reduce rather than NCCL at some sizes, and we ignore overlap and host scheduling. It supports, but does not prove, the view that the prefill and TTFT gap is mostly the halved GPU-to-GPU link bandwidth. On the same view, decode steps carry about 10 KiB per request per all-reduce, inside CUDA graphs, and are too small to feel the link.

9. CPU Offload: No Number

We planned a stage with 20 GB of weights offloaded to host memory, at $c=1$ and $c=8$ with 4c prompts, 1,024 input and 32 output tokens. This is the regime where host memory and link bandwidth should matter most. It crashed identically on both nodes during weight loading, before any request was served (at 20:35:15 on one node and 20:37:56 on the other):

```
"RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu!"
```

The error is raised in the model's normalisation-layer weight loader, which adds 1.0 to an offloaded CPU parameter into a GPU buffer. This is a software defect in that SGLang build for this architecture, independent of platform. The only evidence we have for offload-shaped traffic is the copy bandwidth: 57 against 28 GB/s.

10. Relation to PT-R-2026-021

PT-R-2026-021, measured on 10 September 2026, asked how many nodes a seat should span. One arm ran a 204.5 GB NVFP4 GLM-5.3-Flash at tensor-parallel 4 across two nodes over the RDMA fabric. The other ran a different model, Qwen3.8-Flash-Next, at tensor-parallel 2 on one node. The one-node seat won on every metric except single-stream decode; at about 31,000 tokens of context, for example, prefill ran at 7,412 against 4,711 tok/s. Because the model changed between arms, that paper could not say what the host contributes.

Here the GPUs, model, checkpoint, image, driver, power cap and tensor-parallel width are fixed, and only the host platform changes, within one node. The comparison is narrower but cleaner,



and it is tied to a specific purchase. It also adds the eager-versus-graph collective finding.

11. What These Numbers Will Not Carry

Every serving and diagnostic figure comes from a single run. There are no confidence intervals, and we have no measured estimate of run-to-run noise. Differences under about 2 %, which covers $c=1$ and $c=8$ throughput, should be read as ties. The serving runs used 16, 32, 128, 256 and 32 requests. The $c=1$ run in particular rests on 16 requests.

The comparison is of platforms, not components. CPU microarchitecture (Zen 5 against Zen 3), core count (32 against 64), DRAM (DDR5-4800 against DDR4-2933) and PCIe generation all differ. So do the CPU power-management drivers, the IOMMU and access-control settings, and the container runtime. We cannot apportion any result among them. Attributing the prefill gap to link bandwidth rests on the arithmetic in Section 8, which agrees in order of magnitude but is not a controlled test. A cleaner split would force the newer node's GPU slots to Gen4 and re-run; we did not do this.

The workload was synthetic: random-token prompts at fixed lengths, forced output length, prefix cache off. Real traffic with shared prefixes would cut prefill work and should shrink the prefill-side gap. We tested one model at one tensor-parallel width. Larger or mixture-of-experts models, longer contexts, KV-cache offload, and the two-node tensor-parallel-4 mode, in which the older node would pace the newer one, are all untested. Offload serving produced no number. During every run, the paused resident model held about 4.8 GB per GPU and pinned host memory on both nodes. The host-memory copy figure is not a DRAM bandwidth measurement.

After the serving runs, a watchdog restarted the paused resident model before the reservation ended. It overlapped no serving measurement. All values here come from raw benchmark logs, JSON and session output, not from earlier summaries, some of which had rounded or omitted figures.

Within those limits, three claims survive. First, single-stream and low-concurrency decode, and the steady decode step at every concurrency tested, are level across the two platforms. Second, the older node loses 7 to 9 % throughput at 32 to 64 concurrent requests, about 12 % on long-prompt prefill, and has higher median TTFT. Third, its host-to-GPU and GPU-to-GPU bandwidth is half, and its cold start is 1.68 times as long.

The eager small-message all-reduce penalty is real on that host, but it does not reach graph-captured decode. The claims that do not survive are any per-component attribution, any statement about offload, and any account of the newer node's 18.2 μ s in-graph result. The 4x host-memory ratio also has no standing as a bandwidth result.

12. What We Changed

Nothing. For roughly \$16,000 of DDR5, plus an unpriced board and CPU, the upgrade would buy no single-stream decode gain. It would buy 7 to 9 % more throughput at 32 to 64 concurrent requests and about 13 % more prefill throughput (the newer node's prefill run is 13.1 % faster). It would also buy lower TTFT, twice the host-to-GPU and GPU-to-GPU bandwidth, and a cold start 69.6 s shorter. The newer platform earns its keep on bulk data movement, long-prompt prefill, saturated batches and host-CPU-bound work. Our current workloads are not dominated by any of these. The upgrade is held. We will revisit it if DRAM prices fall, or at the next GPU generation, when a refresh would likely cover all three nodes. The next experiment is the Gen4-forced re-run on the newer node, with repeats.



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-2026-033. All measurements were taken on 29 September 2026 (UTC), and the raw benchmark logs, JSON output and session records are retained.