PURETENSOR
INTELLIGENCE DISTILLED

Workstation Blackwell, Part 3

A 228-Billion-Parameter MoE Reasoning Model via TCP RPC

Paper: *PT-R-2026-006 v1.0*
Author: *PureTensor Ltd*
Date: *19 April 2026*
Status: *Published, Part 3 of a multi-part programme*

Abstract

Part 1 of this programme characterised a two-node, three-GPU workstation-class NVIDIA Blackwell cluster connected by a 200 Gbps Ethernet-based RDMA fabric, and demonstrated cross-node pipeline-parallel inference of a 405-billion-parameter dense language model under a vLLM and Ray stack with NCCL as the collective library. Part 2 retargeted the same cluster to GPU dataframe analytics and gradient-boosted tree training on the Criteo click-through-rate corpus over a Dask-CUDA stack with UCXX as the transport. Part 3 returns to language-model inference but under a deliberately different framework stack and a materially different model architecture: a 228.69-billion-parameter mixture-of-experts reasoning model, quantised to Q8_0 for a 226 GiB on-disk footprint, distributed across the three Blackwells under `llama.cpp` in remote-procedure-call mode over plain TCP. The paper reports model load behaviour, aggregate VRAM residency, prompt-evaluation throughput, sustained generation throughput, the bursty utilisation profile induced by sequential RPC dispatch, and a visible sample of the reasoning model's output. The configuration serves a frontier-scale sparse model at conversational throughput on a workstation-class sovereign compute cluster, without a hyperscaler fabric and without NVLink, and extends the programme's demonstration of workstation-class Blackwell clusters to sparse reasoning models of practical deployment size.

1. Introduction

The research programme introduced in Part 1 of this series evaluates workstation-class professional NVIDIA GPUs connected by Ethernet-based RDMA as a deployment target for workloads that usually presuppose hyperscaler-grade infrastructure. Part 1 demonstrated cross-node inference of a 405-billion-parameter dense language model under a vLLM and Ray stack, with NCCL bus bandwidth reaching approximately 92% of the 200 Gbps fabric line rate. Part 2 shifted the workload to GPU dataframe analytics and gradient-boosted tree training on the Criteo corpus over a Dask-CUDA stack, and established that the same physical fabric sustains a realistic click-through-rate pipeline end-to-end without worker failure.

Part 3 returns to language-model inference, with three deliberate departures from Part 1. First, the model is a sparse mixture-of-experts rather than a dense transformer: **MiniMax-M2.7**, with 228.69 billion total parameters, 256 experts per mixture-of-experts layer, and eight active experts per token. The activation-sparse parameter count is approximately 10 billion per token,

and the model is trained and released as a reasoning model with an explicit thinking channel separated from the final-answer channel. Second, the framework stack is `llama.cpp` rather than vLLM, and the inference topology is driven by `llama.cpp`'s native remote-procedure-call protocol rather than by NCCL collectives. Third, the model is distributed as a GGUF with 8-bit weight quantisation (Q8_0) across six shards, totalling 226.43 GiB on disk, which is below the cluster's aggregate 288 GB of GPU memory but well above any single Blackwell in the cluster.

These three departures exercise a regime complementary to Part 1. NCCL over RDMA is the correct transport for dense collective inference when the parallelism pattern is tensor-parallel or pipeline-parallel and activations are large. `llama.cpp` RPC over TCP is the correct baseline for sparse MoE inference where activations per token are small, expert routing is localised per layer, and the dominant cost is on-device compute on each of the eight routed experts. The question answered here is whether a single-operator, workstation-class cluster with no NVLink and no InfiniBand can serve a 228-billion-parameter sparse reasoning model at conversational throughput, and if so, by how much the sparse activation pattern forgives the slower transport.

2. System Under Test

The hardware is unchanged from Parts 1 and 2. The software stack above CUDA differs: in place of vLLM (Part 1) or RAPIDS and Dask-CUDA (Part 2), this paper uses `llama.cpp` with native remote-procedure-call support compiled in. The transport is TCP on the loopback-facing IP of the 200 Gbps Mellanox NIC, not NCCL and not UCXX; the physical fabric, driver and CUDA runtime are the same.

Component	Specification
Node A (local)	AMD Ryzen Threadripper PRO 9975WX (64 threads), 512 GB DDR5 ECC, two NVIDIA RTX PRO 6000 Blackwell Workstation Edition GPUs, 96 GB VRAM per GPU, PCIe Gen 5 x16 per GPU, 200 Gbps Mellanox ConnectX-6 NIC
Node B (remote)	AMD Ryzen Threadripper 7970X (64 threads), 256 GB DDR5, one NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition GPU, 96 GB VRAM, PCIe Gen 5 x16, 200 Gbps Mellanox ConnectX-6 NIC
Pooled GPU memory	3 x 96 GB VRAM, approximately 288 GB aggregate; 276.98 GiB reported free to the loader at cold start with dictation services resident
Switch and fabric	MikroTik CRS812-4XS-RM with 200 Gbps QSFP56 ports; RoCE-capable Mellanox link. No NVLink, no InfiniBand.
Transport	llama.cpp native RPC protocol over plain TCP on port 50052; rpc-server on Node B exposes its local GPU as a remote device to llama-server on Node A
Storage	Consumer NVMe Gen 4 on both nodes; GGUF shards resident on Node A bulk storage and memory-mapped by the loader; no remote weight streaming

Component	Specification
Software	Linux 6.17, NVIDIA driver 595.58, CUDA 13.2 toolkit, llama.cpp build 8070 with GGML_CUDA=ON, GGML_RPC=ON, CMAKE_CUDA_ARCHITECTURES=120 (sm_120 Blackwell)

3. Model Under Test

The model under test is MiniMax-M2.7, released by MiniMax AI and redistributed in 6-shard Q8_0 GGUF form by Unsloth. Architecturally it is a mixture-of-experts transformer with the following shape, read from the GGUF metadata at load time:

Property	Value
Architecture	minimax-m2 (MoE transformer)
Total parameters	228.69 B
Size label	256 experts x 4.9 B per expert
Active experts per token	8 of 256 (approx. 3.1% of the expert population)
Active parameters per token	approximately 10 B (the per-token compute budget)
Transformer blocks	62 blocks, plus one output projection
Embedding dimension	3072
Feed-forward dimension (per expert)	1536
Attention heads	48 query, 8 key/value (grouped-query attention with 6:1 ratio)
Trained context length	196 608 tokens (192 K)
RoPE base frequency	5 000 000 (linear scaling, no YaRN)
Quantisation	Q8_0 (8-bit weight quantisation, 8.51 bits per weight including metadata)
On-disk size	226.43 GiB across 6 GGUF shards, 809 tensors
Calibration	Unsloth importance-matrix (imatrix) over 81 chunks, 496 entries
Chat template	MiniMax-M2 reasoning template: separate reasoning_content and content fields in OpenAI-compatible responses

Two properties of this architecture are material to the fabric question. First, with 8 active experts per token and approximately 10 B active parameters, the per-token compute is only about 4.4% of the per-token compute a dense 228 B model would require. Second, routing is per-token and per-layer, so the activations that cross GPU boundaries are small compared to the tensor-parallel activations of a dense model of equivalent parameter count. Sparse MoE

inference is therefore well matched to a lower-bandwidth transport than Part 1's NCCL-over-RDMA path and is the natural first workload to exercise `llama.cpp`'s RPC transport on this cluster.

4. Methodology

4.1 Build and deployment

`llama.cpp` is built from source on Node A at build 8070 (`cc45f2ada`) against CUDA 13.2 with the RPC backend enabled. The configure command is `cmake -B build -DGGML_CUDA=ON -DGGML_RPC=ON -DCMAKE_CUDA_ARCHITECTURES=120 -DLLAMA_CURL=OFF` and the targets built are `llama-server`, `rpc-server` and `llama-cli`. The sm_120 architecture flag selects the native Blackwell compute capability and enables the runtime's `BLACKWELL_NATIVE_FP4` code path, although FP4 is not exercised in this paper.

On Node B, the build tree is not regenerated from source. Node B's CUDA toolkit and driver versions match Node A (CUDA 13.2, sm_120), so the Node A binary artefacts are rsynced directly into Node B's `build/bin` directory and executed with `LD_LIBRARY_PATH` pointing at the runtime's own library directory and the system CUDA 13.2 `lib64`. This is the minimum viable cross-node distribution path and avoids rebuilding the full toolchain on the remote node. The `rpc-server` binary on Node B detects its local Blackwell Max-Q at start-up and advertises it as a remote device over TCP.

4.2 Inference topology

`rpc-server` is launched on Node B bound to 10.200.0.1:50052 with sixteen worker threads. `llama-server` is launched on Node A with `--rpc 10.200.0.1:50052 -ngl 999 --tensor-split 1,1,1 --ctx-size 4096 --parallel 1`. The effect is that `llama.cpp` sees three devices in total: the two local Blackwell Workstation Editions on Node A (CUDA0, CUDA1) and the single remote Blackwell Max-Q on Node B (RPC0). The tensor-split flag partitions the model's layers evenly across the three devices. The loader's fit-to-device-memory pass reports the projected residency before loading:

Device	Total (MiB)	Projected used (MiB)	Projected free (MiB)
RPC0 (10.200.0.1, Node B)	97 249	78 539	15 469
CUDA0 (Node A, GPU 0)	97 250	78 538	15 446
CUDA1 (Node A, GPU 1)	97 247	75 733	19 907
Aggregate	291 746	232 810	50 822

The loader reports 232 810 MiB (approximately 227 GiB) projected device residency against 283 634 MiB (approximately 277 GiB) of free device memory at cold start, giving an effective

utilisation of approximately 82% of the GPU memory pool after dictation and base-system residency is accounted for. The split is balanced: two of the three devices carry 78.5 GiB of model weights each, while the third (CUDA1) carries a slightly lower 75.7 GiB because the model's layer count is not an exact multiple of three.

4.3 Service-drain protocol

Before loading the model, GPU-resident services are drained on both nodes to expose the available VRAM to the loader. The drained services on Node A are the `arabic-qa`, `f5-spanish`, `habibi-tts`, `xtts-v2`, `rvc-icelandic` and `vantage-consumers` system services, and the `puretensor-tts` user-scope service. The `whisper` automatic-speech-recognition service is deliberately not drained on either node, because the operator's dictation channel depends on it; approximately 5 to 10 GB of VRAM is retained for whisper residency on each node and is reflected in the loader's cold-start free-memory budget above. The drain is reversible: the original set of services is restored after the benchmark completes.

4.4 Benchmark protocol

Three prompts are issued against the live `llama-server` process. The first exercises the server's base `/completion` endpoint with a short technical prompt and a 128-token generation budget. The second exercises the same endpoint with a 512-token budget to measure sustained generation rate over a longer output window. The third exercises the OpenAI-compatible `/v1/chat/completions` endpoint with the model's native chat template and a 2048-token budget, which is sufficient for the model's reasoning phase to complete before the final-answer channel is reached. Server-reported timings (`prompt_per_second` and `predicted_per_second`) are recorded for each request, along with per-GPU utilisation sampled at one-second resolution during the long-generation request.

5. Results

5.1 Load and residency

Cold-start model load from the six on-disk Q8_0 GGUF shards completes without error. All 62 repeating transformer blocks plus the output projection are offloaded to GPU (63 of 63 layers offloaded), with 622.76 MiB of embedding-table weight retained on the host as a CUDA-mapped buffer. The final per-device residency after load and warm-up is given below; it matches the loader's fit projection within rounding:

Device	Model buffer (MiB)	KV cache (MiB)	Compute (MiB)	Total (MiB)
CUDA0 (Node A, GPU 0)	78 113.21	336.00	100.01	78 549

Device	Model buffer (MiB)	KV cache (MiB)	Compute (MiB)	Total (MiB)
CUDA1 (Node A, GPU 1)	75 016.30	320.00	396.75	75 733
RPC0 (Node B, Max-Q)	78 113.21	336.00	90.01	78 539
Aggregate	231 242.72	992.00	586.77	232 821

The steady-state GPU residency observed externally (via `nvidia-smi`) at idle between requests is 80.0 GiB, 75.7 GiB and 80.0 GiB for CUDA0, CUDA1 and RPC0 respectively, for an aggregate residency of 235.7 GiB, consistent with the loader's internal accounting plus the KV cache pre-allocation at the 4096-token context size used here. The model fits with approximately 50 GB of aggregate GPU memory held in reserve for context growth and for reservation headroom.

5.2 Prompt evaluation

Prompt-evaluation (prefill) throughput, measured on the OpenAI-compatible chat request, is 337.41 tokens per second over 53 tokens of input (157.08 ms wall), and 335.83 tokens per second over 57 tokens of input (169.73 ms wall) on a second independent request. The measurement is server-reported via the `timings.prompt_per_second` field. At Q8_0 over 228.69 B parameters split across three devices, prefill is compute-bound on the active-expert set for each token rather than fabric-bound, and the prefill rate is the correct upper bound on how quickly the server can ingest a prompt at this context size.

5.3 Generation throughput

Generation throughput is measured on three independent requests with increasing output budget. Results are server-reported via the `timings.predicted_per_second` field:

Request	Endpoint	Output tokens	Generation rate	Wall time
Short completion	/completion	128	82.7 tok/s	1.55 s
Long completion	/completion	512	83.3 tok/s	6.14 s
Chat completion	/v1/chat/completions	325 (reasoning + answer)	82.35 tok/s	4.10 s

Sustained generation is between 82 and 83 tokens per second across all three requests, with a run-to-run variance of under 1% of the mean. This is approximately 12 ms per generated token at the measured rate, dominated by a forward pass through 62 MoE layers where each layer executes eight routed expert feed-forward blocks and a shared grouped-query attention block, with the work partitioned across the three devices in the layer order fixed at load time.

5.4 GPU utilisation profile

Per-GPU utilisation, sampled at one-second intervals via `nvidia-smi` during the long-generation request, is bursty and spiky rather than saturated. Peak samples reach 31% on the local Blackwells and 28% on the remote Max-Q; the median sample is in the 18-22% range, with visible zero-utilisation gaps between spikes. This pattern is the expected fingerprint of `llama.cpp`'s sequential RPC dispatch: the three devices are tensor-split across layer groups, so only one device executes any given layer of a forward pass while the other two wait for the result. No pipeline-parallel overlap is performed; the tensor-split strategy in `llama.cpp` RPC is strictly sequential.

The operational consequence is that the 82 to 83 tokens-per-second figure measured in Section 5.3 is a single-stream, non-pipelined throughput. A pipelined or tensor-parallel framework such as vLLM on the same cluster would produce a materially higher peak generation rate and materially higher sustained GPU utilisation, at the cost of a more complex deployment path. `llama.cpp` RPC is the correct baseline for understanding the no-optimisation floor of the cluster on this model.

5.5 Reasoning output behaviour

The chat-completion request exercises the MiniMax-M2 reasoning template. The OpenAI-compatible response returns two populated fields on each assistant message: `reasoning_content` contains the model's internal chain-of-thought, and `content` contains the final user-visible answer. A short generation budget that is consumed during reasoning will terminate with `finish_reason = "length"` and an empty `content` field; a sufficient budget allows the model to complete its thinking, emit the end-of-thought marker, and produce the visible answer. The chat request in Section 5.3 was issued with a 2048-token budget and terminated naturally with `finish_reason = "stop"` after 325 generated tokens, of which approximately 195 were internal reasoning and 130 were visible answer.

The visible answer produced by the model, in response to the prompt "*In two paragraphs, explain how a mixture-of-experts transformer distributes inference across multiple GPUs, then state which design choice in this architecture makes it forgiving of slow interconnects*", is reproduced below (em dashes substituted to parentheses for house style; no other edits):

In a mixture-of-experts transformer, the feed-forward layers are replaced by multiple parallel "expert" networks, with a gating mechanism that routes each token to only a subset of experts (typically top-k). During inference, this sparsity means only the selected experts need to be active and loaded into GPU memory at any given time. Experts are distributed across GPUs, and the router sends each token to the appropriate device(s) based on learned routing weights. This allows the model to scale parameters far beyond what fits on a single GPU while only activating a fraction per token.

The design choice that makes MoE forgiving of slow interconnects is the sparse activation pattern combined with top-k routing. Since each token only communicates with a small number of experts (e.g., 2 out of 128), communication overhead is minimal compared to dense models where all GPUs must synchronize every layer. The router's locality (tokens typically route to the

same few experts) means cross-GPU traffic is predictable and bounded, reducing sensitivity to interconnect bandwidth and latency.

The quantitative detail is not exact to the model under test (MiniMax-M2 uses 8 experts of 256, not 2 of 128), but the mechanism described is correct and corresponds to the architecture reported in Section 3. The sample is included as evidence that the quantisation level, the RPC transport and the cross-node tensor split preserve the model's coherent output.

6. Discussion

Three observations are worth preserving from the measurements. First, 82 to 83 tokens per second on a 228.69 B-parameter sparse model is a throughput that is indistinguishable from conversational pace for a single user; the model serves interactively on a workstation-class cluster with no hyperscaler components. Second, the 235.7 GiB residency figure is the definitive statement that this cluster is load-bearing for frontier-scale sparse models: no single Blackwell in the cluster, and no two Blackwells in the same node, can hold the model alone. Only the three-Blackwell aggregate with cross-node transport makes the deployment possible at Q8_0. Third, the bursty utilisation profile confirms that the fabric is not the bottleneck in this configuration: the devices are most frequently idle while waiting for the sequential RPC dispatch, not while waiting for the 200 Gbps transport.

A corollary is that a pipelined or tensor-parallel MoE inference stack, for example vLLM with mixtral-style expert-parallel sharding across the three Blackwells, is expected to produce a markedly higher throughput on the same hardware. The contribution of this paper is the characterisation of the no-optimisation floor under `llama.cpp` RPC, not the ceiling under a throughput-optimised framework. Future parts of the programme will report the ceiling measurement.

The reasoning-model output behaviour described in Section 5.5 has an operational consequence for any OpenAI-compatible client routing to this endpoint: clients that display only `message.content` and ignore `message.reasoning_content` can observe an apparently empty response when the generation budget is too small for the thinking phase to complete. The practical remedy is either a generation budget large enough to cover the thinking phase and the visible answer, or an explicit instruction to the model to suppress the reasoning channel. This is a property of reasoning-trained models rather than of the transport or cluster.

7. Related Work and Position in the Programme

`llama.cpp`'s RPC backend is a community-maintained cross-device abstraction over the GGML runtime, originally authored for heterogeneous single-node inference across CPU and GPU backends and later extended to multi-host deployments over TCP. Published third-party measurements of `llama.cpp` RPC on workstation-class clusters have until recently focused on

dense models in the 30 B to 70 B range. The measurements reported here extend the known operating envelope to a 228.69 B-parameter sparse MoE model at Q8_0, confirming both correctness and conversational throughput at this scale.

Within this programme, Part 1 established the upper end of the dense-model deployment envelope (405 B dense via vLLM and NCCL) and Part 2 established the dataframe-analytics envelope (Criteo CTR via Dask-CUDA and UCXX). Part 3 adds a third operating point: sparse MoE inference at 228 B under a lower-optimisation framework and a minimal transport, establishing the no-optimisation floor. The three operating points together describe a usable volume in the framework-fabric-model product space rather than a single point in it.

8. Conclusion and Roadmap

Part 3 extends the programme by demonstrating that a workstation-class three-Blackwell cluster with no NVLink and no InfiniBand can serve a 228.69-billion-parameter mixture-of-experts reasoning model, at Q8_0 across 226 GiB of on-disk weights, at a sustained generation rate of 82 to 83 tokens per second over a plain-TCP RPC transport. The model load balances across the three devices to within 3 GiB of perfect symmetry, the aggregate GPU residency is 235.7 GiB, and the reasoning template preserves the model's thinking and final-answer channels through the OpenAI-compatible API. No hyperscaler fabric, no NVLink, and no DGX-class chassis were required to reach this operating point.

Subsequent parts in the programme are scoped as follows, with numbering updated to accommodate this paper:

- **Part 4.** Fabric upgrade to 400 Gbps RoCE (ConnectX-7) and characterisation of the 200 Gbps to 400 Gbps delta for both dense-model collective inference and distributed dataframe shuffle.
- **Part 5.** UCXX-over-InfiniBand once a system-level UCT provider is in place, and the resulting shuffle-throughput delta against the TCP baseline reported in Part 2.
- **Part 6.** Throughput-optimised sparse-MoE inference, targeting vLLM or SGLang with expert-parallel sharding on the same MiniMax-M2.7 model, so the no-optimisation floor reported here can be paired with a ceiling measurement.
- **Part 7.** Full target-encoding of all 26 Criteo categorical features into XGBoost training and a fair Criteo AUC number comparable to published Merlin baselines (carried forward from Part 2).
- **Part 8.** Consolidated whitepaper covering the full programme and an open reproducible benchmark harness.

All launcher scripts and raw result JSON produced for this paper are released with the programme's harness repository. Third parties running the same launcher on comparable workstation-class Blackwell hardware and the same Unsloth Q8_0 GGUF can produce directly comparable result files.

9. Acknowledgements

The cluster described in this paper was assembled under the NVIDIA Inception programme, which grants member startups access to professional NVIDIA hardware through NVIDIA's distributor network. The MiniMax-M2.7 model is published by MiniMax AI and the Q8_0 GGUF redistribution used in this paper is the work of the Unsloth project. The `llama.cpp` framework and its RPC backend are the work of the upstream `llama.cpp` community.