AI-generated living document. This report was written by AI models from our training logs, telemetry and code, and has not been fully reviewed by people. It changes as we tune and adjust the training runs: numbers, figures and conclusions may differ between versions. This version: September 29, 2026.

TECHNICAL REPORT · SEPTEMBER 2026

Parallax: Decentralized Training of a Ternary Mixture-of-Experts Model on Consumer GPUs over Wide-Area Links

The Parallax Team, Chutes
chutes.ai

Abstract. Parallax is a mixture-of-experts language model, 7.8 billion parameters, with 1.25 billion of them active per token. We are training it on 200 consumer RTX 5090 GPUs in 25 hosts on three continents. The hosts have no connections to each other. Synchronization passes through four CPU relays over the public internet, and the hosts read their data from streaming corpus servers. This report covers the current run, kappa, up to 200 billion of a planned 972.9 billion tokens.

There are 64 layers. They mix gated delta-rule recurrence, sliding-window attention and sparse attention, with block attention residuals. The 4096 routed experts have ternary weights. At most two nonzero pairs are allowed in every group of eight, and we store the trits at 1.125 bits per weight.

Trunk synchronization is a relay-side decoupled DiLoCo scheme. Nodes send 4-bit packets, sparse except for the scalar state of the recurrent layers, for every trunk tensor but the routers, which are averaged from dense snapshots. The relays merge the packets with token weights, radially for the matrices of the token mixers, latent projections and shared experts. Error feedback travels on a separate debt lane, and the outer momentum is coupled to that lane. Each routed expert has one owner GPU, which keeps the expert's full-precision master and a momentum buffer. All GPUs train rank-16 adapters on the experts they use. At each fold the owner applies the weighted mean of the contributing nodes' clipped adapter changes to the master through an orthogonalized momentum step, re-quantizes it and publishes a new ternary version. The canonical trunk and expert state lives on the relays, nodes hydrate from it when they join, and training nodes write no periodic checkpoints.

The run takes many optimizer steps per token, 8.2 million tokens per fleet step at launch. It bounds the output logit scale at 3, keeps the logits in fp32 inside the loss kernel and trains on a mixture of web text, code, mathematics, knowledge text and books. At 85 billion tokens we doubled the batch without a restart. Throughput rose by 22%, and the validation loss has since stayed below a curve fitted to the earlier exports. With 20 micro-batches per step the fleet processes 2.36 million tokens per second at a median model FLOP utilization of 43.2%. A node sends about 10.0 bytes upstream per token it processes, a median of 10.7 Mbit/s. After 192 billion tokens the exported model has a validation loss of 2.176 nats and a zero-shot benchmark macro of 53.7% over eleven tasks. In its native format the model needs 0.567 billion multiply-bearing weight operations per decoded token and an estimated 2.17 GB of storage with bf16 trunk weights and fp16 row scales.

1 Introduction

Language models are usually pretrained on accelerators inside a datacenter. There the hosts are connected by a dedicated high-bandwidth network. Data-parallel training with a gradient all-reduce sends about twice the model's size per worker and step. For our 7.8-billion-parameter model in bf16 that is 31 GB per step. At the median step time of this run with 20 micro-batches per step, 6.8 s, each worker would need to send about 37 Gbit/s. The nodes of this run are connected to each other only through the public internet, and each sends a median of 10.7 Mbit/s (Section 10). The RTX 5090 we use also has 32 GB of memory, too little to hold the full-precision weights and optimizer state of a model this size on every device.

Low-communication methods cut this traffic. DiLoCo and its descendants synchronize parameters only once every tens to hundreds of steps [11–13]. SparseLoCo sparsifies and quantizes what is sent [46]. Permissionless runs have used these methods with datacenter-class participants [22, 30]. For a dense model, pipeline-parallel training has been run over consumer GPUs [3]. A mixture-of-experts (MoE) model keeps few of its parameters active per token. The routed experts are still most of the parameters. They are updated sparsely and unevenly, and averaging all of them periodically would dominate the traffic.

Parallax is a 7.8B-parameter MoE language model with 1.25B active parameters. We train it on 200 RTX 5090 GPUs in 25 hosts on three continents. The hosts have no connections to each other. They synchronize through four CPU relays over the public internet, and a separate authority service holds the sparse-attention indexer. This report describes the current training run, kappa. It is planned as one pass over 972.9 billion tokens of a nine-source text mixture, and the measurements reported here were taken at 21% of that budget.

Synchronization is separate for the two parts of the model. The dense trunk holds 7.2% of the parameters. Every node trains it, and the relays merge it following Decoupled DiLoCo. The routed experts hold the other 92.8%. Each expert is owned by one GPU, which keeps the full-precision master but computes no gradients of it. All GPUs train low-rank adapters on the experts they use and send them to the owner. At each fold the owner applies the weighted mean of the contributing nodes' adapter changes to the master through an orthogonalized momentum step and publishes a new version in a ternary, pair-sparse format. Inference uses that same format.

The contributions are:

- A trunk synchronizer on the relays. It accepts asynchronous, heterogeneous nodes. Nodes send 4-bit packets for all trunk tensors except the routers, sparse except for the scalar state of the recurrent layers; the relays merge them with token weights, radially for the matrices of the token mixers, latent projections and shared experts. The routers are averaged from dense snapshots. Error feedback goes on a separate debt lane, and the outer momentum is coupled to the accepted debt (Section 5).
- An expert lane with one owner per expert, in which every change of an expert is a Muon step with Nesterov momentum on the fleet mean of rank-16 adapters, with a norm tied to the mean's norm. The experts run in a 384-dimensional latent space, which the adapters of the fleet can span (Section 6).
- Ternary experts with at most two nonzero pairs in every eight weights. The format has 417 patterns, stored at 1.125 bits per weight, and the pair support is trained with a hysteresis. Decoding one token takes 0.567 billion multiply-bearing weight operations (Sections 4 and 9).
- Membership with no periodic checkpoints on the training nodes. A node joins from relay state. A running learner replaces an expert owner that is lost. Inference models are assembled on the relays (Section 7).
- A training recipe for this setting: many optimizer steps per token, a bounded output logit scale, a loss computed on fp32 logits and a mixed corpus (Sections 4 and 8).
- Measurements from the run: loss and benchmark accuracy, a batch doubling made during training and read against a fitted control, throughput, utilization, heterogeneity and wide-area traffic, and a systems comparison with Agora (Section 10).

With 20 micro-batches per step the fleet processes 2.36 million tokens per second at a median model FLOP utilization of 43.2%. A node sends about 10.0 bytes upstream per token it processes, about 1/4,780 of what a bf16 gradient all-reduce would send at the current batch. After 192 billion tokens the exported model scores a zero-shot benchmark macro of 53.7% and a validation loss of 2.176 nats.

2 Related Work

Low-communication data parallelism. Local SGD and its variants replace per-step gradient averaging with periodic averaging of parameters [32, 49, 56, 64]. In DiLoCo [11], workers run many inner AdamW steps, and an outer Nesterov step is applied to their averaged pseudo-gradient. Streaming DiLoCo [12] synchronizes fragments of the parameters on a staggered schedule and overlaps communication with computation. Charles et al. [7] give scaling laws for the method. Decoupled DiLoCo [13] keeps a single canonical state and the outer optimizer on a synchronizer. That synchronizer admits asynchronous, lossy learners, with quorum, grace windows and token-weighted merging. Our trunk lane has the same structure. We place the synchronizer on relays. SparseLoCo [46] sparsifies and quantizes the pseudo-gradients with error feedback [25, 47] and uses the error-feedback accumulator in place of outer momentum.

Our trunk packets are in that same family. They carry 4-bit values, and the residual is kept on a separate debt lane. Outer momentum is coupled to the accepted debt. The merge is radial, related to regularized dual averaging [60]. MuLoCo [52] studies Muon as the inner optimizer. Muon is our inner optimizer as well. DeMo [42] sends a compressed form of the momentum. Earlier work compressed gradients by quantization [2, 4, 59], by sparsification [1, 33, 50] and by low-rank compression [53].

Decentralized and permissionless pretraining. Learning@home [43] proposed training a mixture of experts on volunteer hardware, with the experts spread across participants. DeDLOC [10], SWARM parallelism [44] and Petals [5] took shared training and inference onto unreliable hosts. INTELLECT-1 [22] used DiLoCo for a 10B dense model, 1T tokens, across up to 14 datacenter nodes on three continents. Covenant-72B [30] trained a 72B dense model with SparseLoCo. Its participants were permissionless, and each had datacenter-class GPUs. Agora [3] trained a dense 8.6B-parameter model on 500B tokens of FineWeb-Edu over consumer GPUs, using pipeline parallelism, so no participant holds the full model. SPES [63] and FoMoE [45] synchronize experts sparsely across workers. FoMoE partitions the experts between workers. It is the closest published analogue to our owner-sharded expert lane.

Low-rank training. LoRA [20] trains a low-rank update of a frozen matrix. ReLoRA [29] periodically merges that update into the base and restarts it, so that the accumulated update has high rank. Our expert lane borrows the periodic merge of this scheme and distributes it: every node trains adapters, and at each merge the expert's owner applies the mean of the contributing nodes' low-rank updates through an orthogonalized momentum step. Two refinements sit on top. rsLoRA [24] scales adapters by $\alpha/\sqrt{r}$. LoRA-Pro [58] corrects the factor gradients so that the low-rank update follows the full gradient more closely. GaLore [65] and LoQT [35] train through low-rank projections of gradients or on quantized bases.

Ternary and sparse weights. BitNet [55] and BitNet b1.58 [37] train transformers with binary and ternary weights. Sparse-BitNet [62] adds 2:4 semi-structured sparsity to ternary weights, the format sparse tensor cores support [38, 66]. Our experts use a pair-granular variant: at most two nonzero pairs in eight weights, with a hysteresis on the pair support. Each expert is quantized by its owner, from a full-precision master that no other worker holds. QMoE [16] compresses mixture-of-experts models below one bit per parameter after they are trained.

Mixture-of-experts routing. Sparsely gated experts [15, 28, 48] are routed by a learned gate, usually with an auxiliary balancing loss. DeepSeek-V3 [8] uses sigmoid gates plus a bias that affects only selection and is adjusted without an auxiliary loss [57]. Quantile balancing [51] sets that bias from quantiles of the routing scores. The router z-loss comes from ST-MoE [67]. LatentMoE [14] runs the experts in a projected latent space. Each expert is smaller as a result, and in our setting the same projection raises the fraction of an expert's input space that rank-16 adapters from the fleet can cover.

Sequence mixers. Our model combines gated delta-rule recurrence [19, 61], sliding-window attention trained with stochastic window sizes [6], MiniMax sparse attention [27] and block attention residuals [26]. Wang et al. [54] study hybrid stacks of linear and full attention. Jamba [31] and Nemotron-H [40] are built from them.

3 System Overview

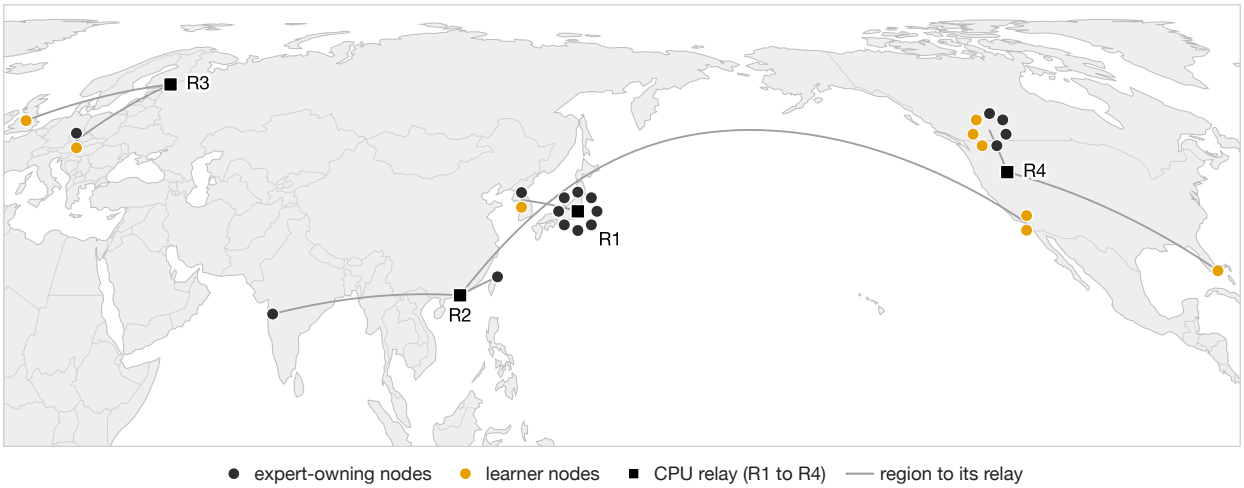


Figure 1. Location of the 25 training nodes and 4 relays. Each dot is a node; nodes of one region are drawn on a ring around the region's location. Lines join each region to its regional relay. Every node also sends trunk packets directly to the home relay of each trunk fragment; these links are not drawn. Relay markers are offset by a few degrees where they coincide with a node region.

Table 1. Hosts and services of the run.

Role	Count	Function
Expert-owning node	16	8 RTX 5090 GPUs; trains the trunk and adapters; each GPU owns 32 routed experts
Learner node	9	8 RTX 5090 GPUs; trains the trunk and adapters
Relay	4	CPU host; holds the canonical trunk and expert state; trunk homes (8, 6, 6 and 4 of 24 fragments), expert version log, evidence mailboxes, model assembly
Data authority	1	global shard cursor; the token clock
Corpus server	5	serve the source files by HTTP byte ranges; four of these hosts also run a relay
Indexer authority	1	publishes the sparse-attention indexer
Coordinator	1	issues leases; detects owner loss and runs failover

3.1 Hosts and roles

The run uses two kinds of hosts (Table 1). There are 25 training nodes with 8 NVIDIA RTX 5090 GPUs each, 200 GPUs in total, located in 9 countries on three continents (Figure 1). Sixteen of them own routed experts (Section 6) and nine are learners. Both roles run the same program; owner GPUs do extra expert-lane work on top of it. Inside a node, the GPUs talk over the node's local interconnect. Nodes have no connections to each other. All synchronization traffic goes through the relays over the public internet. Beyond that, a node contacts only the CPU services listed below, for data, leases and the indexer.

The 4 relays are CPU hosts linked to each other by a private network. They hold the canonical model state and run the trunk's outer optimization. They also store expert versions, deliver expert evidence and assemble inference models (Section 7). The data cursor, the corpus files, the sparse-attention indexer and the membership lease are held by separate CPU services.

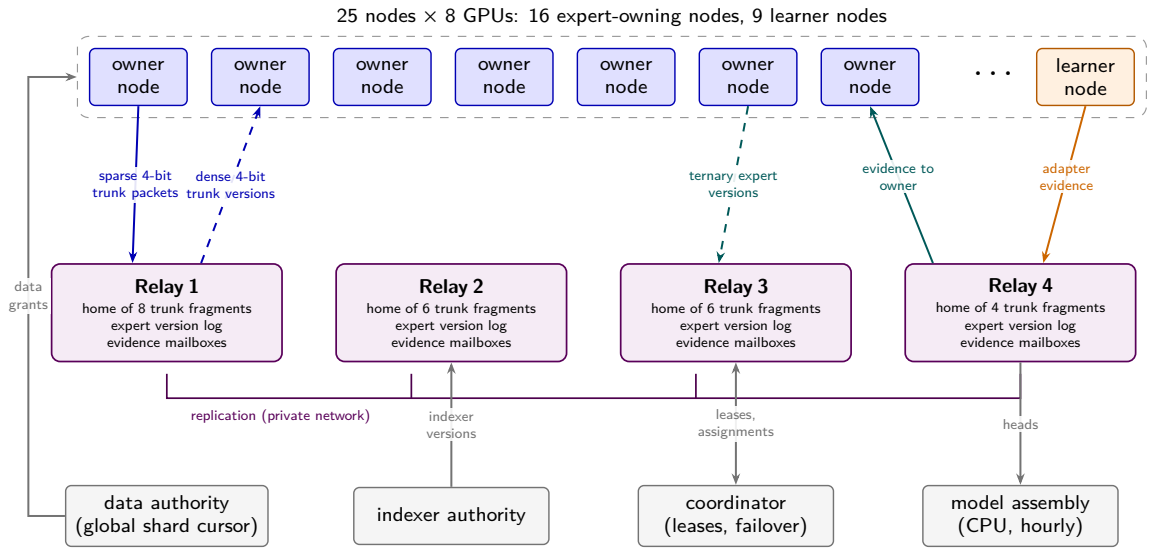


Figure 2. System structure: roles and data flows. Training nodes send sparse 4-bit trunk packets to the home relay of each fragment and receive dense 4-bit versions (the router fragment moves fp32 snapshots); nodes send adapter evidence through relay mailboxes to expert owners, which publish ternary expert versions that every node installs. CPU services provide data grants, indexer versions, leases and model assembly.

Table 2. Synchronization lanes. Cadences are per fragment, per expert or per node.

Lane	Node sends	Node receives	Cadence
Trunk	sparse 4-bit fresh and debt packets, per fragment	dense 4-bit version, per fragment	24 steps
Router	dense fp32 snapshot	fp32 mean	24 steps
Expert	rank-16 adapter factors, 4-bit, with assignment counts	ternary version (trits and row scales)	1.5B tokens per expert
Indexer	indexer evidence	indexer version	per indexer round
Data	grant request	shard grant; corpus byte ranges	one shard per 5,760 sequences

3.2 Lanes

Training state is split into lanes that synchronize on their own, each at its own cadence (Figure 2, Table 2). The trunk lane (Section 5) carries every parameter outside the routed experts. It is cut into 24 fragments. Each fragment is synchronized every 24 optimizer steps. The expert lane (Section 6) carries the 4,096 routed experts. An expert is folded and republished at most once per fold window. The window is 1.5 billion tokens long at the batch sizes of this run. The indexer lane carries the parameters of the sparse-attention indexer. The data lane carries shard grants and corpus reads (Section 8).

No lane requires every node to act on the same step. A slow node can hold up a trunk round only until a capped deadline. A node that goes silent is dropped from the live set after two round periods. A packet built on an outdated version is refused. It goes back into the sender's residual, and that residual is carried into the node's next packet.

3.3 Clocks

Events are ordered by three clocks. Each node counts its own optimizer steps. Warm-up and the architecture switches of Section 8 are functions of this local count, and trunk windows close after a fixed number of steps. The data authority's token clock counts granted tokens and is the same for all nodes. The learning rate and the expert fold schedule are functions of it. Each trunk fragment and each expert has a version number. We describe a node's state by the versions it has installed. Every model the relays assemble records the vector of versions it contains.

Table 3. Architecture of Parallax.

Width d	1152
Layers	64: 32 mixers (even indices) alternating with 32 MoE layers (odd indices)
Mixers	18 GDN2, 8 sliding-window attention (SWA), 6 MiniMax sparse attention (MSA)
GDN2	9 heads, $d_k = d_v = 128$, chunk 64, causal short convolution of width 4
SWA	16 query / 4 key-value heads of width 64, window 2048, RoPE $\theta = 10^6$
MSA	16 query / 4 key-value heads of width 128, KV latent of width 256, at most 16 blocks of 128 tokens
MoE	128 routed experts, top-12, shared latent width 384, expert width 2304, relu^2
Shared expert	1 per MoE layer, dense, width 2304, zero-initialized output
Routed weights	ternary, at most 2 nonzero pairs per 8 weights, fp16 row scales
Router	linear, fp32, sigmoid gates, quantile balancing, z-loss 10^{-4}
Depth mixing	block attention residuals over 8 blocks of 8 layers
Vocabulary	Llama-3 tokenizer (128,256 tokens), embedding padded to 128,384, tied
Training context	4096 tokens

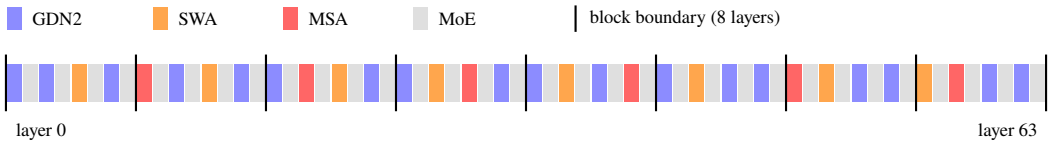


Figure 3. Layer pattern. Mixers occupy even indices and MoE layers odd indices. Vertical rules mark the 8 blocks over which block attention residuals mix depth.

4 Model Architecture

Parallax is a 64-layer hybrid decoder. Token-mixing layers alternate with Mixture-of-Experts (MoE) feed-forward layers. It has 7,813,356,323 parameters, and 1,245,076,259 of them are active per token (Table 4). Neither count includes the transient training adapters of Section 6.

4.1 Token mixers

GatedDeltaNet-2. The 18 GDN2 layers [19] sit inside a pre-norm residual wrapper, $x \mapsto x + \mathrm{GDN2}(\mathrm{RMSNorm}(x))$. Each of the 9 heads reads $h = \mathrm{RMSNorm}(x)$ and computes $q, k, v = \mathrm{SiLU}(\mathrm{conv}_4(W_{q,k,v}h))$. Two gates control the state update. The erase gate $b = \sigma(W_b h)$ acts on the key axis and the write gate $w = \sigma(W_w h)$ on the value axis. The log-decay is channel-wise, $\gamma = -e^{A_{\log}}\,\mathrm{softplus}(W_{f2}W_{f1}h + b_{dt})$, where $W_{f2}W_{f1}$ is a rank-128 factorization. Keys and queries are ℓ_2-normalized as $\hat{k}_t = k_t/\sqrt{\|k_t\|^2 + \epsilon}$ and $\hat{q}_t = d_k^{-1/2} q_t/\sqrt{\|q_t\|^2 + \epsilon}$ with $\epsilon = 10^{-6}$. The per-head state $S_t \in \mathbb{R}^{d_v \times d_k}$ then evolves in fp32 as

$$S_t = S_{t-1}\,\mathrm{diag}(e^{\gamma_t})\left(I - (b_t \odot \hat{k}_t)\hat{k}_t^\top\right) + (w_t \odot v_t)\,\hat{k}_t^\top, \qquad o_t = S_t \hat{q}_t. \tag{1}$$

The head output $\mathrm{RMSNorm}(o_t) \odot \mathrm{SiLU}(W_{g2}W_{g1}h + b_g)$ goes through W_o. We train with the chunkwise WY form of Equation (1) at chunk length 64 and decode with the token recurrence.

Sliding-window attention. The 8 SWA layers use grouped-query attention, with 16 query heads and 4 key-value heads of width 64. They apply RoPE with $\theta = 10^6$ and attend over a 2048-token window. During training the window is sampled each step from $\{128, 2048\}$ with equal probability up to step 112,848, and fixed at 2048 from then on (Section 8).

Sparse attention. The 6 global layers use MiniMax sparse attention (MSA) [27] with 16 query and 4 key-value heads of width 128. For a pre-normalized input h_t, keys and values are up-projected from a 256-dimensional latent $c_t = \mathrm{RMSNorm}(W_{kv}^{\downarrow} h_t)$. In the block-sparse phase we hold the reconstructed keys and values in fp8. The sequence is cut into blocks of 128 tokens, and a query attends to at most 16 of them. One is always the query's own block. The others are the highest-scoring causal blocks under a rank-288 indexer with 4 heads of width 128, which we train with a KL auxiliary loss of weight 0.01 on 256 positions per step. For the first 32,768 optimizer steps attention is dense; after that it is block-sparse.

4.2 Block attention residuals

We replace the plain residual stream with block attention residuals [26]. The 64 layers are grouped into 8 blocks of 8. We write b_0 for the token embedding and b_n for the block sum that block n produces from its layers' residual updates. Layer i in block n receives

$$h_i = \sum_{s \in \mathcal{V}_i} \frac{\exp\big(w_i^\top \mathrm{RMSNorm}(s)\big)}{\sum_{s' \in \mathcal{V}_i} \exp\big(w_i^\top \mathrm{RMSNorm}(s')\big)}\, s, \qquad \mathcal{V}_i = \{b_0, \dots, b_{n-1}\} \cup \{p_i\}, \tag{2}$$

Here $p_i = \sum_{j<i,\, j \in n}(y_j - h_j)$ is the partial sum of the residual updates of the earlier layers in the same block, and the first layer of a block has no p_i term. y_j is the output of layer j. The query $w_i \in \mathbb{R}^d$ starts at zero, so at initialization Equation (2) is a uniform average over its sources. Once the block's last layer has run, the partial sum becomes the block sum, $b_n = \sum_{j \in n}(y_j - h_j)$. A final aggregator of the same form over $\{b_0, \dots, b_8\}$ produces the hidden state for the output head. That makes 65 aggregators in total, each with $2d$ parameters (an RMSNorm gain and w_i).

4.3 Mixture-of-Experts layers

An MoE layer projects its input into a shared latent space, where its 128 routed experts operate, and adds one dense shared expert:

$$y = x + W^{\uparrow} \sum_{e \in \mathcal{S}} g_e\, E_e\Big(W^{\downarrow}\tilde{x}\Big) + W_2^{\mathrm{sh}}\, \mathrm{relu}\Big(W_1^{\mathrm{sh}}\tilde{x}\Big)^2, \qquad \tilde{x} = \mathrm{RMSNorm}(x), \tag{3}$$

All routed experts of the layer share $W^{\downarrow} \in \mathbb{R}^{384\times1152}$ and $W^{\uparrow} \in \mathbb{R}^{1152\times384}$. This is the LatentMoE layout [14], which Nemotron 3 Super also uses [39]. Routed expert e is $E_e(u) = D_e\,\mathrm{relu}(U_e u)^2$ with $U_e \in \mathbb{R}^{2304\times384}$ and $D_e \in \mathbb{R}^{384\times2304}$, or 1,769,472 weights per expert. Working in the 384-dimensional latent cuts per-expert weights and FLOPs by 3× compared with experts on the 1152-dimensional residual. No expert matrix can have rank above 384. The shared expert has width 2304, works on the full residual, and starts with a zero-initialized output projection.

Router. The router computes fp32 logits $z = W_r\tilde{x}$ with $W_r \in \mathbb{R}^{128\times1152}$. We keep selection and weighting apart:

$$\mathcal{S} = \mathrm{TopK}_{12}(z - \beta), \qquad g_e = \frac{\sigma(z_e)}{\sum_{e' \in \mathcal{S}} \sigma(z_{e'})} \quad (e \in \mathcal{S}), \tag{4}$$

where the bias $\beta \in \mathbb{R}^{128}$ affects selection only and never enters the mixture weights. A z-loss $\mathcal{L}_z = 10^{-4}\,\mathrm{mean}_\ell\,\mathrm{mean}_t\,\big(\mathrm{logsumexp}_e\, z_{t,e}\big)^2$ keeps the logit scale in check [67]. We use no auxiliary load-balancing loss.

Quantile balancing. Quantile balancing [51] sets β. For each expert it estimates the bias that would equalize load, then moves β toward that estimate. Take a micro-batch of T tokens with $k = 12$ and $E = 128$, and let a_t be the $(k{+}1)$-th largest entry of $z_t - \beta$. The per-expert target is the descending order statistic

$$\hat{\beta}_e = \mathrm{OS}^{\downarrow}_{Tk/E}\Big(\{z_{t,e} - a_t\}_{t=1}^{T}\Big), \tag{5}$$

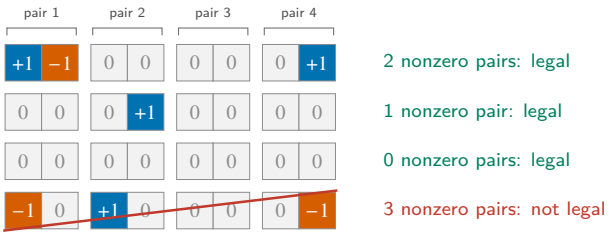
linearly interpolated at the fractional rank Tk/E. In other words, $\hat{\beta}_e$ moves expert e's shifted logits to the rank of its fair share Tk/E of the batch's selections. Targets accumulate over micro-batches and are averaged across the GPUs of a training node. At the optimizer step we commit them as $\beta \leftarrow 0.9\,\beta + 0.1\,\hat{\beta}$ and then center, $\beta \leftarrow \beta - \mathrm{mean}_e\,\beta$. Between nodes, W_r and β are averaged as one parameter family of the trunk synchronization lane (Section 5).

4.4 Pair-sparse ternary experts

Each routed-expert projection has a base weight $W = \mathrm{diag}(\alpha)\,T$, where $T \in \{-1, 0, 1\}^{m\times n}$ and α holds one fp16 scale per output row. That gives 2688 scales per expert (2304 for U_e, 384 for D_e). The base weight is the only expert weight shared across the fleet.

The trits obey a pair-granular sparsity constraint. Along each row, every aligned group of 8 weights splits into 4 pairs, and at most 2 of those pairs may hold a nonzero value. Blackwell FP4 tensor cores use this same pair-granular structured-sparsity pattern; we apply it to ternary values as Sparse-BitNet does [62]. The owner of each expert (Section 6) keeps a full-precision master M and recomputes T and α from it. During training, a forward pass applies the base weight plus the change of a rank-r adapter since its current anchor, $W + s\,(BA - B_0A_0)$. The anchor (B_0, A_0) is the adapter state

(a) 8 consecutive weights of one row, in 4 aligned pairs



Weights are $W = \mathrm{diag}(\alpha)\,T$, $T \in \{-1, 0, +1\}$, one fp16 scale α_r per row.
At least 4 of every 8 trits are zero, the pattern of 2-of-4 pair-structured sparsity.

(b) legal groups of 8 weights

nonzero pairs	pair choices	values per choice	groups
0	1	1	1
1	$\binom{4}{1} = 4$	8	32
2	$\binom{4}{2} = 6$	$8^2 = 64$	384
total			**417**

a nonzero pair takes $3^2 - 1 = 8$ values; $\log_2 417 = 8.70$ bits of information

Codec. 2 pair slots × 4 trits: $\binom{4}{2} \cdot 3^4 = 486 \le 2^9$ symbols,
9 bits per group of 8 weights = **1.125 bits per weight**

Figure 4. Pair-sparse ternary groups. (a) Along a row, every aligned group of 8 weights forms 4 pairs, of which at most 2 are nonzero; the last example violates the constraint. (b) The number of legal groups (Equation (7)) and the 9-bit codec, which stores a group as a choice of two pair slots and four trits.

recorded at the last fold of that expert, as reconstructed from the transmitted update. We use $r = 16$ and $\alpha_{\text{lora}} = 16$, so $s = \alpha_{\text{lora}}/\sqrt{r} = 4$ [24]. Adapters are local training state, and the owner folds them into the base (Section 6).

The quantizer starts from the full-precision master M and the row scales α. It computes $q = \mathrm{clip}(\mathrm{round}(M/\alpha), -1, 1)$ and scores each pair P of each group by how much squared error keeping that pair removes,

$$s_P = \sum_{x \in P,\ q_x \neq 0} \max\left(0,\ 2\alpha|M_x| - \alpha^2\right) + h\,\alpha^2 \mathbb{1}[P \text{ nonzero before}]\,, \tag{6}$$

and then keeps the two highest-scoring pairs and zeroes the other two. The second term is a support hysteresis with $h = 0.0625$ that favors pairs already in the support. Pairs that survive consecutive quantizations also get an element-level band. There, a nonzero element stays nonzero while $|M| \ge \alpha(1-h)/2$, and a zero element turns nonzero only once $|M| > \alpha(1+h)/2$. With the trits frozen, we refit the row scales by least squares as $\alpha'_r = \sum_c M_{rc} q_{rc} / \sum_c q_{rc}^2$. Each new scale is clamped to $\alpha_r(1 \pm 0.01)$ per quantization. A row whose trits are all zero keeps its scale. Section 6 covers when quantization runs and how often trits change.

Alphabet. A legal 8-weight group has zero, one or two nonzero pairs, and a nonzero pair takes one of $3^2 - 1 = 8$ values. Counting the legal groups gives

$$N = 1 + \binom{4}{1}8 + \binom{4}{2}8^2 = 1 + 32 + 384 = 417, \qquad \log_2 417 = 8.70 \text{ bits}. \tag{7}$$

Our wire and export codec encodes a group as one of $\binom{4}{2} \cdot 3^4 = 486 \le 2^9$ symbols, a choice of two pair slots plus four trits. Each group therefore takes 9 bits, and the format costs $9/8 = 1.125$ bits per weight (Figure 4). In this format the routed bank of 7,247,757,312 weights fits in 1.02 GB plus 22 MB of row scales. The same bank in bf16 takes 14.5 GB. Section 9 works out the inference cost.

4.5 Embedding and output head

Input and output embeddings share one tied matrix $E \in \mathbb{R}^{128384 \times 1152}$. Inputs are scaled by $\sqrt{d}$. From the final hidden state x the head computes logits $= e^a\, E\, \mathrm{RMSNorm}_1(x)$. Here $\mathrm{RMSNorm}_1$ has its gain renormalized to unit RMS, and a is a learned fp32 scalar that starts at 0. We bound the scale: the forward pass uses $\min(e^a, 3)$, and every node clamps a to $\log 3$ after each optimizer step. Without a bound, AdamW moves a scalar with a small but persistent gradient bias by close to the full learning rate at every step, so e^a can grow throughout training. The gradient of the tied embedding's output path grows with the scale, and with a global clipping norm it then takes a growing share of every trunk update. Large logits also make the bf16 rounding of the loss kernel costly (Section 8). At scale 3 the largest logits stay near 65 in our synthetic tests.

Table 4. Parameter counts. The unit is one layer, except for routed experts, where it is one expert. The active count includes the tied embedding matrix once, as the output head.

Component	Per unit	Total	Active per token
Tied embedding / output head		147,898,368	147,898,368
GDN2 (×18)	8,569,865	154,257,570	154,257,570
SWA (×8)	2,950,272	23,602,176	23,602,176
MSA attention (×6)	5,277,056	31,662,336	31,662,336
MSA indexer (×6)	847,872	5,087,232	5,087,232
Router (×32)	147,584	4,722,688	4,722,688
Latent projections (×32)	884,736	28,311,552	28,311,552
Shared expert (×32)	5,308,416	169,869,312	169,869,312
Norms, block residual queries, logit scale		187,777	187,777
Non-routed subtotal		565,599,011	565,599,011
Routed experts (32×128)	1,769,472	7,247,757,312	679,477,248
Total		7,813,356,323	1,245,076,259

5 Trunk Synchronization

The model's parameters are split into two lanes. The expert lane (Section 6) holds the 4,096 routed experts. Everything else belongs to the trunk lane, 565.6 million parameters in total: embeddings, token mixers, block attention residuals and routers, along with the latent projections, shared experts and norms. Every training node computes gradients for the full trunk. The progress each node makes on it has to be merged back into one model.

5.1 Topology and roles

The fleet has 25 training nodes of 8 GPUs each, plus 4 relays. Inside a node, the 8 GPUs all-reduce trunk gradients every optimizer step over the local interconnect. As far as the trunk is concerned, a node is a single data-parallel learner. There are no node-to-node connections. A node talks only to relays, over the public internet, and the relays talk to each other over a private network.

We follow Decoupled DiLoCo [13] and run the outer optimization on the relays. The trunk is split into $P = 24$ fragments, each with one home relay. Twenty-three fragments carry the RDA and averaged classes defined below. The remaining one carries the router family (Section 5.9). For an RDA/averaged fragment p, the home keeps the canonical published state Pub_p in bf16, an fp32 outer velocity u_p for each class, and a publication residual R_p. It admits packets from nodes and merges them, then applies the outer step and publishes a new version of Pub_p. The outer step runs on the relay's CPU and never blocks a learner's GPU.

5.2 Parameter classes

Each trunk tensor falls into one of three merge families.

- *RDA class*: the weight matrices of the token mixers, the latent projections and the shared experts. A packet carries a normalized direction and a magnitude. The home merges packets with the radial rule of Section 5.5.
- *Averaged class*: norms, block attention residual parameters, the output logit scale, the dense scalar state of the GDN2 layers, and slices of the tied embedding. Packets here carry sparse parameter displacements. The home averages them with token weights.
- *Router family*: for every MoE layer (Section 4), the router weights W_r, the quantile-balancing bias β, and an adaptive bias that quantile balancing leaves unused. The home merges them with an equal-weight mean of dense snapshots (Section 5.9).

The MSA indexer is not part of the trunk. It trains on a separate lane with its own authority.

5.3 Per-node encoding

Each node keeps a separate window for every RDA/averaged fragment. Fragment p on node i closes its window once $H = 24$ optimizer steps have passed since the node last installed a version of p, provided no install is pending. We

stagger the initial offsets, $o_p = (1 + p) \bmod 24$, so that on average one fragment closes per step. During the window the node counts its steps c_i^{st} and the tokens it processed, c_i^{tok}.

When the window closes, the node forms the displacement $\Delta = \theta_i - \text{Pub}_p$. Here θ_i is the node's highest-precision copy of each parameter. For a bf16 parameter trained with AdamW that is its fp32 master; for fp32 parameters and for matrices trained with Muon (Section 8) it is the parameter itself. The node also keeps an error-feedback residual e per class [25, 47]. Two operators build the packets. TopK_ρ keeps the $\lceil 4096\rho \rceil$ largest-magnitude entries of every block of 4096 values. Q_4 then quantizes the kept values to signed 4-bit integers, using the per-block scale $\max|\cdot|/7$ over the kept entries. For the RDA class, with $g = \|\Delta^{\text{rda}}\|_2$ (and $T_f^{\text{rda}} = 0$ when $g = 0$),

$$T_f^{\text{rda}} = g \cdot \text{Q}_4\Big(\text{TopK}_\rho\Big(\Delta^{\text{rda}}/g\Big)\Big), \qquad \rho = 0.125, \tag{8}$$

and for the averaged class $T_f^{\text{avg}} = \text{Q}_4(\text{TopK}_{0.125}(\Delta^{\text{avg}}))$. SparseLoCo [46] adds the residual to the fresh packet. We send it on a separate debt lane. With the leak coefficient λ defined below and $\tilde{e} = \lambda e$,

$$T_d^c = \text{Q}_4\Big(\text{TopK}_{\rho_c^d}(\omega^c\, \tilde{e}^c)\Big), \qquad \omega^c = \text{clip}\left(\kappa_c\, \gamma(\pi)\, \frac{\|\Delta^c\|}{\|\tilde{e}^c\|},\, 0,\, 1\right), \tag{9}$$

where $\gamma(\pi)$ ramps linearly from 0.15 to 0.5 in the progress fraction $\pi = \min(1, v/63)$, and v is the number of versions of the fragment published before the packet's base. The class constants are $\kappa_{\text{rda}} = 1$ and $\kappa_{\text{avg}} = 0.25$. The debt density is $\rho_c^d = 0.25\rho_c$, except for the dense GDN2 scalar state, where both lanes use density 1. If $\tilde{e}^c = 0$, then $T_d^c = 0$. The node then updates the residual with the decoded values of the bytes it actually sent,

$$e^c \leftarrow \tilde{e}^c + \Delta^c - T_f^c - T_d^c. \tag{10}$$

The error-feedback coefficient is 1. The only decay is a leak. A class gets $\lambda = 0.9$ once the ratio of its residual norm to its fresh-displacement norm has exceeded 3 for 10 consecutive windows; otherwise $\lambda = 1$. When no debt has been accepted for 16 versions and the node's uplink is not throttled, the next debt packet goes out at density $\max(0.5\rho_c, 0.5)$, or 1 for the GDN2 scalar state. For each class present in the fragment, a packet carries the fresh and debt lanes and the norms of the fresh displacement and of the residual. It also carries c_i^{st}, c_i^{tok}, the base version and a digest of the base state.

5.4 Admission and token weighting

Before a packet is admitted, the home checks the base, the sender and the counts. The base version must equal the current head version and the digest must match. The packet must be the first from that node for that version. Its step count must satisfy $1 \le c_i^{\text{st}} \le 2H + \tau = 50$ with $\tau = 2$, and its token count must be consistent with the step count. A packet built on an older version is refused, and the node re-credits the whole packet to its residual. Every admitted packet carries weight

$$w_i = c_i^{\text{tok}}, \tag{11}$$

the number of tokens the node processed in the window. A faster node therefore contributes proportionally more. A node that joins late, or simply processes fewer tokens, contributes less.

Quorum needs at least one expert-owning node among the contributors and at least 3 admitted packets with a nonzero RDA fresh lane. The owner requirement keeps the trunk from advancing while no owner is running and no expert can fold, the same condition under which the coordinator withholds the lease (Section 7). A round closes early once quorum holds, every live member has either sent a packet or passed its expected arrival time, and a short grace period has elapsed. Otherwise the round closes at the deadline $H\,\bar{t}_{\text{step}} + 90\,\text{s}$, where $\bar{t}_{\text{step}}$ is an exponential moving average of the fleet's median step time. The deadline extends to H times the step time of the slowest live member, but we cap that step time at twice the smoothed estimate. If a round closes below quorum, it publishes a no-op version and every contributor re-credits its packet in full. Figure 5 shows the timing of two rounds.

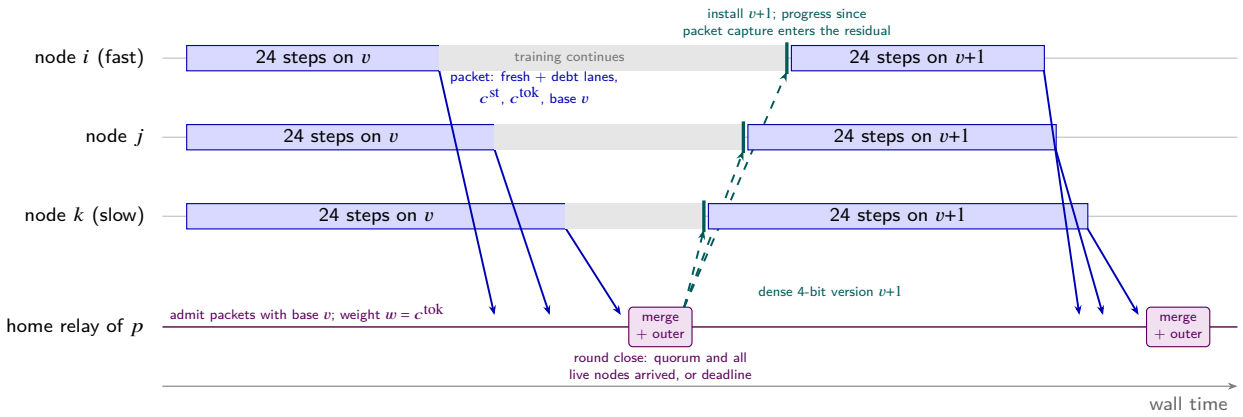


Figure 5. One trunk fragment over two versions. Each node closes a window 24 optimizer steps after installing version v, sends a packet built on v, and continues training. The home relay admits the packets built on v, closes the round, merges them with token weights, applies the outer step and publishes v+1. Each node installs v+1 at its first step boundary after arrival. Progress made since the packet capture enters the node's error-feedback residual. Nodes of different speed send at different times, and none waits for another.

5.5 Merge

Let $\mathcal{A}$ be the admitted packets with a nonzero fresh lane, $q_i = T^{\mathrm{rda}}_{f,i}$ the decoded RDA packets, and $n_i = \|q_i\|$ their norms. From these the home computes four statistics. They are the token-weighted mean $\bar{Q}$ and the weighted mean of unit directions U, together with the consensus of those directions and the weighted mean norm:

$$\bar{Q} = \frac{\sum_{\mathcal{A}} w_i q_i}{\sum_{\mathcal{A}} w_i}, \qquad U = \frac{\sum_{\mathcal{A}} w_i\, q_i/n_i}{\sum_{\mathcal{A}} w_i}, \qquad \mathrm{cons} = \|U\|, \qquad N = \frac{\sum_{\mathcal{A}} w_i n_i}{\sum_{\mathcal{A}} w_i}. \tag{12}$$

The merged fresh update points along the consensus direction. Its norm is bounded both by the mean transmitted norm and by an amplified norm of the mean,

$$F = \frac{U}{\mathrm{cons}}\,\min\!\left(N,\ \chi\,\|\bar{Q}\|\right), \qquad \chi = 1.5, \tag{13}$$

with $F = \bar{Q}$ when $|\mathcal{A}| < 3$ or cons vanishes, and $F = 0$ when $\bar{Q} = 0$. If the packets agree, $\|\bar{Q}\| \approx N$ and $F \approx \bar{Q}$. If they partially cancel, the mean shrinks to $\|\bar{Q}\| < N$, and Equation (13) scales the step along the consensus direction to at most $1.5\|\bar{Q}\|$. The rule follows the regularized dual averaging view of sparse merging [60], where the magnitude the learners transmitted sets the norm of the merged step. When the bound $\chi\|\bar{Q}\|$ is the smaller one, the merged fresh step has 1.5 times the norm of the token-weighted mean and points along the consensus direction. Averaged-class packets are merged as $F = \bar{Q}$. The debt lanes of all admitted packets are averaged with the same weights, $D_d = \sum w_i T_{d,i} / \sum w_i$. Finally, $m = \min(1, \sum_{\mathcal{A}} w_i / W_{\mathrm{full}})$ is the contributing fraction of the nominal fleet, with W_{full} the sum of the members' nominal tokens per window.

5.6 Outer step with coupled momentum and error feedback

The outer step adds an outer momentum to the merged fresh update and the merged debt. We set the weights of these terms per version from the measured state of the fleet. Let π be the progress fraction defined above, evaluated at the version being published. The outer learning rate is $\eta = 0.7$, and ς is a class multiplier ($\varsigma = 1$ for the RDA class, 0.5 for the averaged class). Under the base rule, momentum has to earn its exposure through temporal alignment between the new update and the stored velocity:

$$\mathrm{gate} = \mathrm{clip}\!\left(\frac{\cos(F,u)}{0.5}, 0, 1\right)\ (\mathrm{gate} = 1 \text{ if } u = 0,\ F \neq 0), \qquad \mu = (0.2 + 0.5\pi)\,\mathrm{gate}\,\varsigma, \tag{14}$$

$$u' = \mu\,\tilde{u} + m\,F, \qquad M = \mu\,u', \qquad \tilde{u} = \begin{cases} 0.25\,u & \cos(F,u) < 0 \\ u & \text{otherwise.} \end{cases} \tag{15}$$

When the fleet's error-feedback backlog grows, we throttle momentum. Let $r = \sum w\|e\| / \sum w\, g$ be the weighted ratio of residual to fresh magnitude, and r_0 its baseline. We estimate r_0 per fresh-density stage as the mean of its first 16 finite observations and then track it with an exponential moving average with rate 0.1. The damping $x = \mathrm{clip}((r - r_0)/r_0, 0, 1)$, with $x = 0$ before r_0 is available or when $r_0 = 0$, caps the momentum relative to the fresh update ($\sigma_M = 1$ when $M = 0$):

$$M \leftarrow \sigma_M M, \quad u' \leftarrow \sigma_M u', \qquad \sigma_M = \min\left(1, \frac{(0.25 + 0.5\pi)\, \varsigma\, (1 - x)\, \|F\|}{\|M\|}\right). \tag{16}$$

The accepted fraction of the merged debt has its own bound relative to $\|F\|$, and a second one taken jointly with the momentum:

$$c_{\mathrm{pre}} = \min\left(1, \frac{(0.25 + 0.25\pi)\, \varsigma\, \|F\|}{\|D_d\|}\right) (1 \text{ if } D_d = 0), \qquad c_{\mathrm{debt}} = c_{\mathrm{pre}} \cdot \max\left\{\ell \in [0,1] : \left\|M + \ell\, c_{\mathrm{pre}} D_d\right\| \le (0.5 + 0.5\pi)\, \varsigma\, \|F\|\right\}. \tag{17}$$

Class c applies the update $D^c = \eta(F^c + c^c_{\mathrm{debt}} D^c_d + M^c)$, and the fragment update sums over its classes,

$$D = \sum_c \eta\left(F^c + c^c_{\mathrm{debt}}\, D^c_d + M^c\right). \tag{18}$$

A gate of 0 sets $\mu = 0$, so under the base rule a single round with $\cos(F, u) \le 0$ would reset the velocity to $u' = mF$ rather than decay it.

Error feedback and momentum are coupled in both directions. Momentum is withdrawn as the residual backlog rises. Debt is accepted only inside a budget, and that budget shrinks as momentum takes more of the stale-information allowance. Every cap is proportional to $\|F\|$, so neither memory can dominate the fresh signal. Over the first 64 versions the constants ramp from conservative settings to their final values. The homes read the momentum rule from a runtime control document. In this run it replaces μ in Equation (14) by $0.9\,\varsigma$ with the gate fixed at 1 from the first version, and leaves every cap of Equations (16) and (17) unchanged. Momentum is then exposed whatever the alignment of successive updates, and only the caps limit it.

5.7 Publication and conservation

The home adds the residual left from previous publications to the update and publishes the dense 4-bit quantization of the sum. It installs that into Pub_p in bf16 and keeps the quantization and rounding error as the new residual:

$$\mathrm{blob} = \mathrm{Q}_4\big(D + R_p\big), \qquad \mathrm{Pub}'_p = \mathrm{install}(\mathrm{Pub}_p, \mathrm{blob}), \qquad R'_p = D + R_p - \Big(\mathrm{Pub}'_p - \mathrm{Pub}_p\Big). \tag{19}$$

The version record tells each node whether its packet was accepted, along with the accepted debt fraction c_{debt}. A node whose packet was accepted returns the unaccepted debt to its residual, e += $(1 - c_{\mathrm{debt}})T_d$. After a refused, late or no-op round it returns both lanes, e += $T_f + T_d$. The node then adds the progress it made between packet capture and installation, e += $\theta_i - \theta_i^{\mathrm{capture}}$, to its residual. Only after that does it set its parameters, full-precision masters included, to Pub'_p. The accounting is exact for what a node sends. Apart from the leak and the reset of a rejoining node's residual (Section 7), every unit of local progress is either sent in an accepted packet or held in the node's residual: unsent values, unaccepted debt, refused, late and no-op packets, and the progress made after capture all return to it. The home's publication residual holds the quantization error of each publication. What the home makes of the accepted packets is not conserved. It publishes $\eta(F + c_{\mathrm{debt}} D_d + M)$ with $\eta = 0.7$, not the packets themselves, and nothing credits the difference back. In the averaged class $F = \bar{Q}$. In the RDA class Equation (13) replaces $\bar{Q}$ by a step along the consensus direction U with a bounded norm; that transformation is the regularization of the RDA merge. Algorithm 1 summarizes one round.

Algorithm 1 Trunk synchronization of an RDA/averaged fragment p (node side and home side).

procedure NodeWindow(i, p) ▷ runs after every optimizer step of node i
  $c_i^{\mathrm{st}} \leftarrow c_i^{\mathrm{st}} + 1$; $c_i^{\mathrm{tok}} \leftarrow c_i^{\mathrm{tok}} + 8 \cdot 4096\, a$
  if $c_i^{\mathrm{st}} < H$ **or** a packet or install of p is pending **then**
    return
  $v \leftarrow$ installed version of p; $\theta_i^{\mathrm{cap}} \leftarrow \theta_i$
  for class $c \in \{\mathrm{rda}, \mathrm{avg}\}$ **do**
    $\Delta^c \leftarrow \theta_i^c - \mathrm{Pub}_p^c$; $\tilde{e}^c \leftarrow \lambda_c\, e^c$
    if $c = \mathrm{rda}$ **then**
      $g \leftarrow \|\Delta^c\|$; $T_f^c \leftarrow g\, \mathrm{Q}_4\big(\mathrm{TopK}_{0.125}(\Delta^c / g)\big)$ ▷ Equation (8); $T_f^c = 0$ if $g = 0$
    else
      $T_f^c \leftarrow \mathrm{Q}_4\big(\mathrm{TopK}_{0.125}(\Delta^c)\big)$
    $T_d^c \leftarrow \mathrm{Q}_4\big(\mathrm{TopK}_{\rho_d^c}(\omega^c\, \tilde{e}^c)\big)$ ▷ Equation (9)
    $e^c \leftarrow \tilde{e}^c + \Delta^c - T_f^c - T_d^c$ ▷ decoded values, Equation (10)
  Send$\big(\mathrm{home}(p),\ \{T_f^c, T_d^c, \|\Delta^c\|, \|\tilde{e}^c\|\}_c,\ c_i^{\mathrm{st}},\ c_i^{\mathrm{tok}},\ v,\ \mathrm{digest}(\mathrm{Pub}_p)\big)$

procedure HomeRound(p, v) ▷ on the home relay of p, CPU only
  $\mathcal{A} \leftarrow$ {packets k : $\mathrm{base}_k = v$, digest matches, first from its node, $1 \le c_k^{\mathrm{st}} \le 50$, c_k^{tok} consistent with c_k^{st}}; $w_k \leftarrow c_k^{\mathrm{tok}}$
  wait until quorum, all live nodes arrived or overdue, and grace elapsed; or until the deadline
  if no owner in $\mathcal{A}$ **or** $|\{k \in \mathcal{A} : T_{f,k}^{\mathrm{rda}} \neq 0\}| < 3$ **then**
    Publish($p, v{+}1$, no-op); **return** ▷ every sender re-credits its packet
  $D \leftarrow 0$
  for class $c \in \{\mathrm{rda}, \mathrm{avg}\}$ **do**
    $(F^c, D_d^c, m^c) \leftarrow \mathrm{Merge}^c(\mathcal{A}, w)$ ▷ Equations (12) and (13)
    $(u'^c, M^c, c_{\mathrm{debt}}^c) \leftarrow \mathrm{OuterStep}(F^c, D_d^c, u^c)$ ▷ Equations (14) to (17)
    $D \leftarrow D + \eta\,\big(F^c + c_{\mathrm{debt}}^c D_d^c + M^c\big)$; $u^c \leftarrow u'^c$
  blob $\leftarrow \mathrm{Q}_4(D + R_p)$; $\mathrm{Pub}'_p \leftarrow \mathrm{Install}(\mathrm{Pub}_p, \mathrm{blob})$; $R_p \leftarrow D + R_p - (\mathrm{Pub}'_p - \mathrm{Pub}_p)$ ▷ Equation (19)
  Publish$\big(p, v{+}1, \mathrm{blob}, \{\mathrm{accepted}_k, c_{\mathrm{debt}}\}_k\big)$

procedure NodeInstall(i, p, record) ▷ at the first step boundary after arrival
  if the packet of i was accepted **then**
    $e^c \leftarrow e^c + (1 - c_{\mathrm{debt}}^c)\, T_d^c$ for each class
  else
    $e^c \leftarrow e^c + T_f^c + T_d^c$ for each class
  $e \leftarrow e + (\theta_i - \theta_i^{\mathrm{cap}})$; $\theta_i \leftarrow \mathrm{Pub}'_p$; $c_i^{\mathrm{st}} \leftarrow 0$; $c_i^{\mathrm{tok}} \leftarrow 0$

5.8 Node optimizer state

A node keeps its inner optimizer state, the Muon momentum and the AdamW moments, when it installs a new version. The installed version replaces its parameters, while its momentum continues from its own recent steps. This follows DiLoCo, in which workers keep their inner optimizer state across outer steps [11].

5.9 Router fragment

The router family has its own protocol on its fragment. Once per window of H steps, each node sends a dense fp32 snapshot of the router parameters of all MoE layers. Router windows are anchored to the node's previous capture and have no initial offset. The home admits snapshots built on the current version or the one before it. Each round closes at its deadline. If at least half of the live nodes have contributed by then, the home publishes the equal-weight mean of the snapshots in fp32. Below that quorum the round closes without publishing a version. This lane has no outer learning rate and no momentum. It uses neither error feedback nor quantization.

5.10 Communication volume

Per window of an RDA/averaged fragment, a node sends the fresh lanes at density 0.125 in both classes. Debt lanes go at a quarter of those densities. A forced debt packet uses 0.5, and the GDN2 scalar state is sent dense. Every lane costs 4 bits per kept value plus block scales and sparse indices. In the other direction the node receives one dense 4-bit version for each update round. A no-op round carries no parameter data. The router fragment moves dense fp32 snapshots both ways. Section 10 reports the measured per-node traffic.

6 Expert Lane

The expert lane trains the 4,096 routed experts (32 MoE layers × 128 experts). Every expert has exactly one owner, a single GPU that holds the expert's full-precision master and is the only writer of its published weights. The owner computes no gradients of the master. Every GPU trains a low-rank adapter on each expert it routes tokens to, and each node periodically sends its adapter to the owner as evidence. In a *fold*, the owner averages the evidence of the nodes that contributed in the fold window, its own node always among them, takes one orthogonalized momentum step on the master built from that average and the averages of earlier folds, re-quantizes the expert and publishes a new version to all nodes. The structure follows ReLoRA [29], in which low-rank updates merged periodically add up to a high-rank change. Every change of an expert's master is a fold step built from the adapters of the contributing nodes.

6.1 Ownership

The 16 expert-owning nodes contribute 16×8 = 128 owner GPUs. Expert j of layer ℓ belongs to owner GPU $j \bmod 128$. Each owner GPU therefore owns the experts with the same index in all 32 layers (32 experts), and each owner node owns 256. The remaining nodes own no experts. When an owner fails, its assignments can be remapped (Section 7). For each owned expert, the owner keeps in host memory an fp32 master M_e and an fp32 momentum buffer per projection. All other GPUs hold only the published base weights of Section 4.4.

6.2 Learner adapters

Every GPU trains an adapter on every expert it routes tokens to, whether or not it owns that expert. In the forward pass a routed expert projection uses $W + s(BA - B_0A_0)$ (Section 4.4), where the rank is $r = 16$, $s = 4$, and (B_0, A_0) is the adapter state at the expert's last fold. Adapters start with a nonzero product (both factors uniform, with variance matched to fan-in and a balancing rescale), so the difference $BA - B_0A_0$ starts at zero. We train them with AdamW and the LoRA-Pro gradient correction [58], which adjusts the factor gradients so that the product follows the full-rank gradient more closely.

Adapter gradients are averaged across the 8 GPUs of a node at every step. Each node is left with one adapter per expert, and the node is the unit of evidence. For each expert, one GPU of the node sends the node's adapter along with the per-GPU counts of token assignments routed to the expert since that GPU's last anchor. If the expert's owner is on the node, the owner is the sender; otherwise it is a GPU chosen by a hash of the layer and expert index. The adapter factors travel as 4-bit quantized differences with error feedback.

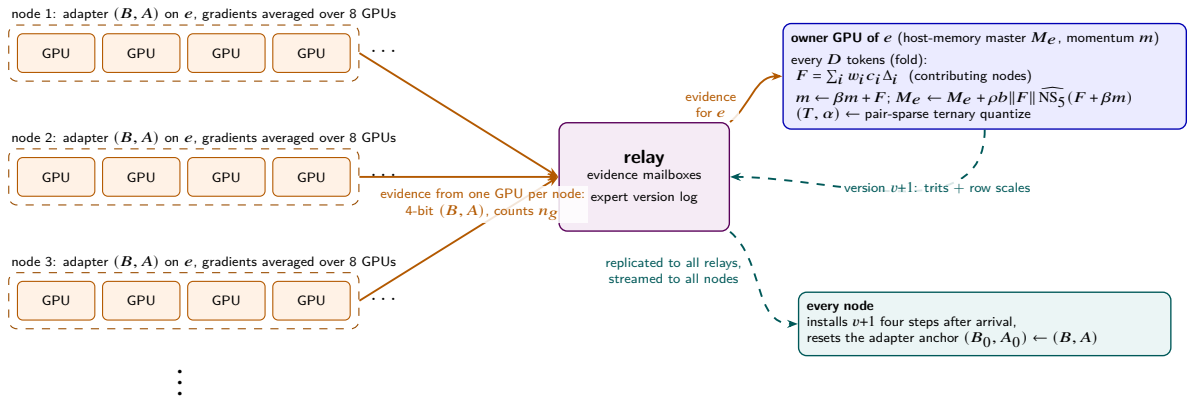


Figure 6. Data flow of the expert lane for one expert e. Each node trains a low-rank adapter on e from the tokens its GPUs route to it and sends the adapter through a relay mailbox as evidence. Once per fold window the owner GPU of e averages the evidence of the contributing nodes, applies one orthogonalized momentum step to the full-precision master, re-quantizes it and publishes a new version, which every node installs.

6.3 Folds

Cadence. Folds are scheduled on the fleet's token clock. The fold window is $D = \max(1.5\times10^9,\ 26\,t_{\text{step}})$ tokens. Here $t_{\text{step}} = 240 \cdot 4096\,a$ is the token count of one optimizer step of a nominal fleet of 240 GPUs at the current accumulation count a. The nominal fleet is larger than the 200 GPUs of this run. At the accumulation counts of Section 8, $26\,t_{\text{step}}$ is 0.26 and 0.51 billion tokens at $a = 10$ and $a = 20$, so $D = 1.5 \times 10^9$. A window spans about 183 fleet steps of 8.19 million tokens at $a = 10$ and about 92 of 16.4 million at $a = 20$. We stagger keys across the window so that folds and their traffic are spread uniformly in time. Key (ℓ, j) has phase $(j + 7\ell)$ mod 76 and folds when the fleet token count crosses its phased boundary.

Fleet mean. When a key folds, its owner collects the node evidence that arrived for the window. Take node i with adapter (A_i, B_i), and let $(\bar{A}_i, \bar{B}_i)$ be the owner's stored copy of that node's previous adapter. The evidence of projection p is

$$\Delta_{i,p} = s\left(B_{i,p}A_{i,p} - \bar{B}_{i,p}\bar{A}_{i,p}\right). \tag{20}$$

We clip each $\Delta_{i,p}$ to three times the median norm, $c_{i,p} = \min(1, 3\,\text{med}_j\|\Delta_{j,p}\|_F/\|\Delta_{i,p}\|_F)$ ($c_{i,p} = 1$ for a zero contribution), with the median taken over contributing GPUs (a node's evidence counts once per member GPU). Clipping is skipped when fewer than 4 GPUs contribute, or when the median is zero or not finite. A node's weight is the sum of the square roots of its member GPUs' assignment counts, $\nu_i = \sum_g \sqrt{n_{i,g}}$ (equal weights if every count is zero), and the fleet mean is

$$F_p = \sum_i w_i\, c_{i,p}\Delta_{i,p}, \qquad w_i = \begin{cases} \nu_i/\sum_j \nu_j & \sum_j \nu_j > 0 \\ 1/n & \text{otherwise,} \end{cases} \tag{21}$$

where the sum runs over all n contributing nodes, the owner's node included. The square-root weights lie between equal weights, which would suit adapters that carry the same information whatever their token count, and assignment weights, which would suit information that grows linearly with tokens. The run contains no measurement that separates these choices.

Fleet step. The owner keeps a momentum buffer m_p per projection and applies the fleet mean through Muon [23, 34] with Nesterov momentum:

$$m_p \leftarrow \beta m_p + F_p, \qquad O_p = \text{NS}_5\left(F_p + \beta m_p\right), \qquad S_p = \rho_k\, b\, \|F_p\|_F\, \frac{O_p}{\|O_p\|_F}\ (S_p = 0 \text{ if } O_p = 0), \qquad M_{e,p} \leftarrow M_{e,p} + S_p, \tag{22}$$

with $\beta = 0.8$. NS_5 is five Newton-Schulz iterations with coefficients $(3.4445, -4.7750, 2.0315)$ on the input divided by 1.01 times its Frobenius norm. It moves the nonzero singular values toward one and leaves the singular vectors unchanged. The step therefore has the direction of the orthogonalized Nesterov momentum of successive fleet means and the Frobenius norm $\rho_k b\|F_p\|_F$ of a scaled fleet mean. We call b the balance. The ramp $\rho_k = \min(1, k/5)$ counts the folds k that the key's buffer has absorbed, including the current one, so a new buffer reaches full step size at its

Algorithm 2 Expert lane for expert e with owner GPU o (fold and install).

procedure FOLD(e) ▷ when the token clock crosses the phased boundary of e
  collect evidence $\{(A_i, B_i, \{n_{i,g}\}_g)\}$ from nodes ▷ 4-bit factors with error feedback
  wait for the owner-coverage quorum or the end of the grace period
  if the owner's own evidence is missing **then**
    return ▷ version v and the buffer stand
  for projection $q \in \{\text{up}, \text{down}\}$ **do**
    $\Delta_{i,q} \leftarrow s\,(B_{i,q}A_{i,q} - \bar{B}_{i,q}\bar{A}_{i,q})$ for every contributing node i ▷ Equation (20)
    $\mu_q \leftarrow$ median of $\|\Delta_{j,q}\|_F$ over contributing GPUs (a node's evidence counted once per member GPU)
    $c_{i,q} \leftarrow \min\big(1,\ 3\mu_q/\|\Delta_{i,q}\|_F\big)$ (1 if $\Delta_{i,q} = 0$) if at least 4 GPUs contribute and μ_q is positive and finite, else $c_{i,q} \leftarrow 1$
    $\nu_i \leftarrow \sum_g \sqrt{n_{i,g}}$; $\quad F_q \leftarrow \sum_i w_i c_{i,q} \Delta_{i,q}$ with w_i of Equation (21) ▷ Equation (21); the mean includes o
    if a contributing row is a repair row **then**
      $m_q \leftarrow 0$; $\quad k \leftarrow 0$; $\quad X_q \leftarrow F_q$ ▷ cold start
    else
      $m_q \leftarrow \beta m_q + F_q$; $\quad X_q \leftarrow F_q + \beta m_q$
    $S_q \leftarrow \min(1, \frac{k+1}{5})\, b\, \|F_q\|_F\, \mathrm{NS}_5(X_q)/\|\mathrm{NS}_5(X_q)\|_F$, or 0 if $\mathrm{NS}_5(X_q) = 0$ ▷ Equation (22)
  if S is not finite **then**
    return
  $M_e \leftarrow M_e + S$; $\quad k \leftarrow k + 1$ unless this fold was a cold start; $\quad (T_e, \alpha_e) \leftarrow$ QUANTIZE$(M_e; T_e^-, \alpha_e^-)$ ▷ Equation (6)
  APPEND$\big(\text{log},\ e,\ v{+}1,\ (T_e, \alpha_e),\ \text{checksum}\big)$; $\quad (\bar{A}_i, \bar{B}_i) \leftarrow (A_i, B_i)$ for every i

procedure NODEINSTALL($e, v{+}1$) ▷ on every node, four steps after arrival
  $W_e \leftarrow \mathrm{diag}(\alpha_e)\, T_e$; $\quad (B_0, A_0) \leftarrow (B, A)$ ▷ adapter difference restarts at zero

fifth fold; $5 = 1/(1-\beta)$ is the horizon of the momentum. A repair fold that restarts a buffer runs at the first ramp level and does not advance the count. The two projections are normalized separately. The owner applies no weight decay to the master. Directly after a fold the master equals the value that the fold wrote, so the step of Equation (22) is the only change a master receives.

A fold that admits a repair row (a node whose anchor was reset after an inconsistency) starts the buffer again: the step uses F_p alone, the buffer is set to zero, and the ramp restarts. If the owner's own evidence is missing, the fold is skipped, the previous version stands and the buffer is not advanced. A step that is not finite is also skipped. After the step the owner re-quantizes the master with the pair-sparse ternary quantizer of Section 4.4 and publishes the result as a new version of the expert.

Quorum. A fold waits for an *owner-coverage quorum*, which counts only the nodes that own experts. The fold proceeds as soon as every live owner node has sent evidence for the window. After a grace period of 157 million tokens on the fleet clock it proceeds once 75% of the live owner nodes have. Once a key's evidence is two fold windows old, evidence from two nodes suffices. Learner evidence is optional. It is folded when it has arrived and is admissible, and it never delays a fold. Every owner-node row that the fold takes must be admissible; a row built on the wrong base refuses the fold and schedules a repair keyframe.

Publication. A version is a checksummed frame of trits and row scales. It is either a patch against the previous version or a full keyframe. The version is appended to the expert version log on the owner's home relay and replicated between relays in per-key order. Nodes receive new versions in the background. Each node installs a version at a step boundary four steps after it arrives, so all GPUs of the node change expert weights at the same step. A node that receives a version late keeps training on its previous one. Algorithm 2 summarizes a fold, and Figure 6 shows the data flow.

6.4 Fold settings

The fold rule is read from a versioned control document on each owner before every fold, so it can change without a restart. In this run every fold from the first one uses the Muon step of Equation (22) with $\beta = 0.8$ and balance $b = 6$, and the rule has not changed. The hysteresis margin is $h = 0.0625$ and the row-scale refit bound is 1%.

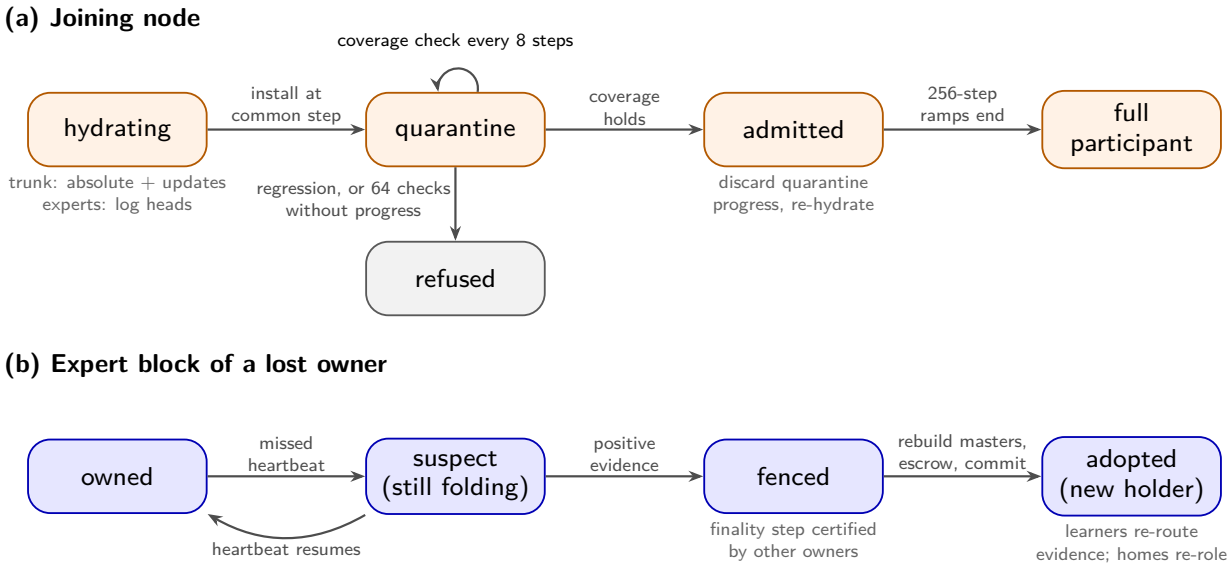


Figure 7. Membership state machines. (a) A joining node hydrates from relay state, trains in quarantine without contributing until every lane is current, and ramps its contribution after admission. (b) An owner that misses heartbeats is marked suspect and keeps folding while alive; once its loss is confirmed, its expert block is fenced at a certified finality step and adopted by a running learner that rebuilds the masters from the published experts.

6.5 Rank coverage under LatentMoE

The current state of a rank-r adapter acts within an r-dimensional input subspace, the row space of its factor A. For the up projection of a LatentMoE expert, $A \in \mathbb{R}^{r\times 384}$ acts on the 384-dimensional latent (Section 4). The current adapters of 25 nodes with $r = 16$ can therefore span all 384 input directions, since $25 \cdot 16 = 400$. Had the expert read the 1152-dimensional residual at the same width, the same adapters would span at most 400 of 1152 directions, or 35%. A fold step is bounded differently. The evidence of one fold, $BA - \bar{B}\bar{A}$, is a difference of two rank-r products and has rank at most $2r$. The step of Equation (22) is an orthogonalized function of the momentum, which accumulates the fleet means of earlier folds, so its rank is bounded by the width of the matrix. The product of the number of nodes and r does not bound it.

7 Membership, Fault Tolerance and Checkpoint-Free State

The published model is kept in services outside the training nodes. The canonical trunk is held by its home relays (Section 5), and the canonical ternary weights of every expert by the expert version log on the relays (Section 6). An indexer authority holds the parameters of the sparse-attention indexer (Section 5). A node that starts, crashes or rejoins rebuilds the published model from these three sources. Some state lives only on nodes: optimizer moments, error-feedback residuals and the owners' full-precision expert masters, among others. On rejoin that state is either reset or reconstructed with a loss of optimizer history (Table 5). The fleet of this run started with 25 nodes, and 15 further node identities are reserved for later joins. No node has joined or left the run in the portion reported here.

7.1 Coordination

A coordinator process issues a short-lived *lease* to the nodes of the run, and a node trains only while it holds a fresh lease. The lease does not depend on any single owner. The coordinator withholds it when no expert owner is live or when it cannot validate the current assignment. A node whose lease delivery fails stops training when its lease expires. Training data comes from a data authority (Section 8), and the relays hold the record of which node holds which block of experts. The coordinator communicates with training nodes through leases and through records on the relays. It starts a training process only to boot a designated cold standby host as an owner. The one case in which it stops a training process is fencing a stalled owner (Algorithm 4).

Algorithm 3 A node joins the run as a learner.

Input: identity n (a reserved one for a new node), launch documents, current lease, run seed

procedure JOIN(n)
  ASSERT(configuration equals a member's, up to identity and the join settings)
  $\theta \leftarrow$ INITFROMSEED() ▷ computed locally; no weights are transferred
  for fragment $p = 0, \ldots, 23$ **do** ▷ trunk hydration from the home relay
    $(v_a, \mathrm{Pub}_p^{(a)}) \leftarrow$ newest replicated absolute of p
    replay committed updates $v_a{+}1, \ldots, v_h$ onto $\mathrm{Pub}_p^{(a)}$, checking the digest chain; stage Pub_p
  in the background: stage the head version of every expert (checksum verified) and the current indexer
  at a common step on all 8 GPUs: install staged state; $e \leftarrow 0$; reset optimizer moments; re-warm 24 steps
  quarantine: train; send no trunk packets, expert evidence, tree-plane updates or indexer evidence
  learning rate from here on: $\eta(s,T) \cdot \big(0.05 + 0.95 \min(1, (s - S)/256)\big)$ ▷ S: join step
  repeat ▷ every 8 steps
    $ok \leftarrow$ every fragment within 2 versions of its head $\wedge$ every routed expert at its head (or never published) $\wedge$ newest stamped expert version $\leq$ 512 steps old (if any version carries a step stamp) $\wedge \geq 1$ tree-plane round applied $\wedge$ indexer $\leq$ 128 steps old
    if a staged install regressed **or** 64 checks passed without progress **then**
      return REFUSED
  until ok
  re-hydrate the trunk to its current version; $e \leftarrow 0$; reset moments, tree planes and controller state; re-warm 24 steps
  for $t = 0, 1, \ldots$ **do** ▷ admitted
    send trunk packets scaled by $0.1 + 0.9 \min(1, t/256)$; discard the scaled-off part; send expert evidence

7.2 Joining

A new node joins as a learner under one of the reserved identities, never as an owner. It starts from the deterministic seed initialization and copies the current published model from the relays and the indexer authority. No checkpoint or model file is transferred from another node. Algorithm 3 gives the procedure and Figure 7a the states.

Quarantine comes first. The node trains, and it contributes nothing. It sends no trunk packets, expert evidence, tree-plane updates or indexer evidence, and the trunk homes do not count it as live. It still exchanges admission, lease and data messages. Admission takes positive evidence that the node's state is current in every lane. Elapsed time alone never admits it. Whatever progress the node made in quarantine is discarded at admission. It re-hydrates the trunk to the current version, resets its error-feedback residuals, optimizer moments, tree planes and controller state, and re-warms its trunk learning rate over 24 steps.

Two ramps follow. The learning rate ramps from 5% to 100% over 256 steps counted from the join step, so this ramp is already under way during quarantine. Trunk packets are scaled from 0.1 to 1 over 256 steps counted from the admission step. The part of a packet removed by this scaling is discarded and does not enter the error-feedback residual. The node's tokens count toward trunk merge weights (Equation (11)) but not toward the nominal fleet size. Once admitted, it takes part in trunk rounds like any other live node, while its expert evidence remains optional for folds.

The node draws training data from the same global shard sequence as every member, starting at the join step. Data it consumes during quarantine is not served again.

A crashed node rejoins through the same procedure, under a fresh reserved identity that records the identity it replaces. The only exception is an owner whose departure all relays recorded as cooperative; such an owner re-enters under its own identity. An identity is never assigned to a different node.

7.3 Owner failover

Ownership moves between nodes as a block, the 256 experts owned by the 8 GPUs of an owner node. A fold requires the owner's own evidence (Section 6), so a dead owner node would stop all folds of its block. The rest of the model keeps training. Learners go on using the last published versions of those experts, and trunk rounds close without the missing node. Algorithm 4 replaces the owner (Figure 7b).

Algorithm 4 Replacement of a lost expert owner (coordinator and adopter).

procedure SUPERVISE(b) ▷ coordinator, every tick, for the owner block b held by node h
  if h missed a heartbeat **then**
    mark b SUSPECT ▷ bookkeeping only; a live owner keeps folding
  if (heartbeats missing $>$ 3 leases $\wedge$ (probe finds no process $\vee$ unreachable for 3 further leases)) $\vee$ (no progress for 600 s $\wedge$ median owner +24 steps) **then**
    $C \leftarrow$ finality step: newest coordinated snapshot, else step certified by a majority of other live owners, else 0 if the block's assignment activated at step 0; if none applies, wait
    fence (b, h) at step C on every relay; stop h's process if the evidence is a stall
    $\mathcal{E} \leftarrow$ {running plan learners and designated standby hosts: expert-log lag $\leq$ 600 s, step lag $\leq$ 256, no quarantined expert unless all warm candidates have it}
    $a \leftarrow$ best of $\mathcal{E}$ by (coverage of b, flagged standby first, experts behind head, log lag, progress, step time); cold standby hosts last
    if $\mathcal{E} = \emptyset$: $a \leftarrow$ best-ranked present candidate that failed only a readiness limit (log lag, step lag or quarantine)
    write promotion intent for (b, a) with a new writer term

procedure ADOPT(a, b) ▷ on the adopter, without a restart
  fence $(b,$ old term) on every relay; freeze capture, evidence, folds and activation of b
  $\{T_e, \alpha_e\}_{e \in b} \leftarrow$ resident copies; verify against the relays' catalog and a peer attestation
  $M_e \leftarrow \mathrm{diag}(\alpha_e)\, T_e$; $m_e \leftarrow 0$; $k_e \leftarrow 0$ ▷ momentum of the lost owner is gone
  reset adapters, residuals and pending evidence of b; persist adoption keyframes at a relay quorum
  if COMPAREANDSWAP(assignment of b: $h \rightarrow a$) fails **then**
    discard staging; **return** ▷ coordinator selects another adopter at a higher term
  activate: nodes route evidence of b to a; trunk homes count a as an owner at their next poll

The coordinator declares an owner lost only on positive evidence. After heartbeats have been missing for three lease periods it probes the host. A probe that finds no running training process is one such case. A host that stays unreachable for three further lease periods is another. The last, which applies whether or not heartbeats arrive, is a live host with no progress for 600 s while the median of the other owners advanced by at least 24 steps. Until one of these holds, a live owner keeps folding. The lease continues to be issued while any owner is live, so the surviving nodes do not wait for the lost one.

The adopting node is normally a learner that is already running and was a member at launch. Nodes that joined later are not eligible, and designated cold standby hosts are ranked last. At launch, the run is refused unless at least two eligible adopters exist. Every learner holds a copy of every expert, namely the published ternary weights it trains on, so the adopter can reconstruct the owner's masters from those weights and continue without a restart. The handover leaves the published experts unchanged, but the masters are only approximated. Reconstruction sets $M_e = \mathrm{diag}(\alpha_e)\, T_e$, so every master weight whose trit is zero becomes exactly zero. By construction that is at least half of the block's weights. Every other master weight is replaced by $\pm\alpha$ of its row. The dead owner's momentum buffers are lost, along with the learner evidence not yet folded and the folds that would have occurred between the loss and the activation. The adopter starts every adopted key with an empty buffer, so the ramp of Equation (22) restarts. The relays hold the published trunk and its outer-optimizer state, so neither is affected. As for any departing node, the lost node's unsent trunk contribution and its residual are lost.

Departures in this run. No node has left the run so far, so Algorithm 4 has not run.

Departure. A node leaves without a protocol. The grants the data authority made to it are lost, which bounds the loss to about 1.25 shards of 5,760 sequences. A trunk home removes a silent node from the live set after two round periods. Until then, a round waits at most for the slowest live node's expected arrival plus a margin, and rounds close on the nodes present. A round of a sparse or averaged fragment without at least one owner and three nonzero contributions of the sparse class publishes a no-op version and returns its packets to the senders' residuals. A router round below its own quorum closes without publishing (Section 5.9). A learner's expert evidence is optional for every fold, and a missing owner node triggers Algorithm 4.

Table 5. Location of training state.

State	Location	Survives a node loss
Trunk weights	home relay of each fragment; checkpoint every 8 versions on at least 2 relays, updates since the checkpoint retained	yes
Trunk outer velocity, publication residual	home relay, in the same replicated checkpoints	yes
Expert weights (trits, row scales)	expert version log, replicated between relays; snapshot every 8 versions	yes
Sparse-attention indexer	indexer authority	yes
Expert masters and fold momentum buffers	owner node, host memory	masters reconstructed from the published trits (zero trits give zero masters), buffers restart empty (Algorithm 4)
Trunk error-feedback residuals	node	reset to zero on rejoin
Adapters	node	reinitialized on rejoin
Data position	data authority (one global shard cursor)	yes; data assigned to a lost node is not re-served

Algorithm 5 Relay-side assembly of an inference model from canonical state.

procedure ASSEMBLE ▷ CPU process on a relay; contacts no training node
  for fragment $p = 0, \ldots, 23$ **do**
    repeat ▷ at most 16 head advances
      $(v_h, d_h) \leftarrow$ head version and digest of p; $(v_a, \mathrm{Pub}_p^{(a)}) \leftarrow$ newest absolute with $v_a \le v_h$
      $\mathrm{Pub}_p \leftarrow$ replay records $v_a{+}1, \ldots, v_h$ onto $\mathrm{Pub}_p^{(a)}$, verifying each digest
    until the head of p is still (v_h, d_h)
    $V[p] \leftarrow (v_h, d_h)$ ▷ version and digest
  for expert $e = 1, \ldots, 4096$ **do**
    $(T_e, \alpha_e, v_e, k_e) \leftarrow$ head version and checksum from the expert version log; verify; $V[e] \leftarrow (v_e, k_e)$
  $V[\text{indexer}] \leftarrow$ current indexer version; fetch the token clock; write the model with version vector V
  decode every tensor again and compare; copy to the partner relay and verify checksums

7.4 State without node checkpoints

The run has no periodic node checkpoints. A coordinated snapshot of model and optimizer state is written only on operator request. Table 5 lists where each part of the training state lives.

Model assembly. Model artifacts are built on relays (Algorithm 5) by a CPU process. It reads the relays' trunk and expert state, the indexer authority and the token count of the data authority, and it contacts no training node. Assembly runs on two relays. On each of them an assembly becomes due one hour after the start of the last successful one and is retried on failure. Each artifact is then copied to a partner relay and verified there by checksum, with retries.

For each trunk fragment the process takes the newest replicated absolute and replays the committed updates up to the head, checking the digest chain as it goes. It repeats this until the head is stable, allowing at most 16 head advances. For each expert it fetches the head version from the expert version log and verifies its checksum. It then assembles and verifies the model and records the version vector it used. That vector holds the version and digest of every trunk fragment, the version and checksum of every expert, and the indexer version. There is no global barrier, so each artifact is a versioned composite of the heads at assembly time. The artifact is the inference model, with experts in the native pair-sparse ternary format. It does not contain optimizer state and is not used to resume training.

Final artifact. At the end of training the trainers drain and publish a signed terminal intent. The assembly service then seals the trunk and expert lanes, builds and verifies a final artifact, and writes a signed acknowledgment. Until the acknowledgment arrives, owners keep their state in memory.

8 Training Schedule and Data

8.1 Corpus

The corpus is a mixture of nine sources in two phases, all tokenized with the Llama-3 tokenizer [17]. The stationary phase covers the first 899.6 billion tokens and draws from seven sources. Two general pretraining mixtures of filtered web text, text extracted from PDF documents, code and mathematics supply 756 billion of them. A filtered web source supplies 40 billion, two mathematics sources 68 billion, a source of encyclopedic and synthetic knowledge text 26 billion, and public-domain books 10 billion. The anneal phase draws the remaining tokens from two further sources of 45.8 and 27.5 billion tokens. It falls inside the learning-rate cool-down. The books and the two anneal sources are used in full. Of the others we use fixed fractions of their shards, close to all of the mathematics and knowledge sources and of the second general mixture, 84% of the first general mixture and 31% of the filtered web source. The run makes one pass over 972.9 billion tokens. The mixture keeps paragraph breaks, so runs of newlines occur in the training text. We did not decontaminate the corpus against the evaluation benchmarks.

8.2 Data assignment

A training sequence is a window of 4097 consecutive tokens of one source file, taken at a stride of 4096. Its first 4096 tokens are the inputs and its last 4096 the targets, and sequences cross document boundaries. Sequences are grouped into shards of 5,760 sequences (23.6 million tokens), and a fixed random permutation orders the shards of all sources of a phase. A data authority holds a single cursor over that permutation. Within a node, the 8 GPUs read disjoint slices of 720 sequences of a shard, each in a pseudo-random order. When its GPUs have consumed 75% of the current shard, or earlier when only four stream blocks of it remain, the node requests its next shard and prefetches it. The authority hands out each shard once, in permutation order, to whichever node asks next. The assignment therefore adapts to node speed without any schedule, because a faster node requests more shards. Shards granted to a node that fails before consuming them are not re-issued, which bounds the data lost per failure to about 1.25 shards (Section 7). When the cursor reaches the end of the stationary phase, the authority continues with the permutation of the anneal phase without a restart. Nodes hold no copy of the corpus. They read sequences from the source files over HTTP byte ranges, in blocks of 2^{18} tokens, with a block cache.

The run's *token clock* is the number of tokens granted by the authority. It counts shards in flight and shards lost to failures, so it is at least the number of tokens consumed. Each node reads the clock from the highest grant it has seen, so the readings of different nodes can differ by the grants still in flight. The learning-rate schedule and the expert fold cadence (Section 6) are functions of it. Warm-up and the architecture switches below are functions of the node's own step count. An exported model records the clock at the time it was cut; this *export clock* lags the fleet's reading by a few billion tokens, and we give benchmark and validation results against it.

8.3 Validation data

The validation set holds 2048 windows of 4096 tokens, stratified over all nine sources and held out from training. The validation loss is the mean cross-entropy over its 8,388,608 target positions, and we also report it per source. We compute it on each exported model, away from the training fleet. The anneal sources are part of the set from the start, although training reaches them only at the end of the run.

8.4 Inner optimizer

Each node trains the trunk with a hybrid optimizer. Two-dimensional hidden weight matrices are updated with Muon [23] in the scaled form of Liu et al. [34]. It uses momentum 0.95 with Nesterov and five Newton-Schulz iterations, and the update is scaled by $0.2\sqrt{\max(m, n)}$ for an $m \times n$ matrix. The momentum is stored in 8-bit blocks of 2048 values, and bf16 writes are stochastically rounded.

AdamW [36] with $(\beta_1, \beta_2) = (0.9, 0.999)$ updates the embeddings, routers, norms, block attention residual parameters and other vectors and scalars. It works on fp32 master copies of bf16 parameters and directly on fp32 parameters such as the norm gains. Its optimizer states are 8-bit and block-wise [9] for tensors of at least 4096 elements, and fp32 for smaller ones. Weight decay is 0.1, except for one-dimensional parameters, gates and the token embedding. Gradients are clipped to a global norm of 1 before the step.

A few parameter groups are handled separately. The indexer authority updates the sparse-attention indexer on its own lane (Section 5). The router weights use a learning rate of 3×10^{-4} scaled by the same schedule, and the GDN2

projections of the first layer use half the learning rate. The output logit scale is clamped to its bound after every optimizer step (Section 4). The routed experts change only at folds, when their owners apply an orthogonalized momentum step built from the mean of the nodes' adapters (Section 6). Each node keeps its optimizer state when it installs a new trunk version.

8.5 Loss computation

The loss is computed by a fused linear cross-entropy kernel that never materializes the full logit matrix. Its stock form rounds each logit to bf16 before the log-sum-exp. The bf16 spacing is 0.25 below 64 and 1 to 2 between 128 and 512, so the rounding error grows with the size of the logits. In a synthetic test with fp64 references, at logit scale 12.6 the rounding biased the loss by 0.036 nats and put a 7.6% relative error on the gradient of the tied weights. We use a copy of the kernel that keeps the logits in fp32. It has the same speed and memory use, and in the same test its gradient error was 0.85% at every scale from 1 to 12, with no measurable loss bias. The gradient with respect to the hidden states is also accumulated in fp32, at a cost of a few milliseconds per micro-batch.

8.6 Learning rate

The learning rate follows a warmup-stable-decay schedule [18, 21]. With the node's step s, the token clock T, peak $\eta_{\max} = 4 \times 10^{-4}$, $T_a = 826{,}967{,}457{,}792$ and $T_e = 972{,}902{,}891{,}520$ tokens,

$$\eta(s,T) = \eta_{\max}\,\min\!\Big(1, \frac{s}{1065}\Big) \cdot \begin{cases} 1, & T < T_a, \\ 0.1 + 0.9\,(1-\sqrt{u}), \quad u = \min\!\Big(1, \frac{T-T_a}{T_e-T_a}\Big), & T \ge T_a, \end{cases} \tag{23}$$

that is, a linear warm-up over 1065 steps, a constant phase, and a 1-sqrt cool-down over the last 15.0% of the tokens to 10% of the peak. The warm-up counts the node's own steps, 8.72 billion tokens at the launch batch, while the constant phase and the cool-down follow the token clock. The nodes finished their warm-up between 7.9 and 12.9 billion tokens of the clock. The run has not yet reached the cool-down.

8.7 Batch size

Each GPU processes micro-batches of one sequence of 4096 tokens and accumulates gradients over a micro-batches per optimizer step, so a node processes $8 \cdot 4096\,a$ tokens per step. The run started with $a = 10$: 327,680 tokens per node step and 8,192,000 per step of the 25-node fleet. We chose a small batch to take many optimizer steps per token. For a fixed token budget, a step that averages fewer tokens gives more updates, and we expected more updates per token to help while the loss falls quickly, even with noisier gradients. We did not train the same schedule with a larger batch from the start. The cost of a small batch is throughput. Each step pays for the optimizer and for host-side work, and at a smaller a that cost is shared by fewer micro-batches. The trunk window is a fixed number of steps (Section 5), so a smaller a also sends more trunk packets per token.

At 85 billion tokens of the token clock we doubled the accumulation to $a = 20$ on all nodes through a runtime control, without a restart: 655,360 tokens per node step and 16,384,000 per fleet step. We did not rescale the learning rate. The update of a Muon step in the trunk's hidden matrices is orthogonalized and has a norm that does not depend on the batch size, so each trunk step now carries twice the tokens for the same step size. The expert fold window is defined in tokens and did not change. Section 10.3 reports the effect on throughput and validation loss.

8.8 Step-keyed switches

Three schedule elements count the node's own optimizer steps: the learning-rate warm-up (1065 steps), the dense phase of sparse attention (32,768 steps, Section 4) and the stochastic SWA window. For the SWA window we follow SWAX [6]. The window of all SWA layers is sampled per step from $\{128, 2048\}$ with equal probability, and once a node has completed more than 112,848 steps it is fixed at 2048, the window used at inference. At 20 micro-batches the dense phase of sparse attention ends at about 450 billion tokens of the clock, and the fixed SWA window would begin only after the planned end of the run. Nodes progress at different speeds, so these switches happen at different token counts on different nodes. At 199 billion tokens of the clock, the step counts of the nodes ranged from 13,313 to 20,527.

Table 6. Weight-matrix operations per decoded token at batch size 1. Context-dependent attention and recurrent-state work is listed separately in the text; the GDN2 depthwise convolutions (0.25 million multiplications) and element-wise operations are omitted. Block attention residual projections are counted once per aggregator; scoring each of up to 9 block states raises this term to at most 673,920.

Component	Operations per token
GDN2 projections (18 layers)	153,944,064 MACs
SWA projections (8 layers)	23,592,960 MACs
MSA projections (6 layers)	31,653,888 MACs
MSA indexer projections (6 layers)	5,087,232 MACs
Routers (32 layers, fp32)	4,718,592 MACs
Latent down and up projections (32 layers)	28,311,552 MACs
Shared experts (32 layers)	169,869,312 MACs
Block attention residual projections	74,880 MACs
Output head (tied embedding)	147,898,368 MACs
Dense subtotal D	565,150,848 MACs
Routed-path scalar multiplications M_r	2,064,384
Routed-path signed additions (at the 50% density cap)	at most 339,738,624
Multiply-bearing work $D + M_r$	567,215,232

9 Inference Efficiency

At batch size 1, decoding one token touches 1,245,076,259 active parameters. Of these, 679,477,248 are routed-expert weights: 12 routed experts in each of 32 layers, with 1,769,472 weights per expert. We store them as pair-sparse ternary values with row scales (Section 4.4). A product with one of them needs no multiplication.

9.1 Operation count

Table 6 counts the weight-matrix operations per token. The dense part is D: every projection outside the routed experts. That covers the shared experts and the latent projections of LatentMoE, and also the fp32 routers and the tied output head. Each weight entry there costs one multiply-accumulate (MAC). A ternary row times the input is just a sum of signed inputs over the nonzero entries. Multiplications remain on the routed path in three places. Each row sum is multiplied by its row scale, the relu^2 activations are squared, and each expert output is scaled by its gate weight. Together that is 2,064,384 multiplications per token.

Summing Table 6 gives $D+M_r = 0.567$ billion multiply-bearing operations per token, 45.6% of the active parameter count. The number of additions depends on how dense the experts are. The pair constraint caps the density at 50%, so the routed experts add at most 339.7 million signed additions per token. If an addition is weighted as a fraction c of a MAC, the operation count is at most $D + M_r + c \cdot 339.7 \times 10^6$.

Evaluation runs packed 2-bit expert kernels on the GPU. On processors without low-bit matrix units, the same pair structure supports a table kernel. For an input pair (x_{2k}, x_{2k+1}), the eight nonzero pair values $(t_1, t_2) \in \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$ give eight partial sums $t_1 x_{2k} + t_2 x_{2k+1}$. Those sums are computed once per input and then shared by every output row. An 8-weight group of a row costs one table lookup and one addition per nonzero pair, so such a kernel does at most 169.9 million lookups per token and as many additions. The cost of building each input's table is spread over all rows that read that input.

The table leaves out work that depends on context. At a context of 4096 tokens, SWA scores and values over the 2048-token window cost 33.6 million MACs per token. MSA over at most 16 blocks of 128 tokens costs 50.3 million. The MSA indexer adds 12.6 million and grows linearly with context. GDN2 state updates take 10.6 million regardless of context length. At inference the SWA window is fixed at 2048 (Section 8).

9.2 Memory

Figure 8 compares storage formats for the routed experts. We assume bf16 for every non-routed parameter and fp16 row scales. bf16 experts put the model at 15.63 GB. Ternary values in 4-bit containers bring it to 4.78 GB. The 1.125-bit codec brings it to 2.17 GB, and 1.04 GB of that is the routed experts with their row scales. Reading every active weight once per token at batch size 1, decoding reads 2.49 GB, 1.47 GB and 1.23 GB in the three formats. In the codec

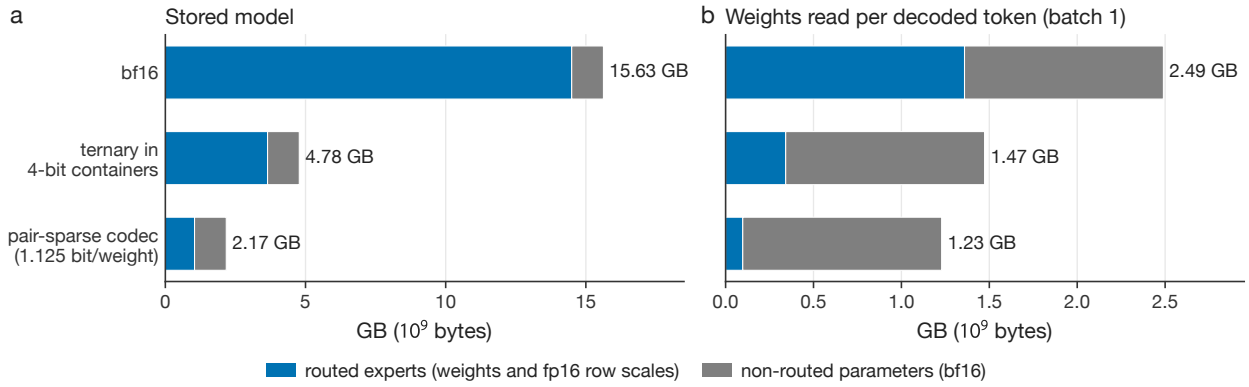


Figure 8. Estimated model size and weight bytes read per token by storage format of the routed experts, with all non-routed parameters in bf16 and fp16 row scales. (a) Stored model. (b) Weight bytes read to decode one token at batch size 1.

format only 0.10 GB of the 1.23 GB comes from the routed experts. Under this estimate the bf16 trunk and output head dominate the weight traffic of decoding.

A uniform distribution over the 417 legal groups would carry $\log_2 417 = 8.70$ bits per group, and the codec spends 9.

Inference state stays bounded. Each of the 8 SWA layers caches keys and values for at most 2048 tokens, with 4 key-value heads of width 64. An MSA layer stores one 256-dimensional latent per token. A GDN2 layer keeps a $9 \times 128 \times 128$ state that does not grow. Past 2048 tokens of context, the caches of the 6 MSA layers are the only ones among the 32 token-mixing layers that keep growing.

10 Results

The run is in progress. We report it as of 29 September 2026, when the fleet's token clock stood at 200 billion tokens, 21% of the planned 972.9 billion. The relays had assembled and the evaluation machines had scored 50 models by then, the newest at 192 billion tokens of the export clock (Section 8). All 25 nodes trained throughout this period. The learning rate is still at its peak, and the anneal data has not been reached.

10.1 Training progress

Figure 9 shows the loss. The training loss is the mean cross-entropy each node computes on its own micro-batches with its local parameters. It starts at 33 nats because of the scaled tied embedding. A node's local parameters are not the model the run produces, so we also score exported models, which contain only the published trunk and experts, on the validation set of Section 8. The validation loss of the exports fell from 5.45 nats at 6.2 billion tokens to 2.70 at 30 billion and 2.176 at 192 billion. The training loss of different nodes spreads widely at any one time, because each node's shards come from different sources and the sources differ in loss by more than a nat (Figure 9c). At 192 billion tokens the mathematics sources scored 1.57 and 1.76 nats, the two general mixtures 2.20 and 2.19, the filtered web source 2.56 and the books 2.80. The anneal sources, which training has not reached, scored 2.85 and 1.37.

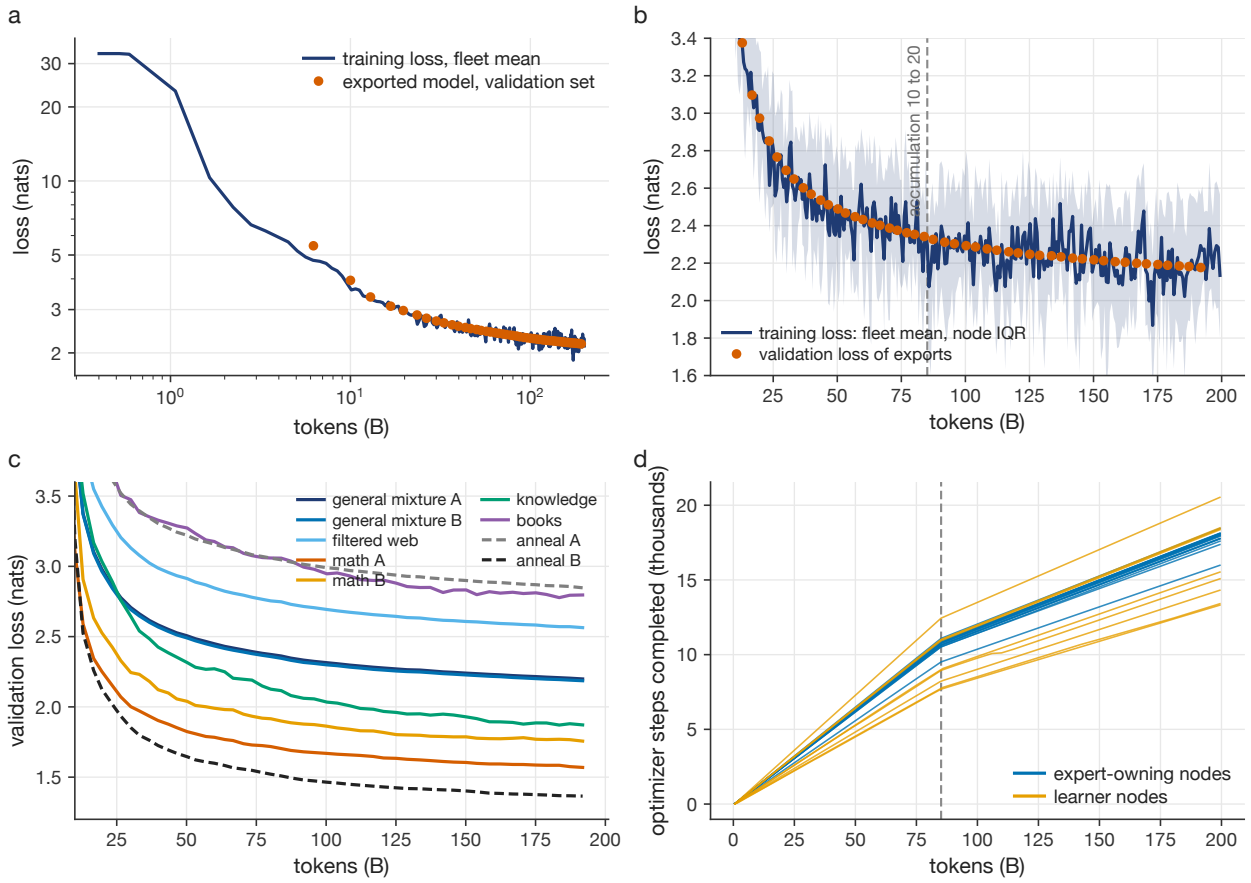


Figure 9. Training progress. (a) Fleet mean of the training loss and validation loss of the exported models on logarithmic axes. (b) From 10B tokens: fleet mean and interquartile range over nodes of the training loss (5-minute means) and validation loss of the exports; the dashed line marks the change from 10 to 20 micro-batches. (c) Validation loss of the exports by source; the anneal sources (dashed) are not yet part of training. (d) Optimizer steps completed by each node against the token clock.

10.2 Benchmarks

We score every export on eleven multiple-choice and cloze tasks, zero-shot and five-shot, on fixed samples of 512 items per task (500 for OpenBookQA and COPA). The headline metric per task is byte-normalized accuracy for ARC, HellaSwag, OpenBookQA, PIQA and SciQ and plain accuracy for the others. The macro is their mean. In the prompts we collapse every run of two or more newlines to one. Our five-shot prompts separate the examples with a blank line, and the tokens for runs of newlines never occur in corpora that strip paragraph breaks. The collapse keeps the prompt format independent of that property of the training data; kappa's own data does contain paragraph breaks. Zero-shot prompts contain blank lines only in LAMBADA. The collapse changes the prompt format, so our five-shot scores are not directly comparable with published results for the same tasks. The scorer's GPU kernels are not deterministic, so two scorings of one export can differ slightly.

Over the first 192 billion tokens the zero-shot macro rose from 34.0% to 53.7% and the five-shot macro from 36.2% to 57.2% (Figure 10). The highest zero-shot macro so far, 54.3% with 57.2% five-shot, came from the export at 187.3 billion tokens; neighbouring exports differ by about a point. Both macros rose fastest in the first 40 billion tokens. Between 80 billion tokens and the newest export they gained 3.1 and 4.3 points. HellaSwag reached 48.6% zero-shot, ARC-Easy 61.3% zero-shot and 67.6% five-shot, and SciQ 92.8% five-shot.

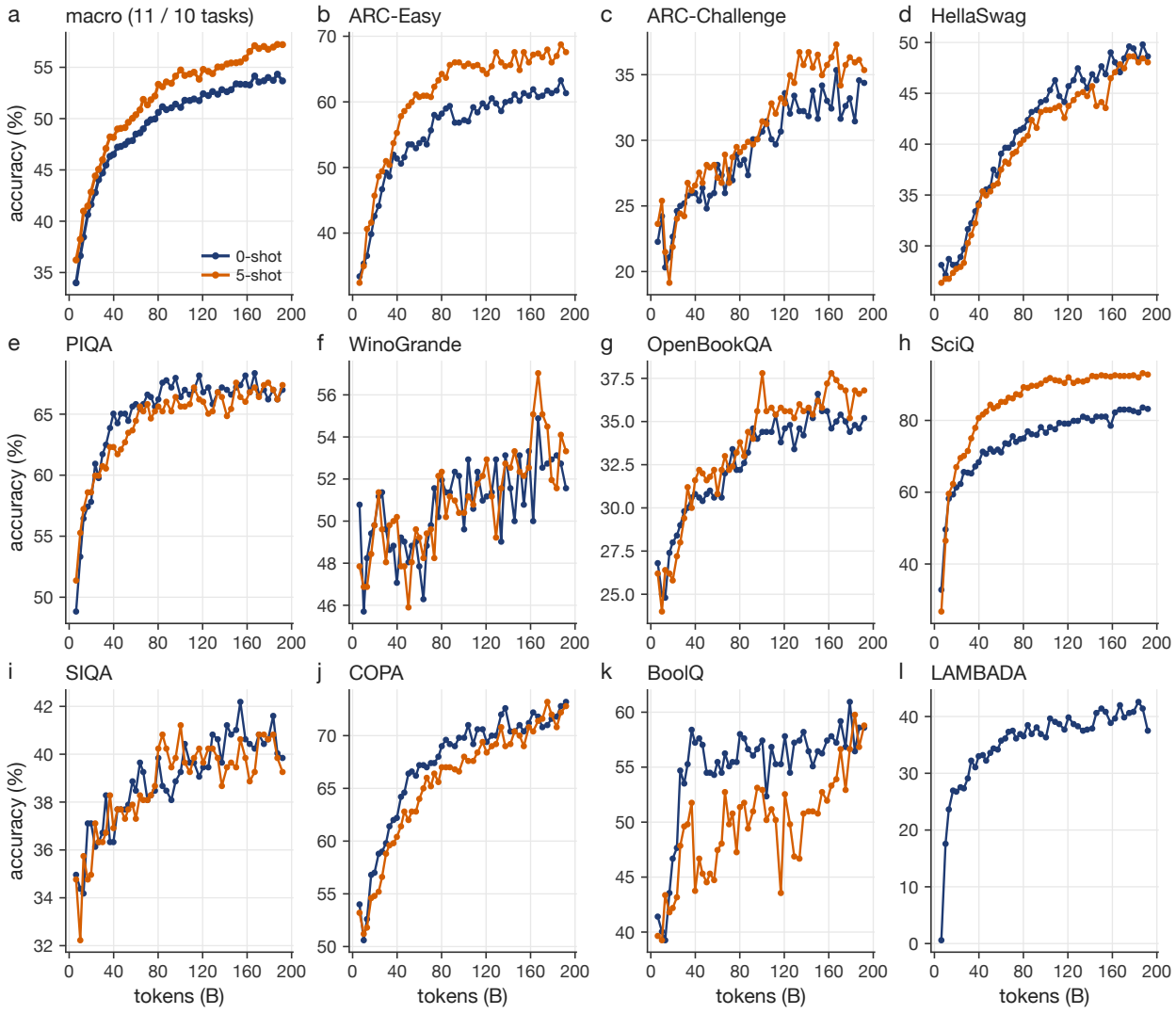


Figure 10. Zero-shot (blue) and five-shot (red) accuracy of every scored export against the export clock. The macro is the mean over the 11 zero-shot and 10 five-shot tasks; LAMBADA is scored zero-shot only. Each task uses 512 items (500 for OpenBookQA and COPA), so one task's standard error is about 2 points.

10.3 Batch doubling during the run

At 85 billion tokens of the token clock we doubled the accumulation from 10 to 20 micro-batches on all nodes (Section 8). Fleet throughput rose from 1.93 to 2.36 million tokens per second, by 22%, averaged over three and a half hours on each side of the change, leaving out the half hour next to it. To judge the effect on the model we fitted $L(T) = \alpha + \beta T^{-\gamma}$, with T in billions of tokens, to the validation loss of the 16 exports saved at 10 micro-batches between 30 and 80 billion tokens of the export clock. The fit gave $\alpha = 2.026$, $\beta = 7.83$ and $\gamma = 0.723$, with a residual standard deviation of 0.0015 nats. The first export saved after the change had trained at most half an hour at 20 micro-batches and lay -0.0015 nats from the fit. Every later export lay below the extrapolated curve, by 0.008 to 0.024 nats, at least 5 residual standard deviations (Figure 11a). The margin has grown with tokens, with some scatter, and was 0.024 nats at 192 billion. So far the larger batch has trained faster with no sign of a higher validation loss per token. This reading rests on extrapolating a three-parameter fit beyond its range, and there is no concurrent run at 10 micro-batches. The margin also depends on the fit range: fitting from 19 billion tokens instead of 30 gives a margin of 0.050 nats for the newest export.

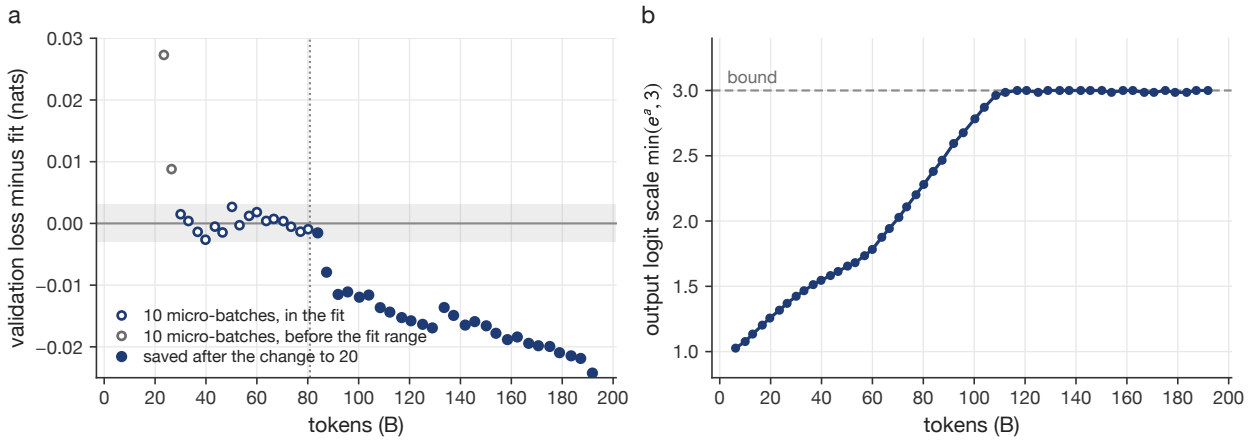


Figure 11. (a) Validation loss of the exports minus a power law fitted to the exports saved at 10 micro-batches between 30B and 80B tokens; the band is ±2 residual standard deviations and the dotted line the export clock at the change to 20. (b) Output logit scale of the exports and its bound.

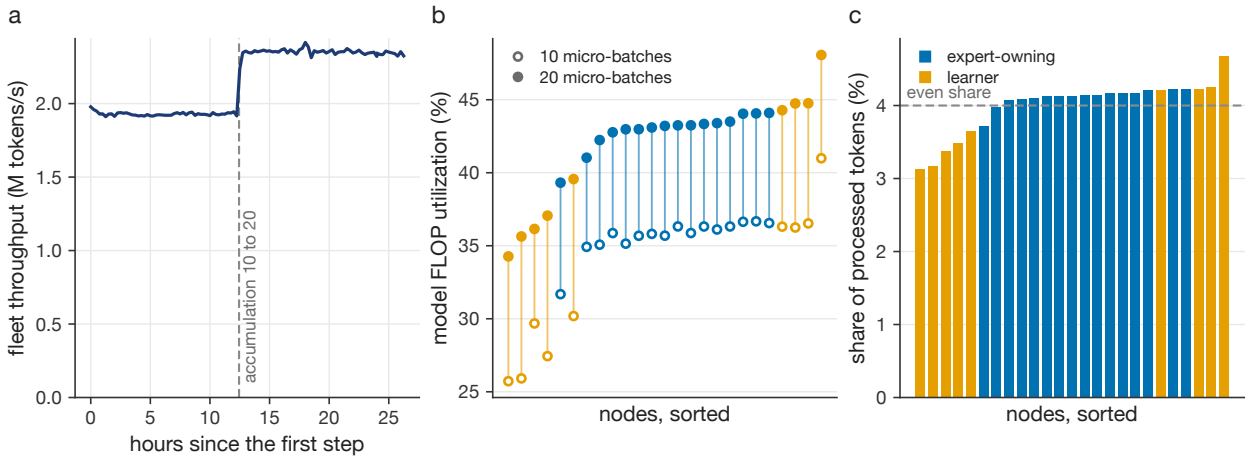


Figure 12. Throughput. (a) Fleet throughput against time since the first step (15-minute means). (b) Model FLOP utilization of each node over three and a half hours on each side of the change from 10 to 20 micro-batches, sorted by the second. (c) Share of all processed tokens by node. Blue: expert-owning nodes; orange: learners.

10.4 Output logit scale

The bound on the output logit scale (Section 4) became active during this period. The scale of the exported models rose steadily from 1.03 at 6 billion tokens to 2.8 at 100 billion and reached the bound of 3 at 112 billion (Figure 11b). Since then every node clamps it after each step. The validation loss shows no break at that point.

10.5 Throughput and utilization

We compute model FLOP utilization as $6N_{\text{active}}$ FLOPs per token over the dense bf16 peak of the node's 8 GPUs, 209.5 TFLOPS each, leaving out attention FLOPs. With 10 micro-batches per step the fleet processed 1.93 million tokens per second at a median utilization of 35.9%. With 20 it processed 2.36 million at 43.2% (Figure 12). The gain comes from the per-step costs, the optimizer step and host-side work, which are now shared by twice as many micro-batches. Per-node throughput at 20 micro-batches ranged from 77 to 108 thousand tokens per second. All nodes have the same GPU; they differ in host CPU, memory and network. Shards are granted on request and trunk packets are weighted by tokens, so a slow node contributes fewer tokens and the others do not wait for it. Up to the snapshot each node processed between 3.1% and 4.7% of all tokens, against an even share of 4%. By the snapshot the nodes had completed between 13,313 and 20,527 optimizer steps (Figure 9d).

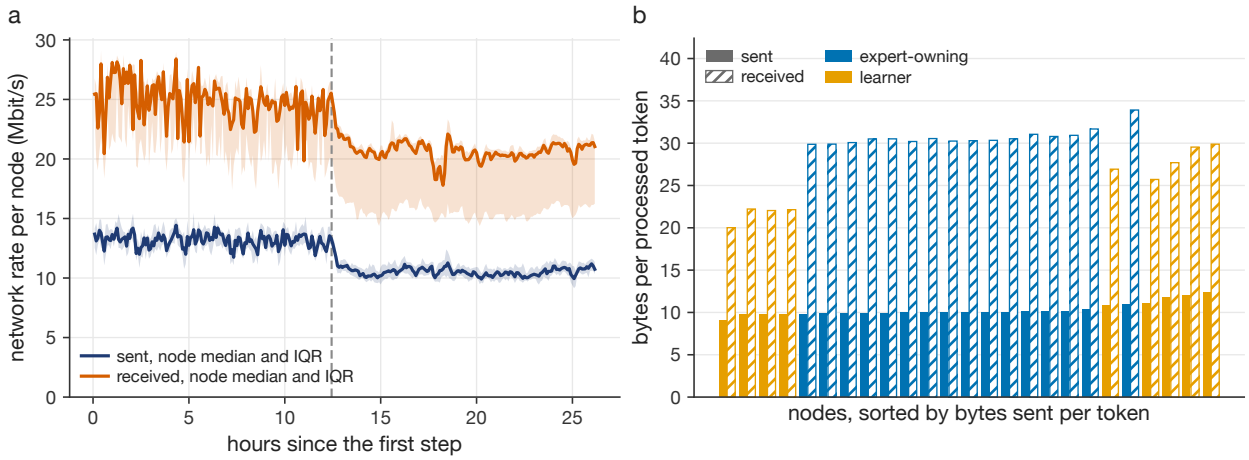


Figure 13. Wide-area traffic of the training nodes. (a) Sent and received rates per node from its network interface, median and interquartile range over nodes; the dashed line marks the change from 10 to 20 micro-batches. (b) Bytes sent and received per processed token, cumulative since the start, per node.

Table 7. Upstream traffic of a node per token it processes. Reference schemes are computed for the full 7.81B-parameter model in bf16 at the batch geometry of each stage; the measured value is the median over nodes of cumulative bytes sent over tokens processed, all lanes included.

Scheme	10 micro-batches	20 micro-batches
All-reduce of bf16 gradients every step	95,378 B	47,689 B
Dense bf16 model every 24 steps	1,987 B	994 B
This run, measured over the whole run so far	10.0 B	

10.6 Communication

A node sent a median of 13.2 Mbit/s at 10 micro-batches and 10.7 Mbit/s at 20, and received 25.1 and 21.2 Mbit/s (Figure 13). Trunk packets made up 93% of the bytes sent and trunk versions 54% of the bytes received. Expert-owning nodes receive more than learners, because the expert evidence of every node arrives at the owners. Doubling the accumulation halved the trunk traffic per token, since trunk windows are counted in steps, and lowered the rate a node sends by about a fifth even though throughput rose. Over the run so far, a node sent a median of 10.0 bytes and received 30.5 (expert-owning) and 25.7 (learner) bytes per token it processed.

Table 7 sets the measured upstream traffic against two analytic reference schemes for the same model and batch geometry. A data-parallel all-reduce of bf16 gradients sends about twice the model size per node and step. Averaging the dense bf16 model every 24 steps, as DiLoCo does, sends the model once per window. The measured value includes every lane.

10.7 Comparison with Agora

The only external run we compare with is Agora [3], a decentralized run that trained a dense 8.6B-parameter model over consumer GPUs. Agora reports its training loss, and Figure 14 sets kappa's training loss against it, as the mean over all nodes in bins of 1 billion tokens of the token clock. Kappa trains on the nine-source mixture of Section 8 and Agora on FineWeb-Edu [41], both with the Llama-3 tokenizer, so the absolute loss levels are not strictly comparable. At 10 billion tokens the two losses were 3.85 (kappa) and 3.82 nats (Agora), at 20 billion 2.95 and 3.27, at 50 billion 2.45 and 2.85, at 100 billion 2.31 and 2.69, and at 190 billion 2.22 and 2.57; kappa's values are means over the bins within 2.5 billion tokens. Counted as $6N_{\text{active}}D$, with all 8.6 billion parameters active in Agora's model, kappa's compute at 190 billion tokens, 1.4×10^{21} FLOPs, equals Agora's at 27.5 billion tokens, where Agora's loss was 3.10 nats. The Agora curve was extracted from the vector figures of its report. Table 8 compares the two systems; its figures for Agora are those of the report.

Kappa's fleet computes about twice the model FLOPs per second of Agora's with far fewer participants, and a kappa node sends about a twentieth of Agora's admission floor. The two numbers measure different things, though. Agora's floor is an admission requirement for any contributor, while ours is the measured median rate of a node with 8 GPUs,

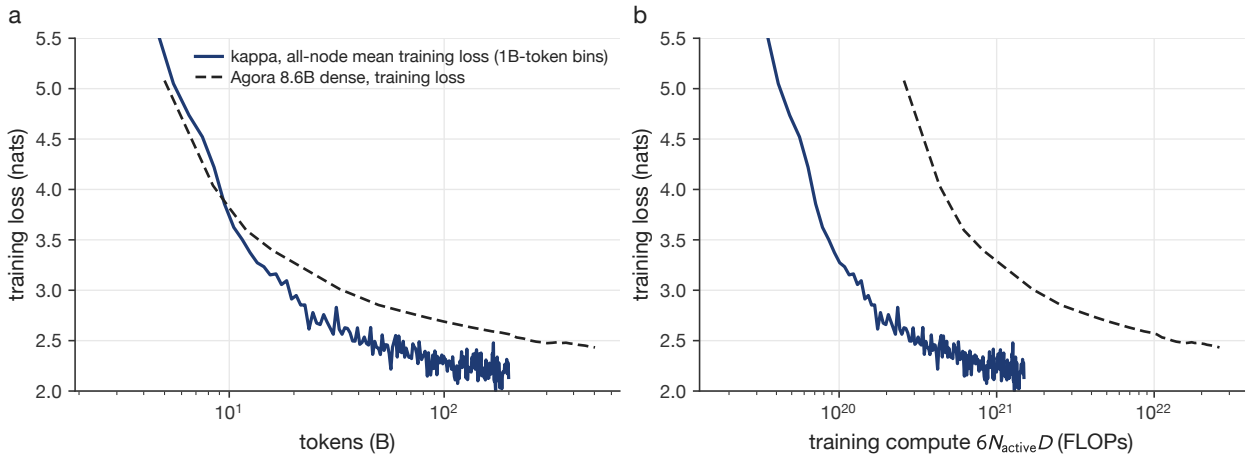


Figure 14. Training loss of kappa and of Agora [3] (a) against tokens and (b) against training compute $6N_{\text{active}}D$, with $N_{\text{active}} = 1.25 \times 10^9$ for kappa and 8.6×10^9 for Agora's dense model. Kappa: mean training loss over all nodes in 1B-token bins of the token clock. Agora: its published training-loss curve. Kappa trains on our nine-source mixture and Agora on FineWeb-Edu, both with the Llama-3 tokenizer, so absolute loss levels are not strictly comparable.

Table 8. Systems comparison with the Agora run. Model FLOP rates count $6N$ per token with N the parameters that are active per token (all of them for the dense model). Agora's figures are from its report [3].

	Agora	**Kappa (this report)**
Model	8.6B dense	7.8B MoE, 1.25B active, ternary experts
Parallelism	7 pipeline stages, no participant holds the full model	data parallel; every node holds the trunk and the published experts
Participants	330 contributors over the run, mostly consumer GPUs	25 nodes of 8 RTX 5090
Corpus	FineWeb-Edu	nine-source mixture
Tokens per optimizer step	about 8.4M	8.19M, then 16.4M
Throughput	about 170,000 tokens/s	2.36 million tokens/s at 20 micro-batches
Model FLOP rate	about 8.8 PFLOP/s	about 18 PFLOP/s
Network per participant	admission floor 200 Mbit/s	median 10.7 Mbit/s sent, 21.2 Mbit/s received per node
Tokens trained	500B	200B so far, 972.9B planned

and a dense model spends about seven times the FLOPs per token of ours.

Table 9. The deployed system compared with the planned dynamic design, which was not used in this run.

Concern	This run	Planned design
Servers	4 fixed relays on a private network	a changing set of servers found through a distributed hash table; trainers only dial out
Authority	one coordinator and one data authority	one replicated log per run (Raft) with a committee of about five voters; a signed, hash-chained record of membership, ownership, data grants and contributions
State placement	static map of 24 fragments to 4 relays	fragments, expert versions, mailboxes and data placed by rendezvous hashing with three replicas; a server joining or leaving changes placement at a fenced cut without restarting trainers
Joining	15 reserved identities, admitted by an operator	any node with a signed ticket, admitted as a learner on probation
Expert owners	16 owner blocks with fixed identities; failover moves a block to a running learner	owner count grows with the fleet; learners with a record of accepted contributions are promoted to owners; ownership is resharded among them
Data	single shard cursor	content-addressed sample tiles replicated across servers; grants recorded in the replicated log
Heterogeneity	token-weighted packets	in addition, contributions from nodes too slow for the round cadence are accepted with an age-dependent weight
Code rollout	sealed release per run	signed packages rolled out to servers one at a time and to trainers in waves

11 Toward a Fully Dynamic Fleet

Much of the run described here is fixed at launch. There are four relays and a fixed map of trunk fragments to them. One coordinator and one data authority serve the whole fleet, and the 16 owner blocks keep fixed identities. Nodes can join, but only into 15 identities reserved at launch. We have designed a successor, and implemented part of it, in which only the trainers are stateless and every service that holds state is replicated over a changing set of servers. None of it was used in this run (Table 9).

The first phase we have implemented is membership without a fixed ceiling. An earlier revision of it passed soak tests in which relay and trainer processes ran on CPUs on a single machine. The tests covered new nodes joining in the middle of a run, and a joined node adopting an expert block. They covered churn, including a relay restart during churn, along with signed heartbeats and survival of the replicated committee when its leader is killed. We have not soak-tested the current revision. No part of the design has run on GPUs or across hosts. The dynamic server set, the tile store, owner promotion and resharding exist only as design. Some elements stay fixed even in the design: a run-creator key and a release key, a bootstrap server for the initial state, an allow-listed committee, and two archive replicas. Trainers are admitted on an honesty assumption. Verification of contributions by spot recomputation, and trusted-execution or Byzantine-tolerant agreement, are left to later phases.

12 Limitations

A run in progress. Every measurement comes from a single run and from its first 21%. The learning-rate cool-down, the anneal data and the fixed SWA window have not been reached, and the newest exports are models at the peak learning rate. We did not train comparison models under the same budget, so we cannot isolate how much individual mechanisms contribute to model quality.

Design choices without controls. The small launch batch, the packet density, the bound on the logit scale and the fp32 loss kernel were set before the run, and the run does not vary them. The reading of the batch doubling rests on a three-parameter fit extrapolated beyond its range, with no concurrent run at 10 micro-batches, and the size of the margin depends on the fit range.

Benchmarks. The lane scores 512 items per task (500 for OpenBookQA and COPA), so a single task has a standard error of about 2 points, and the scorer is not bit-for-bit deterministic. Blank lines in prompts are collapsed, so the

five-shot format differs from that of published evaluations. We did not decontaminate the training data against the benchmarks.

Comparison with other runs. The one external comparison is with Agora, and the two runs also differ in model, parallelism and participants. Agora's network figure is an admission floor while ours is a measured rate.

Fault tolerance. No node has joined or left the run so far, so the joining and owner-failover procedures of Section 7 have not been exercised in it. A failover would rebuild the masters from the published trits, which zeroes every zero-trit weight, at least half of every master, and restarts the fold momentum. The run also depends on single instances of three services, the coordinator, the data authority and the indexer authority, and several corpus sources are served by a single host.

Trust. One organization operated all participants. Nodes are authenticated through the launch configuration. The system does not check that their contributions were computed honestly. Within the admission rules, a malicious node could submit arbitrary packets or evidence.

Measurement. Model FLOP utilization excludes attention FLOPs, and compute counts use $6N_{\text{active}}$ per token without the adapters. Network rates are interface counters and include data streaming. Bytes per token divide a node's cumulative traffic by the tokens it processed, integrated from 5-minute means of its throughput. The inference operation counts and memory figures are analytic, and we do not report decoding throughput on GPUs.

Homogeneous GPUs. All nodes use the same GPU model. Token weighting and on-demand data assignment tolerated the speed differences of this fleet, where the fastest node processed about 1.5 times as many tokens as the slowest. We did not test larger differences in memory or speed.

13 Conclusion

We are training a mixture-of-experts language model on 200 consumer GPUs in 25 hosts that do not connect to each other and synchronize through relays on the public internet, and we report its first 21%. A small dense trunk is merged on the relays from sparse, error-compensated packets, with the routers averaged from dense snapshots. Each routed expert is owned by one GPU, which changes it only at folds, through an orthogonalized momentum step on the weighted mean of the contributing nodes' adapter changes. The run takes many optimizer steps per token, bounds the output logit scale, computes the loss on fp32 logits and trains on a nine-source mixture. After 192 billion tokens its exported model reaches a validation loss of 2.176 nats and a zero-shot benchmark macro of 53.7%. Doubling the batch during the run raised throughput by 22%, and the validation loss has since stayed below the curve fitted before the change. A node sends about 10.0 bytes upstream per token it processes, 10.7 Mbit/s at the current batch, and writes no periodic node checkpoints. The published experts are stored in their inference format. The whole model then needs 0.567 billion multiply-bearing weight-matrix operations per decoded token and an estimated 2.17 GB of storage with the 1.125-bit expert codec, bf16 trunk weights and fp16 row scales.

References

[1] Alham Fikri Aji and Kenneth Heafield. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 440–445, 2017. doi: 10.18653/v1/D17-1045.

[2] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1610.02132.

[3] Gil Avraham, Violetta Shevchenko, Hadi Mohaghegh Dolatabadi, Karol Pajak, James Snewin, Harry Xi, Rodney O'Donnell, Thalaiyasingam Ajanthan, Sameera Ramasinghe, Chamin Hewa Koneputugodage, Shamane Siriwardhana, and Alexander Long. Agora: Collective and permissionless internet-scale pretraining of large language models. *arXiv preprint arXiv:2607.13332*, 2026.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning (ICML)*, 2018. arXiv:1802.04434.

[5] Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, et al. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*, 2022.

[6] Loïc Cabannes, Maximilian Beck, Gergely Szilvasy, Matthijs Douze, Maria Lomeli, Jade Copet, Pierre-Emmanuel Mazaré, Gabriel Synnaeve, and Hervé Jégou. Short window attention enables long-term memorization. *arXiv preprint arXiv:2509.24552*, 2025.

[7] Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, et al. Communication-efficient language model training scales reliably and robustly: Scaling laws for DiLoCo. *arXiv preprint arXiv:2503.09799*, 2025.

[8] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[9] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2110.02861.

[10] Michael Diskin, Alexey Bukhtiyarov, Max Ryabinin, Lucile Saulnier, Quentin Lhoest, Anton Sinitsin, et al. Distributed deep learning in open collaborations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2106.10207.

[11] Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, et al. DiLoCo: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

[12] Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, et al. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

[13] Arthur Douillard, Keith Rush, Yani Donchev, Zachary Charles, Nova Fallen, Ayush Dubey, et al. Decoupled DiLoCo for resilient distributed pre-training. *arXiv preprint arXiv:2604.21428*, 2026.

[14] Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, et al. LatentMoE: Toward optimal accuracy per FLOP and parameter in mixture of experts. *arXiv preprint arXiv:2601.18089*, 2026.

[15] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 2022. arXiv:2101.03961.

[16] Elias Frantar and Dan Alistarh. QMoE: Practical sub-1-bit compression of trillion-parameter models. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024. arXiv:2310.16795.

[17] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[18] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.18392; 1-sqrt cooldown.

[19] Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

[20] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2106.09685.

[21] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024. Warmup-Stable-Decay (WSD) schedule.

[22] Sami Jaghouar, Jack Min Ong, Manveer Basra, Fares Obeid, Jannik Straube, Michael Keiblinger, et al. INTELLECT-1 technical report. *arXiv preprint arXiv:2412.01152*, 2024.

[23] Keller Jordan. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

[24] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with LoRA. *arXiv preprint arXiv:2312.03732*, 2023. rsLoRA.

[25] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *International Conference on Machine Learning (ICML)*, 2019. arXiv:1901.09847.

[26] Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

[27] Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, et al. MiniMax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

[28] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, et al. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[29] Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. ReLoRA: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

[30] Joel Lidin, Amir Sarfi, Erfan Miahi, Quentin Anthony, Shivam Chauhan, Evangelos Pappas, et al. Covenant-72B: Pre-training a 72b LLM with trustless peers over-the-internet. *arXiv preprint arXiv:2603.08163*, 2026.

[31] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, et al. Jamba: A hybrid transformer-Mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[32] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local SGD. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1808.07217.

[33] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations (ICLR)*, 2018. arXiv:1712.01887.

[34] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

[35] Sebastian Loeschcke, Mads Toftrup, Michael J. Kastoryano, Serge Belongie, and Vésteinn Snæbjarnarson. LoQT: Low-rank adapters for quantized pretraining. *arXiv preprint arXiv:2405.16528*, 2024.

[36] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1711.05101.

[37] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, et al. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

[38] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, et al. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[39] NVIDIA. Nemotron 3 Super: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 2026.

[40] NVIDIA, Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, et al. Nemotron-H: A family of accurate and efficient hybrid Mamba-transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

[41] Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024. Introduces FineWeb and FineWeb-Edu.

[42] Bowen Peng, Lizhang Chen, Baiyu Su, Jeffrey Quesnelle, Diederik P. Kingma, and Qiang Liu. DeMo: Decoupled momentum optimization. *arXiv preprint arXiv:2411.19870*, 2024.

[43] Max Ryabinin and Anton Gusev. Towards crowdsourced training of large neural networks using decentralized mixture-of-experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 3659–3672, 2020. arXiv:2002.04013; the hivemind library originates here.

[44] Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. SWARM parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning (ICML)*, 2023. arXiv:2301.11913.

[45] Lorenzo Sani, Zeyu Cao, Meghdad Kurmanji, Alex Iacob, Andrej Jovanovic, Yan Gao, Wanru Zhao, and Nicholas D. Lane. FoMoE: Breaking the full-replica barrier with a federation of MoEs. *arXiv preprint arXiv:2606.19025*, 2026.

[46] Amir Sarfi, Benjamin Thérien, Joel Lidin, and Eugene Belilovsky. Overcoming the communication-performance tradeoff in LLM pretraining. *arXiv preprint arXiv:2508.15706*, 2025. Introduces SparseLoCo.

[47] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Proc. Interspeech*, pages 1058–1062, 2014. doi: 10.21437/Interspeech.2014-274.

[48] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. arXiv:1701.06538.

[49] Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1805.09767.

[50] Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. arXiv:1809.07599.

[51] Jianlin Su. MoE 6 (MoE journey 6: Optimal allocation for balance). Scientific Spaces blog, `https://kexue.fm/archives/11619`, 2026. Proposes Quantile Balancing (auxiliary-loss-free, bias from routing-score quantiles).

[52] Benjamin Thérien, Xiaolong Huang, Aaron Defazio, Irina Rish, and Eugene Belilovsky. MuLoCo: Muon is a practical inner optimizer for DiLoCo. *arXiv preprint arXiv:2505.23725*, 2025.

[53] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1905.13727.

[54] Dustin Wang, Rui-Jie Zhu, Steven Abreu, Yong Shan, Taylor Kergan, Yuqi Pan, et al. A systematic analysis of hybrid linear attention. *arXiv preprint arXiv:2507.06457*, 2025.

[55] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, et al. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

[56] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. SlowMo: Improving communication-efficient distributed SGD with slow momentum. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1910.00643.

[57] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

[58] Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-Pro: Are low-rank adapters properly optimized? In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2407.18242.

[59] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. TernGrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1705.07878.

[60] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

[61] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2412.06464.

[62] Di Zhang, Xun Wu, Shaohan Huang, Yudong Wang, Hanyong Shao, Yingbo Hao, et al. Sparse-BitNet: 1.58-bit LLMs are naturally friendly to semi-structured sparsity. *arXiv preprint arXiv:2603.05168*, 2026.

[63] Jinrui Zhang, Chaodong Xiao, Aoqi Wu, Xindong Zhang, and Lei Zhang. Pretraining a large language model using distributed GPUs: A memory-efficient decentralized paradigm. *arXiv preprint arXiv:2602.11543*, 2026.

[64] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1907.08610.

[65] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2403.03507.

[66] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, et al. Learning N:M fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2102.04010; SR-STE.

[67] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A Benchmark Scores of the Newest Export

Table 10. Accuracy (%) of the export at 192 billion tokens of the export clock on each task, with blank lines in prompts collapsed. Byte-normalized accuracy for ARC, HellaSwag, OpenBookQA, PIQA and SciQ, plain accuracy otherwise; 512 items per task (500 for OpenBookQA and COPA).

Task	0-shot	5-shot
ARC-Easy	61.3	67.6
ARC-Challenge	34.4	35.4
HellaSwag	48.6	48.0
PIQA	67.0	67.4
WinoGrande	51.6	53.3
OpenBookQA	35.2	36.8
SciQ	83.2	92.8
SIQA	39.8	39.3
COPA	73.2	72.8
BoolQ	58.6	58.8
LAMBADA	37.5	–
Macro	53.7	57.2