

ECHO: Error-Compensated 4-Bit Tensor-Parallel Communication for LLM Training and Inference

Xingyu Xie^{1,*} Xiaoyong Xie^{1,*} Zhouchen Lin^{2,†} Kim-Chuan Toh^{1,†}

¹Department of Mathematics, National University of Singapore

²State Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University

ABSTRACT

Tensor parallelism ends every layer with an all-reduce that sits between two dependent matrix multiplications, so its cost cannot be hidden and grows as compute outpaces interconnect bandwidth. Sending 4-bit instead of 16-bit values would cut this cost by nearly $4\times$, but the all-reduce must add the values it carries, and rounding the sum injects an error at every call. Model quality depends on how these errors accumulate over layers, yet existing methods treat each call on its own. We present ECHO, a 4-bit all-reduce for training and inference that controls the accumulated error. ECHO moves data in 4 bits and sums in high precision, so each value is quantized at most twice per call for any number of GPUs. The summation reveals each injected error, and ECHO feeds it back into later calls, so old errors cancel instead of accumulating. We prove that the accumulated error stays bounded independently of depth. On A100 GPUs, ECHO is $2.09\times$ faster than the 16-bit NCCL all-reduce on 1 GB messages. It speeds up the time to first token by up to $1.16\times$ and the training step by up to $1.09\times$ on an idle interconnect, and by up to $1.44\times$ and $1.38\times$ as background traffic grows. It keeps the average accuracy over seven benchmarks within 0.4 points of 16-bit inference on Qwen3-8B. In 10B-token pretraining, its final loss is less than 0.9% above that of 16-bit training. ECHO needs no FP8 or FP4 hardware.

Keywords: Tensor parallelism, collective communication, low-bit quantization, error feedback, large language models.

1 Introduction

Tensor parallelism splits the weight matrices of every attention and MLP layer across several GPUs, and each GPU computes a partial result from its share of the weights (Shoeybi et al., 2019). Turning the partial results into the layer output requires an all-reduce, a collective operation that adds the partial results from all GPUs and returns the sum to each of them. The all-reduce runs between two dependent matrix multiplications, so it is a synchronization barrier on the critical path. Unlike the gradient all-reduce of data parallelism, which runs during the backward pass, or the point-to-point transfers of pipeline parallelism, which are pipelined across microbatches, it cannot be hidden behind the surrounding computation in the same way. Overlapping can hide only part of it (Wang et al., 2024b). In production inference the all-reduce contributes up to 30% of end-to-end latency (Zhao et al., 2025). An 80-layer model issues 160 all-reduce calls in one forward pass, and on GPUs with limited interconnect bandwidth this communication can exceed half of the prefill cost (Li et al., 2024). In large-scale training, tensor-parallel communication takes 50 to 60% of total communication time (Liu et al., 2026). Hardware trends make the problem worse every generation, because peak compute has grown $3.0\times$ every two years while interconnect bandwidth has grown only $1.4\times$ (Gholami et al., 2024). Kernel and decoding optimizations reduce the compute cost of serving (Dao et al., 2022; Xie et al., 2026) and leave the communication untouched. Faster collective algorithms and better use of spare links recover part of the latency but leave the transferred bytes unchanged (Shen et al., 2025; Zhao et al., 2025). Reducing the cost further requires sending fewer bits.

*Equal contribution. [†]Corresponding authors: Zhouchen Lin (zlin@pku.edu.cn) and Kim-Chuan Toh (mattokc@nus.edu.sg).

Sending fewer bits means using a lower-precision format, and lower precision has been adopted by every other part of the LLM stack. Matrix computation has moved from FP16 to FP8 in production-scale training (DeepSeek-AI, 2024) and to FP4 on recent hardware (NVIDIA, 2025), with model quality preserved by careful algorithm design. Model weights are served in 4-bit formats (Frantar et al., 2023; Lin et al., 2024). KV caches are compressed to 2 to 4 bits (Hooper et al., 2024; Liu et al., 2024). Sharded data parallelism communicates weights and gradients at close to 4 bits (Jia et al., 2024), and production mixture-of-experts systems send expert-layer activations at FP8 (DeepSeek-AI, 2024; Jin et al., 2026). Below 8 bits, tensor-parallel communication is the exception. The all-reduce of most deployed systems still transfers 16-bit values, although communication time is proportional to the bits sent and a 4-bit all-reduce would nearly quadruple the effective interconnect bandwidth on unchanged hardware. The reason is that the all-reduce must add the numbers it moves, and a sum of 4-bit values does not fit in 4 bits.

The same constraint appears outside communication. Low-precision matrix multiplication has the same problem, and tensor cores solve it inside the chip. They multiply low-precision operands but accumulate the products in high-precision registers, so the wide sum is never stored and never moved (Micikevicius et al., 2022; NVIDIA, 2025). The all-reduce cannot do the same. The values it adds start on different GPUs and must cross the interconnect before the addition, and the sum must cross the interconnect again to reach every GPU. If both crossings are to stay at 4 bits, the collective must dequantize what arrives, add in high precision, and quantize the sum again for the way back. Every call injects a fresh rounding error into the hidden state. The injections are also frequent. An LLM weight is quantized once, before deployment, and only read afterwards, while the all-reduce quantizes at every layer of every forward pass and, in training, of every backward pass over tens of thousands of steps. The injected errors accumulate, and published results show the consequences. In inference, quantizing tensor-parallel communication to 4 bits without training costs measurable accuracy. On Gemma 2 27B, ARC-challenge accuracy drops from 62.4 to 57.5 and WinoGrande from 79.1 to 74.3, and keeping outlier channels at 16 bits recovers only part of the loss (Dong et al., 2024). In training, naive low-bit communication destabilizes the loss curve or diverges (Liu et al., 2026). The most recent training-side work identifies error accumulation under high-frequency communication as the central obstacle, yet its remedy is a stronger quantizer applied call by call, and it stops at FP8 (Liu et al., 2026). Every existing method works on each call in isolation, but low-bit communication is a dynamic process, a sequence of rounding errors that flows through the network and through the training run. Model quality depends on the accumulated error of this process, which a per-call quantizer does not control.

We control the process rather than the individual calls. We present ECHO, a tensor-parallel all-reduce that sends 4-bit integers, for both inference and training. ECHO first restructures the collective around the constraint that addition must happen in high precision. A quantized all-to-all brings each data segment to one owning GPU, the owner forms the sum in high precision, and a quantized all-gather returns the reduced segments, so each element is quantized at most twice per call (Sec. 4). On top of this primitive, ECHO controls the error process. The reduction step reveals each rounding error exactly, and a two-timescale compensator feeds it back into subsequent calls, so the accumulated error stays bounded instead of growing with the number of calls. The guarantee is independent of network depth and free of the exponential Lipschitz factor of worst-case bounds (Secs. 5 and 6). The name comes from this mechanism, in which each injected error returns to the collective at the calls that follow, like an echo. The compensation is fused into the entry pass of the all-reduce and into an update that runs beside the next matrix multiplication, without changing the computed values (Sec. 5.3). The complete design transports 4-bit integers with half-precision quantization parameters for each group and requires no FP8 or FP4 hardware support.

Our contributions are summarized as follows.

1. A quantized all-reduce primitive in which each element is quantized at most twice per call, independent of the number of GPUs. Computation and communication overlap inside the collective through a four-stage tiled pipeline that runs asynchronously over peer-to-peer direct writes, entirely on the GPUs with no CPU coordination between stages.
2. A two-timescale feedback compensator that measures the rounding error at the reduction step and cancels its accumulation across calls, keeping the total error bounded regardless of the number of calls, with the same mechanism serving inference and training.
3. A theory of the compensated collective. The bounds are independent of network depth and free of the exponential Lipschitz factor of worst-case bounds. The source bound holds for any error sequence on the forward pass and, at every site, in mean square on the backward pass under a smoothness condition. The propagation bound keeps the first-order output deviation, measured against the hidden state, at the level of a single rounding error at every depth.

4. A CUDA implementation that fuses compensation, quantization, and communication into one set of kernels. For 1 GB messages, the compensated 4-bit all-reduce is $2.09\times$ faster than the 16-bit NCCL all-reduce, and the time to first token and the training step improve by up to $1.16\times$ and $1.09\times$ on an idle interconnect and by up to $1.44\times$ and $1.38\times$ as background traffic grows. 8-bit inference matches 16-bit accuracy, and 4-bit inference stays within 0.4 and 1.7 points of it on average over seven tasks for Qwen3-8B and Llama-3-8B. Over 10B tokens of pretraining, 4-bit training with a slightly larger learning rate ends less than 0.9% above the 16-bit loss. To our knowledge, ECHO is the first to run tensor-parallel communication with 4-bit integers for both training and inference.

The code will be released upon acceptance.

2 Related Work

We discuss three directions of related work: methods that hide or accelerate the tensor-parallel all-reduce, methods that quantize it, and error-feedback methods that compensate compression errors in other forms of parallelism. None of them controls the accumulated error of a quantized all-reduce, which is the central object of this paper.

The first direction hides or accelerates the all-reduce without reducing the bits it moves. Domino and FLUX split the computation of a transformer layer into pieces with no mutual dependence, so that one piece computes while another communicates (Chang et al., 2024; Wang et al., 2024b). TokenWeave fuses the all-reduce with the normalization that follows it, using multicast hardware that delivers one write to all GPUs at once (Gond et al., 2026). FlexLink routes part of the traffic over otherwise idle PCIe and RDMA links (Shen et al., 2025). SHARP builds the addition into the network switches, so the data is summed on its way through the network instead of on the GPUs (Graham et al., 2016). SwitchML, which aggregates gradients inside a programmable switch, must first convert them to a fixed-point format with enough headroom to hold the sum (Sapio et al., 2021), the same constraint our collective is built around (Sec. 4). All of these methods transfer 16-bit values. Hiding works only when enough independent computation exists to run behind the communication, and this amount shrinks every hardware generation because peak compute grows faster than interconnect bandwidth (Gholami et al., 2024). Our collective reduces the bytes instead, and a scheduler can treat it as an ordinary all-reduce with less communication to hide.

The second direction quantizes tensor-parallel communication directly and is the closest to our work. For inference, Flash Communication (FlashComm) uses fine-grained group quantization and a two-step all-reduce (Li et al., 2024). FlashComm V2 supports arbitrary bit widths through bit splitting and stores outliers in floating point (Li et al., 2025b). Mixed-precision schemes keep outlier channels at 16 bits and reach an average of 4.2 bits (Dong et al., 2024). Fine-grained quantization of the exchanged activations shortens the time to first token by up to $2\times$ (Hansen-Palmus et al., 2024), and EQuARX builds a block-wise int8 all-reduce into the XLA compiler for TPUs (Ahmed et al., 2025). QuickReduce brings INT4, INT6, and INT8 all-reduce to inference serving on AMD GPUs (Li et al., 2025a). For training, TACO compresses the communicated tensors to FP8 with a Hadamard transform and a two-level scaling scheme (Liu et al., 2026), and MegaScale-MoE replaces the tensor-parallel reduce-scatter of FP8 mixture-of-experts training with an FP8 all-to-all and an FP32 sum (Jin et al., 2026). All of these techniques work within a single call. Finer groups, preserved outliers, rotations, and wider formats reduce the error of one quantization, but no method compensates the errors that earlier calls have already injected, so the accumulated error over layers and steps remains uncontrolled. ECHO controls the accumulated error rather than the per-call error, and since its bounds scale with the per-call error, these techniques can be combined with it. The compensator keeps the accumulation bounded (Sec. 5), so a standard INT4 quantizer is sufficient for both inference and training, and each element is quantized at most twice per call regardless of the number of GPUs (Sec. 4).

The third direction compensates quantization errors in other forms of parallelism. Error feedback is a classical technique from signal processing and entered deep-learning training with 1-bit SGD, which stores the quantization error of the gradient and adds it to the gradient of the next step (Seide et al., 2014). Later analysis proved that this correction removes the bias of aggressive compressors (Karimireddy et al., 2019), and EF21 strengthened the guarantees with a different mechanism that communicates a compressed difference to a reference state (Richtárik et al., 2021). PowerSGD, 1-bit Adam, ZeRO++, and SDP4Bit bring compressed communication to large-scale data-parallel training (Jia et al., 2024; Tang et al., 2021; Vogels et al., 2019; Wang et al., 2024a), and LoCo shows that a moving-average estimate of the compression error, applied before quantization, keeps low-bit gradient communication accurate at scale (Xie et al., 2025). AQ-SGD extends compensation from gradients to the activations at a pipeline-parallel boundary, quantizing the change of each example’s activations since its previous visit (Wang et al., 2022).

Error feedback works in all of these settings because the compressed data is a gradient, or changes slowly over training. The training loop produces a new gradient for the same parameters at every step, so an error left uncorrected at one step can be added to the gradient of the next step, and nothing is lost. A hidden state gives no such opportunity. It is produced once, consumed by the next layer, and never communicated again, and at inference time there is no training loop at all. Accordingly, error feedback applied to the activations of model-parallel training gives no gain over plain compression (Rudakov et al., 2024), and compression of model-parallel activations has been studied empirically across pruning, learned, and quantized schemes (Bian et al., 2023). In tensor parallelism the data does not repeat, but the operation does. The all-reduce runs again at the next layer, and in a pre-normalization transformer all its outputs feed the same residual stream (Sec. 3). ECHO builds its feedback on this repetition. The reduction step exposes every rounding error exactly (Sec. 5), the compensator adds a correction at the following calls, and the corrections cancel the accumulated error along the residual stream, with a bound independent of depth (Sec. 6). To our knowledge, ECHO is the first to place feedback compensation inside a quantized collective for tensor parallelism.

3 Problem Formulation

3.1 The quantized all-reduce and its error

An all-reduce adds the contributions of the P GPUs in the tensor-parallel group, and this addition cannot be done in the low-bit transport format, because a sum of low-bit values overflows its range. The sum is formed in high precision, and the low-bit format only moves data between GPUs, so a quantized all-reduce returns the exact sum plus the rounding error of the data it moved. Section 4 states this constraint precisely and builds the decomposition that keeps the number of roundings as small as possible.

Let $\mathbf{x}_{k,i}$ be the contribution held by GPU i at the k -th all-reduce of a pass, so an exact all-reduce returns $\sum_i \mathbf{x}_{k,i}$. The quantized all-reduce, written QAR, returns

$$\hat{\mathbf{y}}_k = \text{QAR}(\{\mathbf{x}_{k,i}\}_{i=1}^P) = \sum_{i=1}^P \mathbf{x}_{k,i} + \boldsymbol{\delta}_k. \quad (1)$$

The vector $\boldsymbol{\delta}_k$ is the error the quantized all-reduce injects at site k , and it is the only difference between $\hat{\mathbf{y}}_k$ and the exact sum. Its norm is bounded by a constant q , $\|\boldsymbol{\delta}_k\| \leq q$, set by the bit width b , the group size g , and the scale of the data (Sec. 4). Communication volume is proportional to b , so a b -bit all-reduce sends about a fraction $b/16$ of the bytes of a 16-bit all-reduce, the most that lower precision alone can save.

3.2 The communication noise process

A pre-normalization transformer keeps a hidden state and, at each attention and MLP block, normalizes the hidden state, runs the block, and adds the result back (Xiong et al., 2020). In tensor-parallel form each block produces its output in pieces, so GPU i computes a partial output $\mathbf{x}_{k,i}$ and an all-reduce sums the P pieces into the block output. Writing $\mathbf{z}_k$ for the hidden state entering the k -th such block,

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \sum_{i=1}^P \mathbf{x}_{k,i}. \quad (2)$$

Each transformer layer has two of these all-reduces, one for its attention block and one for its MLP block, so a pass has N of them. Qwen3-0.6B, with 28 layers, has $N = 56$.

With the quantized all-reduce the block contributes $\hat{\mathbf{y}}_k = \sum_i \mathbf{x}_{k,i} + \boldsymbol{\delta}_k$ instead, so

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \sum_{i=1}^P \mathbf{x}_{k,i} + \boldsymbol{\delta}_k. \quad (3)$$

The useful block output $\sum_i \mathbf{x}_{k,i}$ is unchanged and the quantized all-reduce only adds the error $\boldsymbol{\delta}_k$. Low-bit tensor-parallel communication is therefore a noise process on the computation graph, injecting an error at each all-reduce site of a pass (Fig. 1). The same process occurs every time the model runs. A single forward pass in inference injects N such errors. In training the same injection happens on the forward and the backward pass of every step, and over a full run it repeats across many steps, so the process acts over far more sites. The point of injection also differs

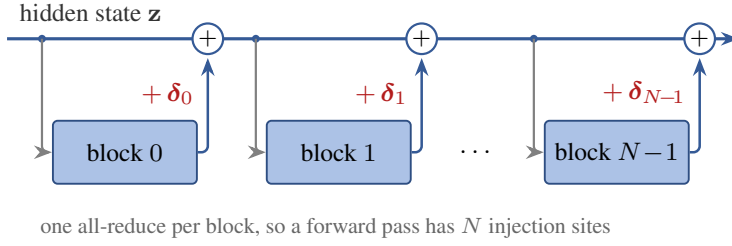


Figure 1. Low-bit tensor-parallel communication as a noise process. A pre-normalization transformer adds each attention and MLP block’s output to the hidden state. In tensor-parallel form the block ends in an all-reduce, and replacing it with the quantized all-reduce leaves the block output in place while adding an error δ_k at that site. A forward pass has N such sites, and the backward pass injects at the same sites (Sec. 5.4).

between the two passes. On the forward pass δ_k joins the hidden state directly, while on the backward pass it enters the gradient of the hidden state after the normalization, so the forward and backward errors are corrected separately (Sec. 5.4).

3.3 Controlling the accumulated error

The output error is not set by any single δ_k . Once injected, an error is carried to the output by the blocks that follow it, which respond to the already-perturbed hidden state and amplify it. The output error is built from the N injections after this amplification. Its size is bounded by two factors. The first is how much the network amplifies a perturbation, which Sec. 6 bounds under one assumption on the trained network. The second is how much the injections reinforce one another as they build up, measured by the largest partial sum of the injected sequence. The amplification is a property of the network, so the factor a communication primitive can change is the partial sum.

The partial sum can be kept from growing because the quantized all-reduce reveals its own error. The residual left on GPU i after the all-reduce, $\mathbf{u}_{k,i} = \mathbf{x}'_{k,i} - \hat{\mathbf{y}}_k/P$, with $\mathbf{x}'_{k,i}$ the (compensated) contribution it sends into the all-reduce (Sec. 5), sums across the GPUs to

$$\sum_{i=1}^P \mathbf{u}_{k,i} = -\delta_k, \quad (4)$$

so the group recovers the error it has just injected, at no extra communication. The primitive feeds this measurement back: before the next quantization it adds a correction $\mathbf{C}_k$, built from the errors measured at earlier sites, to the all-reduce input. Through the skip connection the correction enters the hidden state beside the injection, so the net error added at the site is

$$\rho_k = \delta_k + \mathbf{C}_k. \quad (5)$$

Because $\mathbf{C}_k$ is built from the earlier injected errors, it cancels their buildup in the running sum, so the partial sums stay bounded as the pass proceeds. This is error feedback, and Sec. 5 gives the specific filter.

The factor to control is then the largest partial sum of these net errors. Writing

$$\mathbf{S}_t = \sum_{k \leq t} \rho_k, \quad (6)$$

the requirement on the primitive is that $\max_t \|\mathbf{S}_t\|$ does not grow with the number of all-reduces. Without correction the partial sum is $\sum_{k \leq t} \delta_k$, whose largest value can reach Nq over a forward pass. This growth causes the accuracy loss of a plain 4-bit all-reduce in inference and its instability in training. Since the amplification is shared, inference and training both need a partial-sum bound that stays flat as the number of sites grows. Section 5 builds the correction $\mathbf{C}_k$. Section 6 proves that the partial sum $\max_t \|\mathbf{S}_t\|$ stays bounded independently of N for any error sequence on the forward pass, that its backward counterpart stays bounded in mean square at every site, and that its first-order effect on the output, measured against the hidden state, stays at the level of a single rounding error. Section 7 evaluates the accuracy.

4 Quantized P2P All-Reduce

ECHO replaces the tensor-parallel (TP) all-reduce with a communication primitive that moves data between GPUs in low precision but forms the sum in high precision. Two properties of this design are used throughout the paper. First, the reduction is carried out in high precision and introduces no error of its own, so the all-reduce output differs from the exact sum only because the transported data is quantized. Second, every value passes through the quantizer at

Table 1. Quantized P2P all-reduce against a full-precision ring all-reduce on P GPUs: how many times each element is (re)quantized, the source of the injected noise δ_k , and the bytes on the wire, with b the communication bit width.

	Where the sum is formed	Quant. events per element	Wire format
16-bit ring AR	low or high prec.	0	16-bit
Naive low-bit ring	(invalid in low-bit)	$\mathcal{O}(P)$	b -bit
Ours (P2P)	owner, high prec.	2	b -bit

most twice, regardless of how many GPUs participate, so the number of roundings at each site does not grow with the tensor-parallel degree P . The rest of the section describes an implementation that realizes both properties at low latency.

4.1 A forced decomposition

Turning the lower precision of Sec. 3 into a faster all-reduce runs into a constraint. The all-reduce must add the contributions of the P GPUs, and this addition cannot be carried out in the low-bit transport format. Two 4-bit values can already add up to a number that no longer fits in four bits, so a low-bit accumulator overflows. Even where 4-bit arithmetic is available, four bits are too few to hold a running partial sum without large rounding error, and low-bit accumulation is in any case not widely supported across accelerators. The sum is therefore formed in high precision, and the low-bit format only moves data between GPUs. Carried out naively, a quantized all-reduce becomes a repeated cycle of dequantize, add, and requantize, and each requantization injects one of the errors δ_k of Sec. 3. The rest of this section keeps the number of these requantizations as small as possible.

We first recall the standard structure of an all-reduce. Each GPU, one rank of the TP group, holds a tensor split into P equal segments, one associated with each GPU. A *reduce-scatter* sums the segments across GPUs so that GPU j is left with the sum of the j -th segments. An *all-gather* then copies every summed segment to every GPU, after which all GPUs hold the complete result. This two-phase form is the bandwidth-optimal way to build an all-reduce (Patarasuk and Yuan, 2009). This section treats a single all-reduce and drops the site index k of Sec. 3, writing $\mathbf{x}_i$ for GPU i ’s contribution $\mathbf{x}_{k,i}$ and $\mathbf{x}_i^j$ for its j -th segment,

$$\mathbf{s}^j = \sum_{i=1}^P \mathbf{x}_i^j, \quad (7)$$

$$\text{AllReduce}(\{\mathbf{x}_i\}) = \text{All-Gather}(\mathbf{s}^1, \dots, \mathbf{s}^P).$$

Because the segment sums $\mathbf{s}^j$ must be formed in high precision, we implement the reduce-scatter with an *all-to-all*, the collective in which every GPU sends a different piece to every other GPU. GPU i sends its j -th segment to the owner GPU j , which then holds the P copies it needs and adds them locally. For each segment it does not own, GPU i quantizes that segment to b bits and writes it directly into GPU j ’s buffer over the peer-to-peer link. The segment a GPU owns is never sent over the link, so it stays in full precision and enters the sum without any quantization error, and only the $P - 1$ copies arriving from peers are quantized. GPU j dequantizes those $P - 1$ copies, adds them to its own full-precision segment in high precision to obtain $\mathbf{s}^j$, and quantizes $\mathbf{s}^j$ once more for transport. The all-gather sends each quantized $\mathbf{s}^j$ back to the other GPUs, and every GPU dequantizes what it receives. GPU j reconstructs its own copy of $\mathbf{s}^j$ from that same b -bit code rather than keeping the high-precision value, so that its result is bit-for-bit identical to the copies the other GPUs receive. Figure 2(a) illustrates the process for $P = 4$, and Fig. 4 in Appendix C works the same case through with numbers.

Among bandwidth-optimal schemes, this decomposition also uses the fewest quantization steps. A ring all-reduce passes partial sums from GPU to GPU and requantizes each value once per hop, $\mathcal{O}(P)$ times in total. In our decomposition a value is quantized only when it crosses the link, once in the all-to-all and once in the all-gather, so each value is rounded at most twice however many GPUs participate (Table 1). The two-phase skeleton itself is standard and is shared by quantized systems such as ZeRO++ (Wang et al., 2024a), MegaScale-MoE (Jin et al., 2026), and FlashComm (Li et al., 2024, 2025b). We adopt it because it minimizes the roundings. On top of this, the compensator keeps the error accumulated over a whole forward pass at about twice the error of a single site, however deep the network (Secs. 5 and 6).

4.2 Pipelined execution and synchronization

The all-reduce lies between two dependent matrix multiplications, so its data path cannot overlap with them, and the room for overlap on this path is inside the primitive, between its quantization work and its data transfers. To

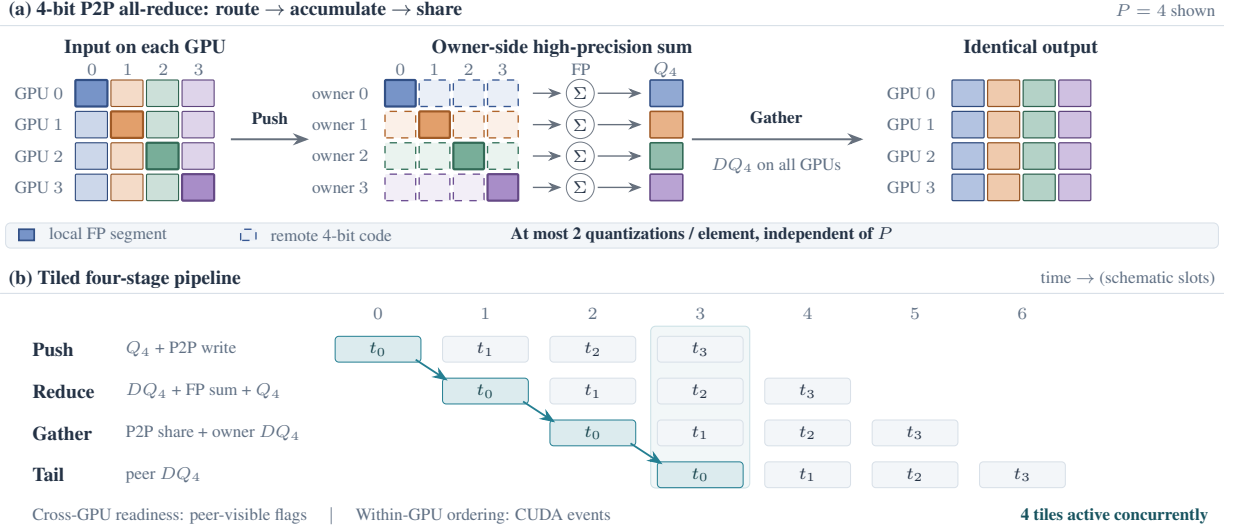


Figure 2. The quantized P2P all-reduce. (a) For $P = 4$, each GPU sends only its non-owned segments as 4-bit codes to their owners. Each owner dequantizes the received codes and sums them with its own unquantized segment in high precision. The reduced segment is quantized once for the all-gather, and every GPU, including the owner, dequantizes the same code to obtain identical outputs. Each element is quantized at most twice, independently of P . (b) Tiles are staggered across push, reduce, gather, and tail. Peer-visible flags signal cross-GPU readiness and CUDA events order work within a GPU, allowing quantization and transfers of different tiles to overlap.

expose it, we split each segment into small contiguous blocks, called *tiles*, and run the tiles through a four-stage pipeline. The stages are push, reduce, gather, and tail. Push quantizes the remote segments and writes them to their owners. Reduce dequantizes the received copies, adds them to the owned segment, and requantizes the result. Gather sends this requantized segment to the other GPUs and, on the owning GPU, dequantizes it to produce that GPU’s own portion of the output. Tail dequantizes the reduced segments arriving from the other owners. Because the tiles pass through the stages independently, one tile can be quantized while an earlier tile is still traveling over NVLink (Fig. 2(b)), so the quantization work of one tile is hidden behind the transfers of the tiles around it. The stream and event schedule is given in Appendix C.

The four stages run on different GPUs and must be kept in order across them. Routing that ordering through the host CPU would add latency and serialize the pipeline, so we synchronize the GPUs directly over their peer-to-peer (P2P) links. Each GPU keeps a small flag in memory that its peers can read over the P2P link. A stage advances the flag after it finishes a tile, and the consuming stage on another GPU waits until the flag reaches the value it expects before it starts. Ordering within a GPU is handled by CUDA events, and ordering across GPUs by these flags.

Signaling after each tile, rather than after the whole tensor, lets communication and quantization use different block sizes, which they need because they favor opposite choices. Peer-to-peer transfers reach their peak bandwidth with a few large blocks, whereas the quantization kernels are most efficient with many small blocks. An implementation that exchanges the tensor only after it has been staged in full must commit to a single block size for both. Tracking readiness at the tile level removes this coupling, because a consuming stage waits only for the specific tiles it needs rather than for the entire tensor. The exact flag values are given in Appendix C.

4.3 Quantization

The quantizer must not overflow the b -bit codes and must preserve model quality. We quantize group-wise. Each group of g consecutive values is sent as b -bit codes together with a small set of per-group parameters.

A symmetric group tracks its absolute maximum and uses a single scale s ,

$$q(x) = \text{clip}(\text{round}(x/s), -(2^{b-1}-1), 2^{b-1}-1), \quad (8)$$

$$s = \max_k |x_k| / (2^{b-1} - 1).$$

The scale sends the largest magnitude to the code $2^{b-1}-1$, so the most negative code -2^{b-1} is never used. Leaving it out keeps the levels symmetric about zero, which we found to be more stable at 4 bits.

Activations are often not centered on zero, so a symmetric grid, whose levels are symmetric about zero, spends much of its range on values that never appear. An asymmetric group instead fits an affine map to the observed range. This map has strictly more freedom, so it should be at least as accurate as the symmetric grid. At 4 bits a naive asymmetric map often is not, and the reason lies in its zero-point. The map is set by a scale s and an integer zero-point z ,

$$s = \frac{\max_k x_k - \min_k x_k}{2^b - 1}, \quad (9)$$

$$q(x) = \text{clip}(\text{round}(x/s) + z, 0, 2^b - 1),$$

and dequantizes as $\hat{x} = s(q - z)$. The integer z is the code that maps to 0, so 0 is represented exactly. A naive map instead anchors the grid at the group minimum, leaving the level nearest 0 at the non-integer position $-\min_k x_k/s$, so 0 falls between two codes. Since activation distributions are sharply peaked at zero, this rounds the densest part of the distribution, and with only sixteen levels the error there outweighs the finer resolution the asymmetric map was meant to gain.

Our policy is zero-aware. When a group straddles zero we round its zero-point to the nearest integer. When a group lies entirely on one side of zero the value 0 never occurs, so we keep the naive map and spend the whole grid on the values that are present. Groups in between blend smoothly between the two. This change alone removes the 4-bit regression, and asymmetric quantization then outperforms the symmetric policy (Sec. 7).

We expose only the two parameters that matter, the bit width $b \in \{4, 8\}$ and the group size g , and otherwise keep both policies fixed. The per-group storage is compact, one value for symmetric and a scale and a zero-point for asymmetric, and a backward-pass scaling prevents small gradient groups from underflowing it (Sec. 5.4).

5 Dual Error Feedback

The quantized all-reduce injects the error δ_k of Eqn. (1) at every site, with $\|\delta_k\| \leq q$. Left uncorrected, these errors accumulate, and the partial sum $\mathbf{S}_t$ of Eqn. (6), which governs the output error, grows to order Nq over a pass (Sec. 3.3). We remove this growth with an error-feedback compensator that realizes the correction $\mathbf{C}_k$ of Eqn. (5). It recovers each injected error, summed over the group, from the output of the all-reduce and feeds a running estimate of it back into the next all-reduce, so the injected errors cancel in $\mathbf{S}_t$ rather than accumulate. Section 5.1 defines the compensator and Sec. 5.2 shows, term by term, why it keeps $\mathbf{S}_t$ bounded.

5.1 Two-timescale error-feedback compensator

On each GPU the compensator keeps two moving averages, applies a blend of them as a correction before each all-reduce, and refreshes them from the residual it then forms. The update at site k on GPU i runs as follows.

It first forms a correction from its current averages and adds it to the contribution, before the all-reduce quantizes it,

$$\mathbf{c}_{k,i} = \eta(a \mathbf{m}_{k-1,i}^{(1)} + (1-a) \mathbf{m}_{k-1,i}^{(2)}), \quad \mathbf{x}'_{k,i} = \mathbf{x}_{k,i} + \mathbf{c}_{k,i}, \quad (10)$$

with blend weight $a \in [0, 1]$ and gain $\eta \in [0, 1]$. The all-reduce acts on the compensated contribution $\mathbf{x}'_{k,i}$ and returns $\hat{\mathbf{y}}_k = \sum_i \mathbf{x}'_{k,i} + \delta_k$ (Eqn. (1)). From it each GPU i forms the residual $\mathbf{u}_{k,i} = \mathbf{x}'_{k,i} - \hat{\mathbf{y}}_k/P$ of Sec. 3.3, whose two operands are already on the device, so it costs no communication, and whose group sum recovers the injected error, $\sum_i \mathbf{u}_{k,i} = -\delta_k$ (Eqn. (4)). The two averages are then refreshed with this residual, at decay rates $\beta_1, \beta_2 \in [0, 1]$,

$$\mathbf{m}_{k,i}^{(j)} = \beta_j \mathbf{m}_{k-1,i}^{(j)} + (1 - \beta_j) \mathbf{u}_{k,i}, \quad j = 1, 2. \quad (11)$$

Equation (10) is the error-feedback step. Summed over the group, the averages estimate the errors injected at earlier sites, and feeding them back into the input corrects those errors rather than passing them on. Each GPU stores and updates its own averages, so the compensator adds no communication, and summed over the group the corrections give the aggregate $\mathbf{C}_k = \sum_i \mathbf{c}_{k,i}$ of Eqn. (5). The averages are reset to zero at the start of each pass. One pair serves all sites of a pass, so the state costs two tensors of the size of the contribution per pass direction. Algorithm 1 lists the update.

Algorithm 1: Compensated all-reduce at site k (GPU i)

Input: contribution $\mathbf{x}_{k,i}$, state $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$, and parameters $\beta_1, \beta_2, a, \eta$
Output: block output $\hat{\mathbf{y}}_k$

```

1  $\mathbf{c} \leftarrow \eta(a \mathbf{m}^{(1)} + (1-a) \mathbf{m}^{(2)})$  // correction
2  $\mathbf{x}' \leftarrow \mathbf{x}_{k,i} + \mathbf{c}$  // apply
3  $\hat{\mathbf{y}}_k \leftarrow \text{QAR}(\{\mathbf{x}'_{k,j}\}_j)$  // all-reduce
4  $\mathbf{u} \leftarrow \mathbf{x}' - \hat{\mathbf{y}}_k / P$  // residual
5  $\mathbf{m}^{(1)} \leftarrow \beta_1 \mathbf{m}^{(1)} + (1-\beta_1) \mathbf{u}$  // update
6  $\mathbf{m}^{(2)} \leftarrow \beta_2 \mathbf{m}^{(2)} + (1-\beta_2) \mathbf{u}$  // update

```

5.2 Suppression of error accumulation

We first write the correction in closed form. Unrolling Eqn. (11) from the reset gives $\mathbf{m}_{k,i}^{(j)} = (1-\beta_j) \sum_{s \leq k} \beta_j^{k-s} \mathbf{u}_{s,i}$, so each average is a decaying sum of past residuals. Substituting into Eqn. (10), summing over the group, and using $\sum_i \mathbf{u}_{s,i} = -\delta_s$ gives the correction as a weighted sum of the earlier injected errors,

$$\begin{aligned} \mathbf{C}_k &= -\eta \sum_{s < k} w_{k-1-s} \delta_s, \\ w_r &:= a(1-\beta_1)\beta_1^r + (1-a)(1-\beta_2)\beta_2^r, \end{aligned} \quad (12)$$

where w_r is the weight placed on an error of age r , and the weights sum to one, $\sum_{r \geq 0} w_r = 1$.

We can now read off how much of each injected error survives in the partial sum. Putting $\rho_k = \delta_k + \mathbf{C}_k$ into $\mathbf{S}_t = \sum_{k \leq t} \rho_k$ (Eqn. (6)) and collecting the terms in each δ_s ,

$$\mathbf{S}_t = \sum_{s \leq t} \phi_{t-s} \delta_s, \quad \phi_m := 1 - \eta \sum_{r=0}^{m-1} w_r. \quad (13)$$

The coefficient ϕ_m is the fraction of an injected error that still remains in the partial sum m sites after it enters. It starts at $\phi_0 = 1$ and decreases with age, since $\sum_{r=0}^{m-1} w_r$ climbs toward $\sum_{r \geq 0} w_r = 1$. At gain $\eta = 1$ it decreases to zero. An old error is then subtracted away and eventually leaves $\mathbf{S}_t$, so the partial sum carries only a bounded window of recent errors. As a result, $\max_t \|\mathbf{S}_t\|$ stays bounded as the pass lengthens, in place of the order- Nq growth of the uncorrected sum $\sum_{k \leq t} \delta_k$ (Sec. 3.3). Section 6 turns Eqn. (13) into the exact bound, and treats the case $\eta < 1$, where ϕ_m settles at $1 - \eta$ and a fixed share of each error persists.

The two averages, with decay rates β_1 and β_2 , give the correction two time scales. A single average with decay β weights past residuals by $(1-\beta)\beta^r$, which shrink by the factor β each site, so it spans about $1/(1-\beta)$ sites. The decay rate β_1 sets a longer memory of about $1/(1-\beta_1)$ sites, and β_2 sets a shorter memory of about $1/(1-\beta_2)$ sites. The longer one estimates the part of the injected error that stays similar from site to site, and the shorter one adapts to the part that changes quickly. One average alone cannot do both. Combining moving averages at several time scales in this way is a known technique for accelerating first-order optimization (Pagliardini et al., 2025; Xie et al., 2024b). The two decay rates and the blend a are studied in Sec. 7.

5.3 Fusing the compensator into the all-reduce

The compensator of Sec. 5.1 runs two steps at every site, the apply step $\mathbf{x}' = \mathbf{x} + \mathbf{c}$ of Eqn. (10) before the all-reduce and, after it returns $\hat{\mathbf{y}}$, the update step of Eqn. (11) on the residual $\mathbf{u} = \mathbf{x}' - \hat{\mathbf{y}}/P$. Here and in the rest of this section we drop the site and GPU indices. Each step reads and writes tensors as large as $\mathbf{x}$ itself, so implemented as separate kernels the two steps add full memory passes of their own. This subsection shows that the apply step can instead run inside the entry pass of the quantized all-reduce of Sec. 4 and the update beside the next matrix multiplication, at a small added latency (Table 2). Section 5.4 then describes the deployment of the compensator on the backward pass.

A direct implementation surrounds the all-reduce with two kernels, an apply kernel that forms $\mathbf{x}' = \mathbf{x} + \mathbf{c}$ and writes it to a buffer of its own, and an update kernel that forms $\mathbf{u} = \mathbf{x}' - \hat{\mathbf{y}}/P$ and refreshes $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$. The two kernels are costly in two ways. They move about ten tensor-sized arrays through memory, while the all-reduce itself reads $\mathbf{x}$ once and writes $\hat{\mathbf{y}}$ once. They also stretch the critical path: the apply kernel must deliver $\mathbf{x}'$ before the all-reduce can

start, and the update kernel needs $\hat{\mathbf{y}}$, so it starts only after the all-reduce returns. The all-reduce is bandwidth bound, so the extra traffic cannot hide behind computation, and the two effects together place the full cost of compensation in the latency of every site.

On entry the all-reduce already moves the full tensor through registers. Every element of $\mathbf{x}$ is loaded, the quantization statistics are formed, and the value is quantized for transport (Sec. 4.2). The apply step needs $\mathbf{x}$, which the entry pass is holding. The update step needs $\hat{\mathbf{y}}$, which the next matrix multiplication only reads, so the update can run while that multiplication does.

The apply step is therefore fused into the entry pass. As each element of $\mathbf{x}$ arrives in registers, the kernel also loads $\mathbf{m}^{(1)}$ and $\mathbf{m}^{(2)}$, forms $\mathbf{c} = \eta(a\mathbf{m}^{(1)} + (1-a)\mathbf{m}^{(2)})$ and $\mathbf{x}' = \mathbf{x} + \mathbf{c}$ in registers, and only then computes the group statistics and quantizes. The tensor $\mathbf{x}'$ exists only in registers and is never written to memory. This is also the only place where the step can go. The all-reduce quantizes $\mathbf{x}'$, not $\mathbf{x}$ (Sec. 5.1), so the quantization statistics must be taken on $\mathbf{x}'$, and the entry load is the last moment before they are formed.

The update step is taken off the critical path. Once the exit pass has written $\hat{\mathbf{y}}$, the update runs on a separate stream alongside the next matrix multiplication. It recomputes $\mathbf{x}' = \mathbf{x} + \mathbf{c}$ from $\mathbf{x}$, $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$, forms $\mathbf{u} = \mathbf{x}' - \hat{\mathbf{y}}/P$, and applies $\mathbf{m}^{(j)} \leftarrow \beta_j \mathbf{m}^{(j)} + (1 - \beta_j) \mathbf{u}$ in place. The apply step of the next site waits for it, so it always reads current averages. Every GPU, owner included, dequantizes $\hat{\mathbf{y}}$ from the same transported code (Sec. 4.1), so the residuals across the group are formed from a common $\hat{\mathbf{y}}$, as the identity $\sum_i \mathbf{u}_i = -\delta$ of Eqn. (4) requires.

After fusion the apply kernel and the $\mathbf{x}'$ buffer are gone, and neither step runs as a serial phase on the critical path. The apply step inherits the pipelining of Sec. 4.2 and runs tile by tile inside the entry stage, overlapped with the communication of the other tiles, and the update runs beside the next matrix multiplication. The extra traffic falls from about ten tensor-sized arrays to about eight, reading $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ at entry, reading $\mathbf{x}$, $\hat{\mathbf{y}}$, $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ in the update, and writing $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ back. Six of the eight are the cost any implementation pays to consult the averages and refresh them, and the update traffic overlaps with the matrix multiplication. The added arithmetic hides behind the memory accesses of the entry pass and of the update, and Table 2 measures the resulting cost. Fusion changes when and where each step executes, not what it computes, and $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ remain the only state, updated in place.

5.4 The backward pass

Training runs the all-reduce in both directions. The compensator is unchanged on the backward pass, with the same recurrence and the same coefficients, but its working conditions change in two ways.

First, backward errors enter the computation graph at a different point. A forward site delivers its output directly to the residual addition, while the output of a backward site reaches the gradient stream through the Jacobian of the normalization (Sec. 3.2), the setting in which Appendix A proves the backward source bound. Residuals of the two directions therefore have different coordinates and different magnitudes, and a single pair $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ fed by both streams would form a correction that matches neither pass. Each direction keeps its own pair and runs the recurrence of Sec. 5.1 along its own sequence of sites.

Second, gradients are small enough to underflow the storage of the quantizer’s scale. The quantizer of Sec. 4.3 stores one scale s per group in half precision. Backward gradients are orders of magnitude smaller than activations, so a group scale can fall below the smallest positive half-precision number and load back as zero. Every value in that group then dequantizes to zero. The remedy is a change of units around the communication. Before the quantized all-reduce, every GPU multiplies its gradient by the same power-of-two factor, chosen from the largest gradient magnitude across the tensor-parallel group with a one-scalar all-reduce. After the all-reduce, it divides the result by that factor. The two steps of the compensator run outside the scaling, so $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ stay in original gradient units. The scaling is exact and leaves the analysis of Secs. 5.1 and 6 unchanged, as Appendix C verifies.

6 Theoretical Guarantees

6.1 Decomposition of the output error

The quantized all-reduce injects an error at each of the N sites of a pass. Let $\mathbf{z}_k$ be the hidden state carried by the exact all-reduce and $\hat{\mathbf{z}}_k$ the hidden state carried by the quantized one, so that

$$\mathbf{z}_{k+1} = \mathbf{z}_k + F_k(\mathbf{z}_k), \quad \hat{\mathbf{z}}_{k+1} = \hat{\mathbf{z}}_k + F_k(\hat{\mathbf{z}}_k) + \boldsymbol{\rho}_k, \quad (14)$$

where F_k is the block at site k , the map that normalizes the hidden state and applies the attention or MLP transformation, and $\rho_k = \delta_k + \mathbf{C}_k$ is the net error of Eqn. (5). Write $\mathbf{E}_k = \hat{\mathbf{z}}_k - \mathbf{z}_k$ for the deviation between the two, with $\mathbf{E}_0 = \mathbf{0}$. Subtracting the recursions of Eqn. (14),

$$\mathbf{E}_{k+1} = \mathbf{E}_k + \psi_k + \rho_k, \quad \psi_k = F_k(\hat{\mathbf{z}}_k) - F_k(\mathbf{z}_k). \quad (15)$$

Equation (15) is exact, and it separates the two roles of a site. The term ψ_k propagates the deviation accumulated so far, and adds nothing of its own, since $\mathbf{E}_k$ determines it. The term ρ_k injects the new error of site k . Expanding F_k about $\mathbf{z}_k$,

$$\psi_k = \mathbf{J}_k \mathbf{E}_k + \mathbf{r}_k, \quad \|\mathbf{r}_k\| \leq \frac{1}{2} H_k \|\mathbf{E}_k\|^2, \quad (16)$$

with $\mathbf{J}_k$ the Jacobian of F_k at $\mathbf{z}_k$ and H_k a bound on its second derivative on the segment from $\mathbf{z}_k$ to $\hat{\mathbf{z}}_k$. Substituting Eqn. (16) into Eqn. (15) and unrolling from $\mathbf{E}_0 = \mathbf{0}$,

$$\begin{aligned} \mathbf{E}_N &= \sum_{t=0}^{N-1} \Phi_{N,t+1} (\rho_t + \mathbf{r}_t), \\ \Phi_{N,j} &= (\mathbf{I} + \mathbf{J}_{N-1}) \cdots (\mathbf{I} + \mathbf{J}_j). \end{aligned} \quad (17)$$

The deviation at the output is a sum of the injected errors, each propagated there by the Jacobians of the sites that follow it, together with the second-order remainders. The remainders are second order in the injected errors, and Proposition 1 in Appendix A quantifies them. The first-order terms therefore carry the analysis, and two quantities remain to be bounded, the propagation Φ and the accumulation of the ρ_t .

The accumulation must be measured through the partial sum $\mathbf{S}_t$ of Eqn. (6), not through the individual errors. A term-by-term triangle inequality on Eqn. (17) allows $\|\rho_k\|$ as large as $2q$ and gives a bound of order Nq . This bound shows no benefit from error feedback, because feedback does not reduce any single net error. Instead, feedback makes the errors cancel as they accumulate (Sec. 3.3). Summation by parts rewrites the first-order part of Eqn. (17) so that the errors enter only through their partial sums, with coefficients formed from the propagation (Appendix A).

Section 6.2 bounds the two in turn, the partial sums by a constant for any error sequence, and the propagation in mean square under one assumption on the trained network. Neither result restates existing error feedback theory. That theory concerns gradient compression, where the compressed errors perturb an optimization step and the outcome is a convergence rate. Here the errors enter the forward computation and are then carried through the network itself. This propagation has no counterpart in the gradient setting, and Assumption 1 and Theorem 2 address it.

6.2 Depth-independent bounds

This subsection contains two results. Theorem 1 bounds the source. With compensation, the accumulated error at the sites stays below a constant, about $2q$ at gain $\eta = 1$, for any error sequence, while without compensation it can reach Nq . This requires no probabilistic model and no property of the network. Theorem 2 bounds the output. Under a noise model for the injected errors and one assumption on the trained network, the mean-square first-order deviation of the output, measured against the energy of the hidden state, stays at the level set by a single rounding error, at every depth. Theorem 2 builds on Theorem 1, whose constant reappears inside its bound. The new element is the propagation, which Assumption 1 controls.

We start with the source. At gain $\eta = 1$ the corrections return each error in full. Of an error injected m sites ago, the fraction $\sum_{r < m} w_r$ has been subtracted back out by later corrections, and the fraction

$$T_m := \sum_{r \geq m} w_r \quad (18)$$

remains. These tails are exactly the coefficients ϕ_m of Eqn. (13), so each error enters the partial sum with weight 1 and decays toward zero, and no partial sum can exceed the total of the tails.

Theorem 1 (source bound). *At gain $\eta = 1$, for every error sequence with $\|\delta_k\| \leq q$,*

$$\begin{aligned} \max_t \|\mathbf{S}_t\| &\leq q \sum_{m \geq 0} T_m = (1 + \mu_w) q, \\ \mu_w &= \sum_{r \geq 0} r w_r, \end{aligned} \quad (19)$$

where μ_w is the mean age, in sites, at which the compensator returns an injected error. Without compensation the same quantity reaches Nq .

The theorem has one hypothesis, the bound q on each error. The T_m are nonnegative, so the proof is a triangle inequality, and it covers errors that are dependent, correlated with the activations, or chosen adversarially. The constant reflects how the compensator works. An injected error stays in the partial sum until later corrections have subtracted it back out, which takes $1 + \mu_w$ sites on average, about 2.1 for the decay rates and blend used in the experiments. At any moment the partial sum therefore holds only the errors of the last few sites, about two errors in total. Without compensation errors are never subtracted, all N of them accumulate, and the partial sum reaches Nq . Compensation reduces the accumulated source from Nq to $(1 + \mu_w)q$, a factor of $N/(1 + \mu_w)$, about 27 at the $N = 56$ of Sec. 3.2.

Theorem 1 controls what the sites accumulate. The deviation at the output also depends on the factor Φ of Eqn. (17), the transformation the network applies to each error on its way to the output. Bounds that take the worst case of every site grow with depth. The standard estimate is $\|\Phi\| \leq (1 + L)^N$, with L a Lipschitz constant of the blocks. Growth with depth is not an artifact of the estimate. Proposition 2(a) in Appendix A constructs a network that amplifies one fixed direction at every site, and in it even isotropic injections produce a deviation that grows exponentially in $\sqrt{N}$. A depth-independent bound must therefore exclude networks that keep amplifying one direction. The noise model rules out injections aligned with such a direction, and Assumption 1 rules out the networks. These are the two hypotheses of Theorem 2.

We first treat the injections. A rounding error is fixed by the quantizer at the moment of injection, from the activations alone, with no reference to the directions the network later amplifies. We model the errors accordingly. Conditioned on everything injected earlier, δ_s has mean zero, no preferred direction, and norm at most q (Widrow and Kollár, 2008). Under this model the contributions of different sites are uncorrelated, and the linearized deviation $\mathbf{E}_N^{\text{lin}}$, the first-order part of Eqn. (17), satisfies

$$\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2 \leq \frac{q^2}{d} \sum_s \|\mathbf{K}_s\|_F^2, \quad (20)$$

where d is the dimension of the hidden state, $\mathbf{K}_s$ is the matrix carrying δ_s to the output, built from the tails T_m and the propagation Φ (Appendix A), and $\|\cdot\|_F$ is the Frobenius norm.

We next treat the networks. The reference scale here is the hidden state itself. Let $\sigma_k^2 = \|\mathbf{z}_k\|^2/d$, the mean-square entry of the hidden state at site k . In a pre-normalization transformer σ_k rises with depth, because every site adds new content to the hidden state (Sun et al., 2025). The hidden state therefore gains energy at a definite rate, and Assumption 1 allows a deviation to gain energy at most at that same rate.

Assumption 1 (bounded reinforcement). *There are $\varepsilon_k \geq 0$ with $\sum_k \varepsilon_k \leq \Lambda$, for a Λ independent of N , such that for every site k , every earlier site s , and every isotropic injection δ_s from the noise model, the deviation $\mathbf{v} = \Phi_{k,s+1} \delta_s$ it produces at site k satisfies*

$$\mathbb{E} \|(\mathbf{I} + \mathbf{J}_k) \mathbf{v}\|^2 \leq (1 + \varepsilon_k) \frac{\sigma_{k+1}^2}{\sigma_k^2} \mathbb{E} \|\mathbf{v}\|^2.$$

Appendix A completes the family of deviations $\mathbf{v}$ used in the proofs.

The assumption does not restrict what a block does to any single vector. A block may rotate an arriving deviation, spread it across coordinates, or overwrite it. The ε_k give further room. Single sites may amplify beyond the ratio $\sigma_{k+1}^2/\sigma_k^2$, and only their total over all sites must stay bounded independently of depth. The assumption excludes one behavior, a network that keeps amplifying the arriving errors from site to site. Trained transformers show the opposite tendency. Consecutive blocks tend to cancel each other's contributions (Patrawala et al., 2025), and small perturbations grow with depth at the rate of the hidden state, not faster (Budzinskiy et al., 2025). Such an exclusion is also necessary. The amplifying network of Appendix A satisfies the noise model, so no depth-independent bound holds without an assumption of this kind.

Write $\bar{q} = q/(\sqrt{d} \sigma_1)$, the per-entry size of one injected error relative to the scale σ_1 of the hidden state after the first site.

Theorem 2 (propagation). *Under Assumption 1, the noise model, and the regularity conditions of Appendix A, the following hold.*

(i) In mean square,

$$\frac{\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} \leq 2 \left[\tau_2 \frac{\sigma_1^2}{\sigma_N^2} + (1 + \mu_w)^2 C_0 e^\Lambda \right] \bar{q}^2,$$

with $\tau_2 = \sum_m T_m^2$ a constant of the kernel and C_0 a constant of the checkpoint defined in Appendix A.

(ii) For every $p \in (0, 1)$, with probability at least $1 - p$,

$$\frac{\|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} \leq C \bar{q}^2 \log \frac{2}{p} + C' \bar{q}^2 \log^2 \frac{2}{p},$$

with C and C' the depth-independent constants of Appendix A.

The term $\tau_2 \sigma_1^2 / \sigma_N^2$ in (i) decays with depth. No other quantity in (i) and (ii) grows with N .

The quantity bounded is the energy of the deviation measured against the energy of the hidden state. The second-order remainders are at most $C_2 \bar{q} \sqrt{N}$ times the scale $\bar{q} \sqrt{d} \sigma_N$ of the linear bound, for as long as the deviation stays within twice its linear bound (Proposition 1, sharp by Proposition 2(b), see also Corollary 2). Here C_2 depends on the curvature and the energy growth of the network and grows at most as $\log^2(N/p)$. Up to the logarithm in C_2 , the factor $C_2 \bar{q} \sqrt{N}$ is small whenever the noise injected over a pass is a small fraction of the hidden state.

In the same mean-square terms the advantage over the uncompensated primitive is direct. With compensation the variance of the accumulated source stays below $\tau_2 \bar{q}^2$ at every site, about $1.3 \bar{q}^2$ for the decay rates and blend used in the experiments. Theorem 2 turns this bounded source into a first-order output deviation at the level of a single rounding error, independent of depth. Without compensation each site keeps its full contribution and the same bound grows as $\log N$ (Remark 2). Compensation multiplies the term of each site in the bound by at most $2(1 + \mu_w)^2 C_0 \Delta_k / \sigma_k^2$, where $\Delta_k = \sigma_{k+1}^2 - \sigma_k^2$ is the energy that block k adds, so the sum stays bounded.

The source analysis extends to the backward pass. Its injected errors enter through the Jacobian of the normalization (Sec. 3.2), and Appendix A proves the corresponding depth-independent source bound, in mean square and under a smoothness condition on the normalization Jacobians. Section 5.4 describes the backward implementation.

6.3 Choice of the compensator weights

Theorems 1 and 2 fix the gain at $\eta = 1$ and take the decay rates β_1, β_2 of Sec. 5 as given. Both choices affect the guarantee.

The gain interpolates between no compensation and full compensation. At gain η the coefficient of an injected error in the partial sum is $(1 - \eta) + \eta T_{t-s}$, and the source bound follows the same split,

$$\max_t \|\mathbf{S}_t\| \leq (1 - \eta) Nq + \eta (1 + \mu_w) q. \quad (21)$$

Theorem 1 is the case $\eta = 1$, the only gain at which the growing term vanishes. At $\eta < 1$, the fraction $1 - \eta$ of every error is never corrected and accumulates as it would without compensation. The compensator state is reset at the start of each pass, so this growth does not carry over between passes. In mean square this uncorrected share is small. The net error is linear in the gain, so the first-order output deviation is $1 - \eta$ times the uncompensated deviation plus η times the deviation of Theorem 2 (Corollary 1). At the gain $\eta = 0.75$ used in the experiments, the part that grows with depth is the uncompensated bound scaled by $2(1 - \eta)^2 = 1/8$, which grows only as $\log N$. At $N = 56$ the variance of the source is about $5q^2$, against $56q^2$ without compensation.

The decay rates enter the source bound only through μ_w , so every choice of β_1, β_2 keeps the source below a constant independent of N . The constant grows with the memory of the averages, and is about $2q$ at the values used in the experiments. The smallest constant belongs to the limit $\beta_1, \beta_2 \rightarrow 0$, where the compensator keeps only the error of the preceding site and reduces to single-step error feedback. This optimum is a worst-case one, attained on adversarial sequences, and practical error sequences are not adversarial. Away from that extreme, correcting with a short-range and a long-range average together does better than correcting with the last error alone, and Sec. 7 reports the ablation over β_1, β_2 , and η around the values used in the experiments.

Table 2. Median all-reduce latency (ms) at $P = 4$ over message sizes, measured in one run. Lower is better. Bold marks the minimum in each column. The last two rows attach the compensator to our 4-bit all-reduce, unfused and fused. The fused row updates its state beside the next matrix multiplication and reports the time the call adds to the critical path, measured as the time of the matrix multiplications and the all-reduce together minus that of the matrix multiplications alone.

	64 MB	128 MB	256 MB	512 MB	1 GB
NCCL 16-bit	0.70	1.27	2.45	4.60	9.00
Two-collective, 8-bit	0.74	1.26	2.27	4.14	7.87
Two-collective, 4-bit	0.62	1.00	1.76	3.25	5.76
Ours, 8-bit	0.56	0.91	1.42	2.80	5.51
Ours, 4-bit	0.36	0.62	0.94	1.73	3.40
+ unfused compensator	1.27	2.33	4.50	8.78	17.49
+ fused compensator	0.44	0.75	1.19	2.15	4.30

Table 3. End-to-end speedup ($\times$) of our fused, compensated 4-bit all-reduce over the 16-bit NCCL all-reduce on eight A100 GPUs ($P = 8, g = 128$). Speedup is $T_{\text{NCCL}}/T_{\text{ours}}$. Inference reports the time to first token of Llama-3-8B and Qwen3-8B, training the mean step time. Load is the duty cycle of background peer-to-peer copies. AR slowdown is $T_{\text{AR}}^{\text{NCCL}}(\ell)/T_{\text{AR}}^{\text{NCCL}}(0)$ for a fixed-size 1 GB 16-bit NCCL all-reduce, the per-call message of the batch-64 runs, at load ℓ against no load.

Load	AR slowdown ($\times$)	Inference (TTFT)		Training (step time)	
		Llama	Qwen	Llama	Qwen
0%	1.00	1.16	1.09	1.09	1.08
50%	1.80	1.33	1.21	1.22	1.20
75%	2.07	1.38	1.24	1.27	1.25
100%	2.73	1.44	1.31	1.38	1.34

7 Experiments

The experiments cover four parts. We first measure speed, including the latency of the quantized all-reduce, the overhead of the compensator before and after fusion, and end-to-end speed in inference and training (Sec. 7.1). We then evaluate inference accuracy under 4-bit communication (Sec. 7.2), compare training loss curves against the 16-bit tensor-parallel baseline (Sec. 7.3), and report ablations, including the sweep over β_1 , β_2 , and η (Sec. 7.4).

Setup. All experiments run on a single server with eight A100 GPUs connected by NVLink. The inference experiments serve Llama-3-8B (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025) at $P = 4$ and evaluate language modeling perplexity together with standard downstream accuracy benchmarks. The training experiments pretrain Qwen3-0.6B from scratch at $P = 4$ with BF16 mixed precision and sequence length 2,048. Data and schedules are given in Sec. 7.3. End-to-end speed is measured on Llama-3-8B and Qwen3-8B at $P = 8$ (Sec. 7.1). Unless stated otherwise, communication uses the 4-bit quantized all-reduce of Sec. 4 with group size $g = 128$ and the zero-aware asymmetric quantizer, and the compensator uses $\eta = 0.75$, $a = 0.35$, $\beta_1 = 0.7$, and $\beta_2 = 0.3$ (Sec. 7.4).

7.1 Latency and end-to-end speed

Table 2 reports the median latency of one all-reduce at $P = 4$ for 16-bit messages of 64 MB to 1 GB per GPU. Each value is the median of 80 iterations after 30 warmup iterations, and all rows come from the same run. The fused compensator updates its state while the next matrix multiplication runs, so for the fused row we time the matrix multiplications and the all-reduce together and subtract the time of the matrix multiplications alone. The rows are the 16-bit NCCL baseline, a two-collective quantized all-reduce assembled from DeepSpeed’s quantized all-to-all and quantized all-gather primitives in the style of ZeRO++ (Wang et al., 2024a) at 8 and 4 bits, our all-reduce at 8 and 4 bits without compensation, and the 4-bit all-reduce with the compensator attached in two forms, as separate PyTorch operations and fused as described in Sec. 5.3.

On the 1 GB message, our uncompensated 8-bit all-reduce is $1.63\times$ faster than NCCL, and our uncompensated 4-bit all-reduce is $2.65\times$ faster and takes 37.8% of the NCCL time. At equal bit width, our all-reduce is $1.43\times$ faster than the two-collective baseline at 8 bits and $1.69\times$ faster at 4 bits. The gap comes from the pipelined design of Sec. 4, which overlaps quantization with communication instead of running two dependent collectives back to back.

The measured 4-bit speedup is below the $4\times$ byte ratio of 4-bit against 16-bit transport, for two reasons. First, the payload is not the only traffic. With group size $g = 128$, every 128 quantized values share one pair of half-precision quantization parameters, so a group costs 68 bytes instead of 64, which alone lowers the ideal ratio from $4\times$ to about $3.8\times$. Second, quantization adds computation on the datapath. Each message is quantized before it is sent and dequantized after it is received, and the owner performs the sum in high precision between a dequantize and a requantize, none of which exists in the 16-bit all-reduce. The pipeline hides most of this computation behind the transfers, and the remaining gap to $3.8\times$ is the part it cannot hide.

The two compensator rows measure the cost of compensation. Built from separate PyTorch operations, the compensator adds two full passes over the tensor and an intermediate buffer, and on the 1 GB message it adds 14.09 ms to the all-reduce. With the apply step fused into the entry pass and the update running beside the next matrix

Table 4. WikiText perplexity for Llama-3-8B and Qwen3-8B. Lower is better. The measured rows use $P = 4$. The FlashComm rows are quoted from (Li et al., 2024) and set in gray. That paper reports no 16-bit perplexity, so these rows are compared through the change from 8 to 4 bits. Dashes mark results not reported in the cited work.

	Llama-3-8B	Qwen3-8B
16-bit	6.07	9.70
4-bit, no compensation	6.73	10.45
Ours, 8-bit	6.07	9.70
Ours, 4-bit	6.36	9.80
FlashComm 8-bit (quoted)	6.14	–
FlashComm 4-bit (quoted)	6.70	–

Table 5. Math reasoning accuracy (%) for Llama-3-8B and Qwen3-8B at $P = 4$. GSM8K is scored by strict-match and flexible-extract. Early-site 8-bit promotes the all-reduce calls of the first half of the layers to 8 bits, 32 calls on Llama-3-8B and 36 on Qwen3-8B, and keeps the rest at 4 bits. The cited work reports neither benchmark.

Model	Method	GPQA	GSM8K	
			strict	flexible
Llama-3-8B	16-bit	33.71	49.81	49.96
	Ours, 4-bit	33.03	46.17	46.40
	+ early-site 8-bit	33.39	49.81	50.11
Qwen3-8B	16-bit	38.51	82.49	82.56
	Ours, 4-bit	38.84	80.06	80.52
	+ early-site 8-bit	39.95	81.73	82.18

multiplication, it adds 0.90 ms, 10% of the NCCL time. The fused path matches a PyTorch reference implementation of the compensator exactly, with a maximum absolute difference of zero at every size. With fused compensation, the 4-bit all-reduce adds 4.30 ms to the critical path for the 1 GB message, against 9.00 ms for NCCL, a $2.09\times$ reduction.

Table 3 reports end-to-end speed on the same server at $P = 8$ with group size $g = 128$. We compare the 16-bit NCCL all-reduce with our fused, compensated 4-bit all-reduce without background load and with background peer-to-peer copies running between the GPUs for 50%, 75%, and 100% of the time, which emulates an interconnect shared with other traffic. In each setting both collectives run under the same load, and we report the speedup $T_{\text{NCCL}}/T_{\text{ours}}$.

For inference, we measure the time to first token of Llama-3-8B and Qwen3-8B with batch size 64 and a 2,048-token prefill. For training, we measure the mean step time of Llama-3-8B with the same batch size and sequence length and of Qwen3-8B with batch size 32 and sequence length 2,048. The training measurement covers the all-reduce calls of both the forward and the backward pass, including the gradient rescaling of the 4-bit path. Without background load, the speedup is $1.16\times$ on Llama-3-8B and $1.09\times$ on Qwen3-8B for inference, and $1.09\times$ and $1.08\times$ for training. It grows with the load and reaches $1.44\times$ and $1.31\times$ for inference and $1.38\times$ and $1.34\times$ for training at 100% load.

These speedups are bounded by the share of time the all-reduce blocks the 16-bit baseline. On Llama-3-8B without background load, this share is 20.3% of the time to first token and 18.8% of the training step, which bounds the speedups at $1.25\times$ and $1.23\times$. At 100% load, the shares rise to 43.0% and 36.2%, and the bounds to $1.75\times$ and $1.57\times$. A prefill waits on every all-reduce, and our all-reduce synchronizes through flags that the GPUs set and poll themselves, tile by tile (Sec. 4.2), so it starts on each tile as soon as its data arrives and loses little time when links are congested. At 100% load, the fixed-size 16-bit NCCL all-reduce takes $2.73\times$ as long as it does without background load. The speedup thus grows with the share of the step that the all-reduce takes.

7.2 Inference accuracy

The speedups reported in Sec. 7.1 matter only if accuracy is preserved. We evaluate Llama-3-8B and Qwen3-8B with the 4-bit all-reduce at every forward site, on three groups of tasks: language modeling perplexity on WikiText (Merity et al., 2017), accuracy on seven standard benchmarks (PIQA (Bisk et al., 2020), ARC-easy, ARC-challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), MMLU (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024c)), and reasoning on GPQA (Rein et al., 2024) and GSM8K (Cobbe et al., 2021). All measured rows, including the 16-bit baselines, are obtained by us in a single evaluation harness (Gao et al., 2023), and each task uses the standard shot count of the harness. We compare two rows on a task with a two-sided test at the 5% level, using the binomial standard error of each accuracy. The FlashComm rows are quoted from the original papers (Li et al., 2024, 2025b) and are set apart in Tables 4 and 6.

Perplexity. Table 4 reports WikiText perplexity, where lower is better. The 4-bit path raises perplexity by 0.29 on Llama-3-8B (6.07 to 6.36) and by 0.10 on Qwen3-8B (9.70 to 9.80), and the 8-bit path leaves it unchanged. An increase of this size is small at these perplexity levels, and it is much smaller than the increase that the raw quantizer error causes. Without compensation the 4-bit perplexity is 6.73 and 10.45, and the compensator closes most of the distance between those values and the 16-bit model. Moving from 8 to 4 bits raises the perplexity of Llama-3-8B by 0.29 with our all-reduce and by 0.56 with FlashComm (Li et al., 2024). Without compensation our 4-bit path

Table 6. Downstream accuracy (%). The FlashComm rows are quoted from (Li et al., 2024, 2025b), set in gray, and are read against the 16-bit references of those papers. All other rows are measured in our harness at $P = 4$ and are read against the 16-bit row of the same model. Dashes mark results not reported in the cited work.

Model	Method	PIQA	ARC-e	ARC-c	HellaSwag	WinoGrande	MMLU	MMLU-Pro
Llama-3-8B	16-bit	80.52	77.69	53.39	79.26	73.80	61.54	32.34
	4-bit, no compensation	78.18	72.22	48.38	77.35	70.64	58.82	28.76
	Ours, 8-bit	80.68	77.77	53.07	79.38	74.03	61.50	31.27
	Ours, 4-bit	79.65	76.68	51.79	78.93	70.72	59.01	30.13
	FlashComm 8-bit (quoted)	80.58	77.40	52.47	79.09	73.09	–	–
	FlashComm 4-bit (quoted)	80.09	75.84	51.02	78.11	70.48	–	–
Qwen3-8B	16-bit	78.28	80.22	56.81	78.76	72.22	73.02	47.20
	4-bit, no compensation	76.77	77.73	55.54	76.47	67.10	71.08	45.51
	Ours, 8-bit	79.39	79.88	56.14	78.65	71.74	72.95	47.75
	Ours, 4-bit	79.38	81.21	56.00	77.81	70.32	72.65	46.38
	FlashComm 8-bit (quoted)	77.58	80.89	56.57	74.92	68.11	–	–
	FlashComm 4-bit (quoted)	76.82	79.38	55.63	74.16	67.72	–	–

risers by 0.66, close to the 0.56 of FlashComm, so the difference in perplexity comes from the compensator. The task benchmarks below test whether the remaining difference affects answers.

Downstream accuracy. Table 6 reports seven standard benchmarks. At 8 bits, our all-reduce matches 16-bit inference, with average scores within 0.12 points on both models. At 4 bits, Qwen3-8B loses only 0.39 points on average and scores higher than 16-bit on PIQA and ARC-easy, and Llama-3-8B loses 1.66 points on average. The compensator keeps these losses small. Without it, the average drop is 3.46 points on Llama-3-8B and 2.33 on Qwen3-8B, so compensation removes about half of the loss on Llama-3-8B and 83% of it on Qwen3-8B. Against the 16-bit row, the uncompensated 4-bit path loses significant accuracy on 10 of the 14 model and task pairs. Compensation raises accuracy on all 14 pairs, and significantly on 6 of them. On the five tasks that FlashComm also reports, moving from 8 to 4 bits costs our all-reduce 1.43 points on Llama-3-8B and 0.22 on Qwen3-8B on average, against 1.42 and 0.87 for FlashComm.

Math reasoning. Table 5 reports GPQA and GSM8K. For GSM8K we report both scoring modes of the harness: strict-match, which accepts the answer only in the exact expected format, and flexible-extract, which accepts any correctly extracted number. On GPQA the two paths differ by less than a point on both models and in either direction, 33.03 against 33.71 on Llama-3-8B and 38.84 against 38.51 on Qwen3-8B. On GSM8K the 4-bit path scores 46.17 strict-match against 49.81 for 16-bit, a loss of 48 of 1,319 problems. The two tasks differ in what they ask of the model. GPQA is answered with a single choice, while a GSM8K solution is a generated chain of several hundred tokens in which each token conditions on the ones before it, so one flipped intermediate token can make the rest of the derivation wrong. The compensator keeps the accumulated deviation small, but at 4 bits even a single rounding error occasionally flips a token, and a long chain offers many chances. We therefore spend more bits on a small part of the traffic. An error injected at an early call enters the residual stream and reaches every later layer, while an error injected near the output reaches only a few. In a long generated chain, early errors therefore have more chances to change a token. We promote the all-reduce calls of the first half of the layers, on each prefill and each decode forward, to 8 bits, and call this policy early-site 8-bit. This recovers most of the loss. It moves about 1.5 times the bytes of uniform 4-bit communication. On Llama-3-8B it recovers strict-match to 49.81, the 16-bit score, and reaches 50.11 on flexible-extract against 49.96 for 16-bit. On Qwen3-8B it reaches 81.73 and 82.18 against 82.49 and 82.56 for 16-bit. Early-site 8-bit is used only in the serving experiments of this section. The training experiments of Sec. 7.3 use uniform 4-bit communication throughout.

7.3 Training

We pretrain Qwen3-0.6B (Yang et al., 2025) from scratch at $P = 4$ with the 4-bit all-reduce at every forward and backward site. The main experiment trains on 10B tokens of Dolma C4 (Raffel et al., 2020; Soldaini et al., 2024) with a global batch of 384 sequences for 12,716 steps, using AdamW, 100 warmup steps, and cosine decay (Appendix B). The 16-bit and 4-bit runs share the data, batch size, optimizer, and schedule. Low-bit training has been observed to benefit from a larger learning rate than its full-precision counterpart (Wang et al., 2023; Xie et al., 2025), so we give the 4-bit run a slightly larger peak learning rate, 3.5×10^{-4} against 3.0×10^{-4} for 16-bit.

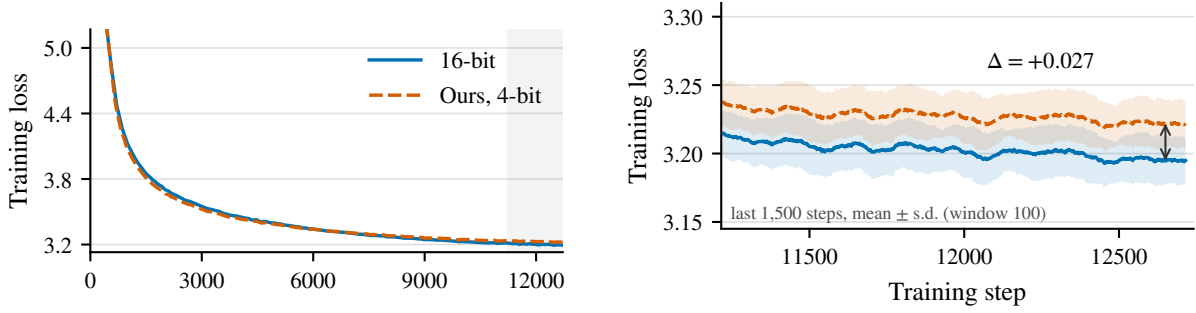


Figure 3. Training loss of Qwen3-0.6B at $P = 4$ on 10B tokens of Dolma C4, 16-bit at peak LR 3.0×10^{-4} against our 4-bit all-reduce at peak LR 3.5×10^{-4} . Left: full run (50-step moving average, y axis clipped, gray band marks the steps on the right). Right: last 1,500 steps (100-step average, ± 1 s.d. band).

Table 7. Training loss of Qwen3-0.6B at $P = 4$ on Dolma C4. (a) Full 10B-token training runs. (b) A separate 4-bit learning-rate ablation with batch size 32 and 4,096 steps. Tail-100 is the mean loss over the last 100 steps. In (a), Δ is relative to the 16-bit run, and in (b), $\Delta_{3.5}$ is relative to the run at LR 3.5×10^{-4} . All differences use unrounded tail-100 losses.

(a) Full training: 10B tokens, batch size 384					(b) LR ablation: batch size 32, 4,096 steps				
Method	Max LR ($\times 10^{-4}$)	Step 1000	Tail-100	Δ	Max LR ($\times 10^{-4}$)	2.0	2.5	3.0	3.5
16-bit	3.0	4.095	3.197	–	Tail-100 loss	4.135	4.078	4.016	3.952
Ours, 4-bit	3.5	4.042	3.225	+0.027	$\Delta_{3.5}$	+0.183	+0.126	+0.064	0.000

Figure 3 compares the two runs. Their loss curves track each other closely over all 12,716 steps, and the right panel magnifies the last 1,500 steps. Over the last 100 steps (tail-100), the 4-bit loss averages 3.225 and the 16-bit loss 3.197, a gap of 0.027 computed before rounding, less than 0.9% of the 16-bit loss. Early in training the 4-bit run, with its larger learning rate, has the lower loss, 4.042 against 4.095 at step 1,000 (Table 7a).

Table 7b supports this choice of learning rate. In separate, shorter 4-bit runs on Dolma C4 with a global batch of 32 sequences for 4,096 steps, the tail-100 loss falls steadily as the peak learning rate rises, from 4.135 at 2.0×10^{-4} to 4.078, 4.016, and 3.952 at 2.5, 3.0, and 3.5×10^{-4} . The final loss is sensitive to the learning rate (Xie et al., 2024a), and the 4-bit run gains from the larger value.

7.4 Ablations

We now remove or replace one component of the method at a time and measure the effect on perplexity, where lower is better. All runs serve Llama-3-8B at $P = 4$ with the same evaluation as Table 4, on WikiText and C4.

Table 8 evaluates every variant at 4 and at 8 bits. At 8 bits the variants span only 0.011 perplexity on WikiText and 0.020 on C4. At 4 bits they span 2.842 and 5.115. At 4 bits, the full design has much lower perplexity than the naive zero-point and no-compensation variants. Changing any single coefficient moves the 4-bit perplexity by at most 0.022 on WikiText and 0.041 on C4, so the method is not sensitive to these coefficients. We examine the 4-bit columns below.

The first factor is the group size. Coarsening the groups from 128 to 256 raises perplexity by 0.35 on WikiText and 0.79 on C4 with the symmetric quantizer, and by 1.9 and 3.3 with the naive zero-point. A group has sixteen levels and one scale, so groups must stay small for the scale to resolve the values it covers. Since each group also carries its own scale and zero-point, finer groups add bytes to the message (Sec. 7.1), and we use $g = 128$.

The second factor is the zero-point. With a naive zero-point the asymmetric quantizer loses to the symmetric grid by 0.73 on WikiText and 1.43 on C4. As shown in Sec. 4.3, the naive map represents zero inexactly, and the error this injects into the densest part of the activation distribution outweighs the gain from the finer resolution of the fitted range. The zero-aware assignment removes this error, and the asymmetric quantizer reaches 6.362 on WikiText and 9.484 on C4, lower than both alternatives.

The third factor is the compensator. Removing it makes each call quantize independently, so the errors accumulate across sites, which is the setting of one-shot compression in prior work. Perplexity rises to 6.730 on WikiText and 10.130 on C4. The accuracy counterpart of this row appears in Table 6. Keeping the compensator but collapsing its

Table 8. Component ablations, Llama-3-8B at $P = 4$, WikiText and C4 perplexity (WT = WikiText), lower is better. Each row changes one component of the method, and the $g = 256$ rows also change the group size. Bold marks the minimum within each bit-width column, excluding the 16-bit reference.

	4-bit		8-bit			4-bit		8-bit	
	WT ↓	C4 ↓	WT ↓	C4 ↓		WT ↓	C4 ↓	WT ↓	C4 ↓
Full method	6.362	9.484	6.070	8.970	<i>Coefficients</i>				
<i>Quantizer</i>					(full: $\eta = 0.75, a = 0.35, (\beta_1, \beta_2) = (0.7, 0.3)$)				
(full: zero-aware asymmetric, $g = 128$)					$\eta = 0.7$	6.365	9.469	6.071	8.972
naive zero-point	7.296	11.260	6.072	8.973	$\eta = 0.9$	6.384	9.525	6.071	8.972
symmetric grid	6.568	9.826	6.075	8.979	$a = 0.25$	6.365	9.467	6.072	8.972
symmetric grid, $g = 256$	6.921	10.611	6.081	8.990	$a = 0.45$	6.367	9.481	6.072	8.971
naive zero-point, $g = 256$	9.204	14.582	6.075	8.978	$(\beta_1, \beta_2) = (0.8, 0.2)$	6.370	9.489	6.071	8.973
<i>Compensator</i>					$(\beta_1, \beta_2) = (0.6, 0.4)$	6.370	9.489	6.071	8.973
(full: dual, two averages)					16-bit reference	6.070	8.966	6.070	8.966
single average ($\beta = 0.4$)	6.506	9.796	6.071	8.972					
no compensation	6.730	10.130	6.079	8.984					

two averages into one loses 0.14 on WikiText and 0.31 on C4, the contribution of the second time scale. At 8 bits the same substitution moves C4 perplexity only from 8.970 to 8.972.

The last rows vary the coefficients. The decay rates and the blend enter the guarantee only through a constant (Sec. 6.3). Consistent with this, every a and (β_1, β_2) row stays within 0.022 of the full method. The gain trades two errors against each other. At $\eta = 1$ nothing is left uncorrected (Eqn. (21)), but the correction is an estimate of an error the compensator has not yet seen, and at full strength an imperfect estimate overcorrects whenever the error changes between sites. A gain below one leaves the fraction $1 - \eta$ of each error uncorrected in exchange for less overcorrection. The theory is stated at $\eta = 1$, and the sweep favors a smaller gain. The gain $\eta = 0.9$ is worse than both 0.7 and 0.75 on WikiText and C4. We use $\eta = 0.75, a = 0.35, \beta_1 = 0.7$, and $\beta_2 = 0.3$ throughout, and nothing is tuned per model.

The 4-bit accuracy of Sec. 7.2 thus depends on both components. The quantizer bounds the error each site injects, and the compensator keeps the remaining error from accumulating.

8 Conclusion

ECHO starts from the arithmetic constraint of a low-bit collective. The all-reduce must add the values it moves, and the sum cannot be formed in the transport format. The collective is therefore built as low-bit transport around a high-precision reduction, which quantizes each element at most twice per call for any tensor-parallel degree. The reduction also reveals each injected error at no extra communication, and the fused compensator feeds it back into later calls, so the accumulated error stays bounded with guarantees independent of network depth. At 8 bits, inference matches 16-bit accuracy, and 4-bit inference stays within 0.4 and 1.7 points of it on average over seven tasks for Qwen3-8B and Llama-3-8B. With a slightly larger learning rate, 4-bit pretraining over 10B tokens ends less than 0.9% above the 16-bit loss. For 1 GB messages the compensated all-reduce is $2.09\times$ faster than the 16-bit NCCL all-reduce.

The end-to-end gains grow with the traffic on the interconnect. The time to first token of Llama-3-8B improves by $1.16\times$ without background load and by $1.44\times$ at 100% load, because communication then takes a larger share of each step. The compensation itself relies on two structural properties. The collective reveals the error it injects, and its outputs join the same residual stream call after call, so a later correction can cancel an earlier error. The reduce-scatter of sequence parallelism and the combine step of expert parallelism share both, and are natural next targets for the compensated collective.

References

- I. Ahmed, C. Schaefer, G. Tabak, D. Vnukov, Z. Zhang, F. Chern, et al. EQuARX: Efficient quantized AllReduce in XLA for distributed machine learning acceleration. *arXiv preprint arXiv:2506.17615*, 2025.
- S. Bian, D. Li, H. Wang, E. P. Xing, and S. Venkataraman. Does compressing activations help model parallel training? *arXiv preprint arXiv:2301.02654*, 2023.

- Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.
- S. Budzinskiy, W. Fang, L. Zeng, and P. Petersen. Numerical stability analysis of large language models. *arXiv preprint arXiv:2503.10251*, 2025.
- L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, et al. FLUX: Fast software-based communication overlap on GPUs through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024.
- P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, et al. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.
- DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- H. Dong, T. Johnson, M. Cho, and E. Soroush. Towards low-bit communication for tensor parallel LLM inference. In *Advances in Neural Information Processing Systems, Workshop on Efficient Natural Language and Speech Processing*, 2024. arXiv:2411.07942.
- E. Frantar, S. Ashkboos, T. Hoeffler, and D. Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.
- L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, et al. A framework for few-shot language model evaluation. Zenodo, Dec. 2023. Version v0.4.0.
- A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer. AI and memory wall. *IEEE Micro*, 44(3):33–39, 2024.
- R. Gond, N. Kwatra, and R. Ramjee. TokenWeave: Efficient compute-communication overlap for distributed LLM inference. In *Proceedings of Machine Learning and Systems*, volume 8, pages 36–56, 2026.
- R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, et al. Scalable hierarchical aggregation protocol (SHaRP): A hardware architecture for efficient data reduction. In *Proceedings of the First International Workshop on Communication Optimizations in HPC*, pages 1–10. IEEE, 2016.
- A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- J. Hansen-Palmus, M. T. Le, O. Hausdörfer, and A. Verma. Communication compression for tensor parallel LLM inference. *arXiv preprint arXiv:2411.09510*, 2024.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, et al. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.
- C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, et al. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, pages 1270–1303, 2024.
- J. Jia, C. Xie, H. Lu, D. Wang, H. Feng, C. Zhang, et al. SDP4Bit: Toward 4-bit communication quantization in sharded data parallelism for LLM training. In *Advances in Neural Information Processing Systems*, volume 37, pages 8734–8759, 2024. doi: 10.52202/079017-0279.
- C. Jin, Z. Jiang, Z. Bai, Z. Zhong, J. Liu, X. Li, et al. MegaScale-MoE: Large-scale communication-efficient training of Mixture-of-Experts models in production. In *Proceedings of the 21st European Conference on Computer Systems*, pages 366–382. ACM, 2026. doi: 10.1145/3767295.3769325.
- S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3252–3261. PMLR, 2019.
- H. Li, W. Luo, X. Niu, S. Tiwari, K. Wang, J. Ren, A. Sirasao, and D. Lehr. QuickReduce: Up to 3x faster all-reduce for vLLM and SGLang. AMD ROCm Blogs, <https://rocm.blogs.amd.com/artificial-intelligence/quick-reduce/README.html>, 2025a.
- Q. Li, B. Zhang, L. Ye, Y. Zhang, W. Wu, Y. Sun, et al. Flash Communication: Reducing tensor parallelization bottleneck for fast large language model inference. *arXiv preprint arXiv:2412.04964*, 2024.
- Q. Li, B. Zhang, H. Kang, T. Xu, Y. Qian, Y. Xie, et al. FlashCommunication V2: Bit splitting and spike reserving for any bit communication. *arXiv preprint arXiv:2508.03760*, 2025b.
- J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, et al. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pages 87–100, 2024.

- M. Liu, X. Liu, X. Tian, B. Lu, S. Lyu, S. Yin, et al. TACO: Efficient communication compression of intermediate tensors for scalable tensor-parallel LLM training. In *Proceedings of the 35th International Symposium on High-Performance Parallel and Distributed Computing*, pages 235–248. ACM, 2026. doi: 10.1145/3806645.3807584.
- Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, et al. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344. PMLR, 2024.
- S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.
- P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.
- M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.
- NVIDIA. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025.
- M. Pagliardini, P. Ablin, and D. Grangier. The AdEMAMix optimizer: Better, faster, older. In *International Conference on Learning Representations*, 2025.
- P. Patarasuk and X. Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.
- A. Patrawala, J. Feng, E. Jones, and J. Steinhardt. LLM layers immediately correct each other. In *Advances in Neural Information Processing Systems*, volume 38, pages 139071–139113, 2025.
- I. Pinelis. Optimum bounds for the distributions of martingales in Banach spaces. *The Annals of Probability*, 22(4):1679–1706, 1994.
- C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
- D. Rein, B. L. Hou, A. Cooper Stickland, J. Petty, R. Y. Pang, J. Dirani, et al. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.
- P. Richtárik, I. Sokolov, and I. Fatkhullin. EF21: A new, simpler, theoretically better, and practically faster error feedback. In *Advances in Neural Information Processing Systems*, volume 34, pages 4384–4396, 2021.
- M. Rudakov, A. Beznosikov, Y. Kholodov, and A. Gasnikov. Activations and gradients compression for model-parallel training. *arXiv preprint arXiv:2401.07788*, 2024.
- K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.
- A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, et al. Scaling distributed machine learning with in-network aggregation. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 785–808. USENIX Association, 2021.
- F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Interspeech*, pages 1058–1062, 2014.
- A. Shen, R. Zhang, and J. Zhao. FlexLink: Boosting your NVLink bandwidth by 27% without accuracy concern. *arXiv preprint arXiv:2510.15882*, 2025.
- M. Shoenberger, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.
- L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15725–15788, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.840.
- W. Sun, X. Song, P. Li, L. Yin, Y. Zheng, and S. Liu. The curse of depth in large language models. In *Advances in Neural Information Processing Systems*, volume 38, pages 181014–181046, 2025.
- H. Tang, S. Gan, A. A. Awan, S. Rajbhandari, C. Li, X. Lian, et al. 1-bit Adam: Communication efficient large-scale training with Adam’s convergence speed. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10118–10129. PMLR, 2021.
- T. Vogels, S. P. Karimireddy, and M. Jaggi. PowerSGD: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 14259–14268, 2019.
- G. Wang, H. Qin, S. A. Jacobs, X. Wu, C. Holmes, Z. Yao, et al. ZeRO++: Extremely efficient collective communication for large model training. In *International Conference on Learning Representations*, 2024a.

- G. Wang, C. Zhang, Z. Shen, A. Li, and O. Ruwase. Domino: Eliminating communication in LLM training via generic tensor slicing and overlapping. *arXiv preprint arXiv:2409.15241*, 2024b.
- H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.
- J. Wang, B. Yuan, L. Rimanic, Y. He, T. Dao, B. Chen, et al. Fine-tuning language models over slow networks using activation quantization with guarantees. In *Advances in Neural Information Processing Systems*, volume 35, pages 19215–19230, 2022.
- Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pages 95266–95290, 2024c. doi: 10.52202/079017-3018.
- B. Widrow and I. Kollár. *Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications*. Cambridge University Press, 2008.
- X. Xie, K. Ding, S. Yan, K.-C. Toh, and T. Wei. Optimization hyper-parameter laws for large language models. *arXiv preprint arXiv:2409.04777*, 2024a.
- X. Xie, P. Zhou, H. Li, Z. Lin, and S. Yan. Adan: Adaptive Nesterov momentum algorithm for faster optimizing deep models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9508–9520, 2024b. doi: 10.1109/TPAMI.2024.3423382.
- X. Xie, Z. Lin, K.-C. Toh, and P. Zhou. LoCo: Low-bit communication adaptor for large-scale model training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(6):4285–4298, 2025.
- X. Xie, Z. Yu, Y. Liao, T. Wang, K.-C. Toh, and S. Yan. Slow-fast inference: Training-free inference acceleration via within-sentence support stability. *arXiv preprint arXiv:2603.12038*, 2026.
- R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, et al. On layer normalization in the Transformer architecture. In *International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020.
- A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.
- C. Zhao, X. Wang, and J. Huang. Scaling LLM inference: Innovations in tensor parallelism, context parallelism, and expert parallelism. Engineering at Meta blog, <https://engineering.fb.com/2025/10/17/ai-research/scaling-llm-inference-innovations-tensor-parallelism-context-parallelism-expert-parallelism/>, Oct. 2025.

A Complete Theoretical Proofs

This appendix proves the results of Sec. 6 in the order in which the section states them: notation and regularity, Theorem 1 together with its gain and variance forms, the partial-sum and coefficient forms behind Eqn. (20), the completion of Assumption 1, Theorem 2, the constructions behind the sharpness discussion of Sec. 6.2, and the backward source bound.

A.1 Notation and regularity

Sites are indexed $k = 0, \dots, N - 1$. The clean trajectory $\mathbf{z}_k$, the deviation $\mathbf{E}_k$, the Jacobians $\mathbf{J}_k$, and the products $\Phi_{n,j} = (\mathbf{I} + \mathbf{J}_{n-1}) \cdots (\mathbf{I} + \mathbf{J}_j)$, with $\Phi_{n,n} = \mathbf{I}$, are those of Eqn. (17). The scale is $\sigma_k^2 = \|\mathbf{z}_k\|^2/d$, with increment $\Delta_k = \sigma_{k+1}^2 - \sigma_k^2$. Three per-block quantities enter the constants, all uniform over the class of trained checkpoints considered, as is the budget Λ of Assumption 1:

$$(R1) \quad 0 < \Delta_{\min} \leq \Delta_k \leq \Delta_{\max},$$

$$(R2) \quad \|\mathbf{J}_k\|_F^2/d \leq C_0 \Delta_k/\sigma_k^2,$$

$$(R3) \quad H_k \leq \bar{h}/(\sigma_k^2 \sqrt{d}),$$

with H_k the second-derivative bound of Eqn. (16). R1 records that every site adds a bounded, nonvanishing amount of energy to the hidden state, and R2 and R3 record the scaling that the normalization in front of each block imposes on its derivatives. A bound is called depth independent when its constant is built from $\Delta_{\min}$, $\Delta_{\max}$, C_0 , $\bar{h}$, the scales σ_0, σ_1 , the compensator constants μ_w, τ_2 , the budget Λ of Assumption 1, and possibly the hidden dimension d , and from nothing that grows with N .

With $\mathcal{F}_s$ the history of the injections up to and including site s , the noise model is

$$\mathbb{E}[\delta_s \mid \mathcal{F}_{s-1}] = \mathbf{0}, \quad \mathbb{E}[\delta_s \delta_s^\top \mid \mathcal{F}_{s-1}] \preceq \frac{q^2}{d} \mathbf{I}, \quad \|\delta_s\| \leq q \text{ a.s.} \quad (22)$$

The clean trajectory is fixed, so the $\mathbf{J}_k$ and $\Phi_{n,j}$ are deterministic and all probability is over the injected errors.

A.2 The source bound

Lemma 1 (tail representation). *At gain $\eta = 1$, $\mathbf{S}_t = \sum_{s \leq t} T_{t-s} \delta_s$.*

Proof. Equation (13) gives the coefficients $\phi_m = 1 - \eta \sum_{r < m} w_r$. At $\eta = 1$, since $\sum_r w_r = 1$,

$$\phi_m = 1 - \sum_{r < m} w_r = \sum_{r \geq m} w_r = T_m. \quad \square$$

Proof of Theorem 1. The T_m are nonnegative, so for every sequence with $\|\delta_k\| \leq q$,

$$\|\mathbf{S}_t\| \leq \sum_{s \leq t} T_{t-s} \|\delta_s\| \leq q \sum_{m \geq 0} T_m = q \sum_{m \geq 0} \sum_{r \geq m} w_r = q \sum_{r \geq 0} (r+1) w_r = (1 + \mu_w) q.$$

Without compensation $\phi_m \equiv 1$, and for any fixed unit vector $\mathbf{e}$ the constant sequence $\delta_k = q \mathbf{e}$ attains

$$\|\mathbf{S}_{N-1}\| = \left\| \sum_{k < N} q \mathbf{e} \right\| = Nq.$$

At gain $\eta < 1$ the coefficient is $\phi_m = (1 - \eta) + \eta T_m$, and the same estimate gives

$$\max_t \|\mathbf{S}_t\| \leq (1 - \eta) Nq + \eta (1 + \mu_w) q,$$

which is Eqn. (21). As $\beta_1, \beta_2 \rightarrow 0$ the kernel concentrates at age zero, $T_0 = 1$, $T_{m \geq 1} = 0$, and the bound is q . $\square$

Remark 1 (source variance). *Under the noise model the cross terms vanish, since for $s < s'$, $\mathbb{E}\langle \delta_s, \delta_{s'} \rangle = \mathbb{E}\langle \delta_s, \mathbb{E}[\delta_{s'} | \mathcal{F}_{s'-1}] \rangle = 0$. Hence, by Lemma 1,*

$$\mathbb{E} \|\mathbf{S}_t\|^2 = \sum_{s \leq t} T_{t-s}^2 \mathbb{E} \|\delta_s\|^2 \leq q^2 \sum_{m \geq 0} T_m^2 = \tau_2 q^2,$$

the variance form quoted after Theorem 2. At gain η the same computation with $\phi_m = (1 - \eta) + \eta T_m$ gives $\mathbb{E} \|\mathbf{S}_t\|^2 \leq q^2 \sum_{m \leq t} \phi_m^2$.

A.3 The linearized deviation

The linearized deviation is $\mathbf{E}_N^{\text{lin}} = \sum_t \Phi_{N,t+1} \rho_t$, the first-order part of Eqn. (17). From here on the gain is $\eta = 1$.

Lemma 2 (partial-sum form).

$$\mathbf{E}_N^{\text{lin}} = \mathbf{S}_{N-1} + \sum_{t=0}^{N-2} \Phi_{N,t+2} \mathbf{J}_{t+1} \mathbf{S}_t.$$

Proof. Since $\rho_t = \mathbf{S}_t - \mathbf{S}_{t-1}$ with $\mathbf{S}_{-1} = \mathbf{0}$ (Eqn. (6)), summation by parts with $\Phi_{N,t+1} - \Phi_{N,t+2} = \Phi_{N,t+2} \mathbf{J}_{t+1}$ and $\Phi_{N,N} = \mathbf{I}$ gives

$$\sum_{t=0}^{N-1} \Phi_{N,t+1} (\mathbf{S}_t - \mathbf{S}_{t-1}) = \Phi_{N,N} \mathbf{S}_{N-1} + \sum_{t=0}^{N-2} (\Phi_{N,t+1} - \Phi_{N,t+2}) \mathbf{S}_t. \quad \square$$

The deviation therefore depends on the injected errors only through their partial sums, the form promised in Sec. 6.1. The final partial sum is carried by the identity path to the output, and each earlier partial sum enters through the Jacobian of the site that follows it and is then propagated forward. This is where the compensator enters the output bound, through the partial sums $\mathbf{S}_t$.

Lemma 3 (coefficient matrices).

$$\mathbf{E}_N^{\text{lin}} = \sum_{s=0}^{N-1} \mathbf{K}_s \delta_s, \quad \mathbf{K}_s = T_{N-1-s} \mathbf{I} + \sum_{t=s}^{N-2} T_{t-s} \Phi_{N,t+2} \mathbf{J}_{t+1}.$$

Proof. Substitute $\mathbf{S}_t = \sum_{s \leq t} T_{t-s} \delta_s$ (Lemma 1) into Lemma 2 and collect the coefficient of δ_s : the term $\mathbf{S}_{N-1}$ contributes $T_{N-1-s} \mathbf{I}$, and the term at $t \geq s$ contributes $T_{t-s} \Phi_{N,t+2} \mathbf{J}_{t+1}$. $\square$

The first term of $\mathbf{K}_s$ is the tail of the error not yet returned by the end of the pass, carried by the identity path. The second collects the corrections applied at the later sites, each entering through the block Jacobian there and then propagated to the output.

Lemma 4 (mean square). *Under Eqn. (22), Eqn. (20) holds: $\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2 \leq (q^2/d) \sum_s \|\mathbf{K}_s\|_F^2$.*

Proof. The $\mathbf{K}_s$ are deterministic, so conditioning on $\mathcal{F}_{t-1}$,

$$\mathbb{E}\langle \mathbf{K}_s \delta_s, \mathbf{K}_t \delta_t \rangle = \mathbb{E}\langle \mathbf{K}_s \delta_s, \mathbf{K}_t \mathbb{E}[\delta_t | \mathcal{F}_{t-1}] \rangle = 0, \quad s < t.$$

Hence

$$\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2 = \sum_s \mathbb{E} \|\mathbf{K}_s \delta_s\|^2 = \sum_s \mathbb{E} \text{tr}(\mathbf{K}_s^\top \mathbf{K}_s \delta_s \delta_s^\top) \leq \frac{q^2}{d} \sum_s \|\mathbf{K}_s\|_F^2. \quad \square$$

A.4 The completed assumption

The statement in Sec. 6.2 covers the deviations $\Phi_{k,s+1} \delta_s$, with covariances proportional to $(q^2/d) \Phi_{k,s+1} \Phi_{k,s+1}^\top$. The proofs apply the same inequality to two further collections produced by the same propagation. The completed family $\mathcal{C}_k$ is the closed convex hull of the normalizations of

$$\Phi_{k,\ell} \Phi_{k,\ell}^\top, \quad \Phi_{k,t+2} \mathbf{J}_{t+1} \mathbf{J}_{t+1}^\top \Phi_{k,t+2}^\top, \quad \Phi_{k,j+1} \mathbb{E}[\mathbf{r}_j^\top (\mathbf{r}_j^\top)^\top] \Phi_{k,j+1}^\top, \quad (23)$$

over $\ell \leq k$, $t+2 \leq k$, and $j+1 \leq k$, with $\mathbf{r}_j^\tau$ the stopped remainder of Proposition 1 below, included at the noise level q in use. Every member is the covariance of a deviation that injected noise, or its second-order response, produces at site k . The third collection varies with the truncation threshold of Proposition 1, which depends on p and N , and the assumption is understood to hold with the same budget Λ for every truncation threshold. For $\Sigma = \mathbb{E} \mathbf{v} \mathbf{v}^\top$ the inequality of Assumption 1 is

$$\text{tr}((\mathbf{I} + \mathbf{J}_k)^\top (\mathbf{I} + \mathbf{J}_k) \Sigma) \leq (1 + \varepsilon_k) \frac{\sigma_{k+1}^2}{\sigma_k^2} \text{tr} \Sigma, \quad \Sigma \in \mathcal{C}_k. \quad (24)$$

Lemma 5 (propagation of covariances). *Let Σ be a member of one of the three collections in Eqn. (23) at site j . Then for $j \leq n \leq N$,*

$$\frac{\text{tr}(\Phi_{n,j} \Sigma \Phi_{n,j}^\top)}{\sigma_n^2} \leq e^\Lambda \frac{\text{tr} \Sigma}{\sigma_j^2}.$$

Proof. Set $\Sigma_m = \Phi_{m,j} \Sigma \Phi_{m,j}^\top$, so that $\Sigma_{m+1} = (\mathbf{I} + \mathbf{J}_m) \Sigma_m (\mathbf{I} + \mathbf{J}_m)^\top$. Since $\Phi_{m,j} \Phi_{j,\ell} = \Phi_{m,\ell}$, each Σ_m again belongs to the same collection of Eqn. (23) at site m , so Eqn. (24) applies at every step,

$$\frac{\text{tr} \Sigma_{m+1}}{\sigma_{m+1}^2} \leq (1 + \varepsilon_m) \frac{\text{tr} \Sigma_m}{\sigma_m^2},$$

and $\prod_m (1 + \varepsilon_m) \leq e^{\sum_m \varepsilon_m} \leq e^\Lambda$. □

Lemma 6 (propagated sources). *Under Assumption 1,*

$$\sum_{t=0}^{N-2} \frac{\|\Phi_{N,t+2} \mathbf{J}_{t+1}\|_F^2}{d} \leq e^\Lambda C_0 \frac{\sigma_N^2}{\sigma_1^2}.$$

Proof. The matrix $\mathbf{J}_{t+1} \mathbf{J}_{t+1}^\top$ is the member of the second collection at site $j = t+2$, the case $k = t+2$, where $\Phi_{t+2,t+2} = \mathbf{I}$. Lemma 5, then R2 and $\Delta_{t+1}/(\sigma_{t+1}^2 \sigma_{t+2}^2) = 1/\sigma_{t+1}^2 - 1/\sigma_{t+2}^2$, give

$$\frac{\|\Phi_{N,t+2} \mathbf{J}_{t+1}\|_F^2}{\sigma_N^2} \leq e^\Lambda \frac{\|\mathbf{J}_{t+1}\|_F^2}{\sigma_{t+2}^2} \leq e^\Lambda C_0 d \left(\frac{1}{\sigma_{t+1}^2} - \frac{1}{\sigma_{t+2}^2} \right),$$

and the sum over t telescopes to at most $e^\Lambda C_0 d / \sigma_1^2$. □

A.5 Proof of the propagation theorem

Proof of Theorem 2, part (i). Write $f_t = \|\Phi_{N,t+2} \mathbf{J}_{t+1}\|_F$. Taking Frobenius norms in Lemma 3,

$$\|\mathbf{K}_s\|_F \leq \sqrt{d} T_{N-1-s} + \sum_{t \geq s} T_{t-s} f_t.$$

Squaring, summing over s , and applying Young's inequality $\|T * f\|_{\ell_2} \leq \|T\|_{\ell_1} \|f\|_{\ell_2}$ with $\|T\|_{\ell_1} = 1 + \mu_w$, then Lemma 6,

$$\sum_s \|\mathbf{K}_s\|_F^2 \leq 2d\tau_2 + 2(1 + \mu_w)^2 \sum_t f_t^2 \leq 2d\tau_2 + 2(1 + \mu_w)^2 d e^\Lambda C_0 \frac{\sigma_N^2}{\sigma_1^2}.$$

Inserting into Lemma 4 and dividing by $d\sigma_N^2$,

$$\frac{\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} \leq \frac{q^2}{d^2 \sigma_N^2} \left[2d\tau_2 + 2(1 + \mu_w)^2 d e^\Lambda C_0 \frac{\sigma_N^2}{\sigma_1^2} \right] = 2 \left[\tau_2 \frac{\sigma_1^2}{\sigma_N^2} + (1 + \mu_w)^2 C_0 e^\Lambda \right] \bar{q}^2. \quad \square$$

Remark 2 (without compensation). *Here $\rho_s = \delta_s$ and $\mathbf{K}_s = \Phi_{N,s+1}$. The identity is the member of the first collection of Eqn. (23) at site $s+1$ (the case $\ell = k$), so Lemma 5 gives $\|\Phi_{N,s+1}\|_F^2 \leq e^\Lambda d \sigma_N^2 / \sigma_{s+1}^2$, and Lemma 4*

with $\sigma_{s+1}^2 \geq \sigma_1^2 + s\Delta_{\min}$ (RI) gives

$$\frac{\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} \leq e^\Lambda \bar{q}^2 \sum_{s=0}^{N-1} \frac{\sigma_1^2}{\sigma_{s+1}^2} \leq e^\Lambda \bar{q}^2 \left[1 + \frac{\sigma_1^2}{\Delta_{\min}} \log \left(1 + \frac{(N-1)\Delta_{\min}}{\sigma_1^2} \right) \right].$$

Each site enters with the weight $1/\sigma_{s+1}^2$, and the sum grows as $\log N$. With compensation the same weight enters only through Lemma 6, multiplied by $C_0 \Delta_{t+1}/\sigma_{t+2}^2 \leq C_0 \Delta_{t+1}/\sigma_{t+1}^2$, and the sum telescopes to $e^\Lambda C_0 d/\sigma_1^2$, which gives the term $(1 + \mu_w)^2 C_0 e^\Lambda$ of part (i).

Corollary 1 (general gain). *Under the hypotheses of Theorem 2, at gain $\eta \in [0, 1]$,*

$$\frac{\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} \leq 4\eta^2 \left[\tau_2 \frac{\sigma_1^2}{\sigma_N^2} + (1 + \mu_w)^2 C_0 e^\Lambda \right] \bar{q}^2 + 2(1 - \eta)^2 e^\Lambda \bar{q}^2 \sum_{s=0}^{N-1} \frac{\sigma_1^2}{\sigma_{s+1}^2}.$$

Proof. By Eqn. (12) the correction is linear in the gain, $\mathbf{C}_k = \eta \mathbf{C}_k^{(1)}$ with $\mathbf{C}_k^{(1)}$ the correction at $\eta = 1$, so $\rho_k = (1 - \eta) \delta_k + \eta (\delta_k + \mathbf{C}_k^{(1)})$. The linearized deviation is linear in the net errors, so it is $1 - \eta$ times the uncompensated deviation plus η times the deviation at $\eta = 1$, both driven by the same injections. The bound follows from $(x + y)^2 \leq 2x^2 + 2y^2$, part (i), and Remark 2. $\square$

Proof of part (ii). The sums $\mathbf{M}_n = \sum_{s \leq n} \mathbf{K}_s \delta_s$ form a martingale in $\mathbb{R}^d$. Its predictable quadratic variation is at most the deterministic majorant

$$v_N := \frac{q^2}{d} \sum_s \|\mathbf{K}_s\|_F^2, \quad \|\mathbf{K}_s \delta_s\| \leq \zeta_N := q \max_s \|\mathbf{K}_s\|.$$

By the first display in the proof of Lemma 6, with $\Delta_{t+1} \leq \Delta_{\max}$,

$$\|\Phi_{N,t+2} \mathbf{J}_{t+1}\| \leq \sigma_N \frac{\sqrt{d e^\Lambda C_0 \Delta_{\max}}}{\sigma_{t+1} \sigma_{t+2}},$$

so, by Lemma 3, $T_m \leq 1$, $\sum_m T_m = 1 + \mu_w$, and $\sigma_{t+1} \sigma_{t+2} \geq \sigma_1^2$,

$$\|\mathbf{K}_s\| \leq 1 + (1 + \mu_w) \frac{\sigma_N}{\sigma_1^2} \sqrt{d e^\Lambda C_0 \Delta_{\max}}.$$

The Bennett–Bernstein inequality for Hilbert-space martingales (Pinelis, 1994), applied with v_N and ζ_N , gives

$$\Pr \left[\max_n \|\mathbf{M}_n\| \geq \sqrt{2v_N \lambda_p} + \frac{2}{3} \zeta_N \lambda_p \right] \leq p, \quad \lambda_p = \log \frac{2}{p}.$$

On the complementary event, $(x + y)^2 \leq 2x^2 + 2y^2$ gives $\|\mathbf{E}_N^{\text{lin}}\|^2 \leq 4v_N \lambda_p + \frac{8}{9} \zeta_N^2 \lambda_p^2$. Dividing by $d\sigma_N^2$ and using part (i) for v_N and the display above for ζ_N ,

$$\begin{aligned} \frac{\|\mathbf{E}_N^{\text{lin}}\|^2}{d\sigma_N^2} &\leq C \bar{q}^2 \lambda_p + C' \bar{q}^2 \lambda_p^2, \quad C := 8 \left[\tau_2 \frac{\sigma_1^2}{\sigma_N^2} + (1 + \mu_w)^2 C_0 e^\Lambda \right], \\ C' &:= \frac{8}{9} \left(\frac{\sigma_1}{\sigma_N} + (1 + \mu_w) \frac{\sqrt{d e^\Lambda C_0 \Delta_{\max}}}{\sigma_1} \right)^2, \end{aligned}$$

since $\zeta_N/(\sqrt{d} \sigma_N) \leq \bar{q} \sigma_1/\sigma_N + \bar{q} (1 + \mu_w) \sqrt{d e^\Lambda C_0 \Delta_{\max}}/\sigma_1$. Both constants are depth independent. The factor d inside C' comes from bounding the operator norm of $\Phi_{N,t+2} \mathbf{J}_{t+1}$ by its Frobenius norm, and it multiplies only the $\log^2$ term.

The statement is for the fixed endpoint N . For a prefix endpoint n the same proof gives the constants with σ_n in place of σ_N , and since $\sigma_1/\sigma_n \leq 1$, all of them are dominated by the uniform constants

$$\bar{C} := 8[\tau_2 + (1 + \mu_w)^2 C_0 e^\Lambda], \quad \bar{C}' := \frac{8}{9} \left(1 + (1 + \mu_w) \frac{\sqrt{d e^\Lambda C_0 \Delta_{\max}}}{\sigma_1} \right)^2.$$

Covering all prefixes at level p/N replaces λ_p by $\log(2N/p)$ via a union bound. $\square$

Proposition 1 (second-order remainder). *Under the hypotheses of Theorem 2, fix $p \in (0, 1)$, let Θ_n be the bound that the proof of Theorem 2(ii) gives for the prefix ending at site n , at level p/N , let τ be the first site with $\|\mathbf{E}_n\| > 2\Theta_n$ ($\tau = N$ if there is none), and set $\mathbf{r}_n^\tau = \mathbf{r}_n$ for $n < \tau$ and $\mathbf{0}$ otherwise. Then*

$$\left(\mathbb{E} \left\| \sum_{n < N} \Phi_{N,n+1} \mathbf{r}_n^\tau \right\|^2 \right)^{1/2} \leq C_2 \bar{q}^2 \sqrt{N} \sqrt{d} \sigma_N,$$

with C_2 growing at most as $\log^2(N/p)$. The sum equals the second-order part $\mathbf{E}_N - \mathbf{E}_N^{\text{lin}}$ whenever no site exceeds the threshold.

Proof of Proposition 1. Set $\kappa_p^2 := \bar{C} \log(2N/p) + \bar{C}' \log^2(2N/p)$, the uniform constant with which part (ii), applied at level p/N and extended to all prefixes, bounds every prefix deviation, so that $\Theta_n = \kappa_p \bar{q} \sqrt{d} \sigma_n$, with τ and $\mathbf{r}_n^\tau$ as in the statement. The truncated sum $\sum_{n < N} \Phi_{N,n+1} \mathbf{r}_n^\tau$ has the deterministic endpoint N , and on the event $\tau = N$ it equals $\mathbf{E}_N - \mathbf{E}_N^{\text{lin}}$, the second-order part of Eqn. (17).

For $n < \tau$, R3 and the definition of τ give

$$\|\mathbf{r}_n^\tau\| \leq \frac{1}{2} H_n \|\mathbf{E}_n\|^2 \leq \frac{\bar{h}}{2\sigma_n^2 \sqrt{d}} (2\Theta_n)^2 = 2\bar{h} \kappa_p^2 \bar{q}^2 \sqrt{d}.$$

The covariances of the $\mathbf{r}_n^\tau$ form the third collection of Eqn. (23), so Lemma 5, Minkowski's inequality, and $\sum_n 1/\sigma_{n+1} \leq 2\sigma_N/\Delta_{\min}$ (from R1) give

$$\left\| \sum_{n < N} \Phi_{N,n+1} \mathbf{r}_n^\tau \right\|_{L^2} \leq \sum_n e^{\Lambda/2} \frac{\sigma_N}{\sigma_{n+1}} \|\mathbf{r}_n^\tau\|_{L^2} \leq 2\bar{h} \kappa_p^2 e^{\Lambda/2} \bar{q}^2 \sqrt{d} \frac{2\sigma_N^2}{\Delta_{\min}}.$$

Since $\sigma_N \leq \sigma_1 + \sqrt{\Delta_{\max} N} \leq (\sigma_1 + \sqrt{\Delta_{\max}}) \sqrt{N}$,

$$\frac{4\bar{h} \kappa_p^2 e^{\Lambda/2}}{\Delta_{\min}} \bar{q}^2 \sqrt{d} \sigma_N^2 \leq \underbrace{\frac{4\bar{h} e^{\Lambda/2} (\sigma_1 + \sqrt{\Delta_{\max}})}{\Delta_{\min}} \kappa_p^2}_{=: C_2} \bar{q}^2 \sqrt{N} \sqrt{d} \sigma_N. \quad \square$$

Corollary 2 (second-order budget). *If in addition*

$$\sum_n \frac{(\mathbb{E} \|\mathbf{r}_n^\tau\|^2)^{1/2}}{\sqrt{d} \sigma_{n+1}} \leq \Lambda_2 \bar{q}^2 \quad (25)$$

with Λ_2 independent of N , then

$$\left(\mathbb{E} \left\| \sum_{n < N} \Phi_{N,n+1} \mathbf{r}_n^\tau \right\|^2 \right)^{1/2} \leq \sum_n e^{\Lambda/2} \frac{\sigma_N}{\sigma_{n+1}} \|\mathbf{r}_n^\tau\|_{L^2} \leq e^{\Lambda/2} \Lambda_2 \bar{q}^2 \sqrt{d} \sigma_N,$$

with no factor $\sqrt{N}$.

Condition (25) asks that the total second-order response along a pass stay at the level of a single response. Standard post-training bias correction shows that quantization-induced mean shifts can be compensated explicitly (Nagel et al., 2019), which motivates the mean part of this budget without proving it.

A.6 Sharpness

Proposition 2 (sharpness). (a) *There is a network satisfying R1–R3 and the noise model in which $\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2 / (d\sigma_N^2)$ grows exponentially in $\sqrt{N}$. Its budget Λ in Assumption 1 grows as $\sqrt{N}$.* (b) *There is a network satisfying R1–R3, the noise model, and Eqn. (24) with every $\varepsilon_k = 0$ for every positive semidefinite Σ , in which the stopped remainder of Proposition 1 attains the order $\bar{q}^2 \sqrt{N}$.*

Proof of (a). Fix orthonormal vectors $\mathbf{e}, \mathbf{e}_0$, a constant $c > 0$, and the scale path $\sigma_k^2 = \sigma_0^2 + k\Delta$, and take the affine blocks

$$F_k(\mathbf{x}) = \frac{c}{\sigma_k} \mathbf{e} \mathbf{e}^\top \mathbf{x} + \sqrt{d} (\sigma_{k+1} - \sigma_k) \mathbf{e}_0$$

on the clean ray $\mathbf{z}_k = \sqrt{d} \sigma_k \mathbf{e}_0$. Then $\mathbf{z}_{k+1} = \mathbf{z}_k + F_k(\mathbf{z}_k)$ stays on the ray,

$$\mathbf{J}_k = \frac{c}{\sigma_k} \mathbf{e} \mathbf{e}^\top, \quad (\mathbf{I} + \mathbf{J}_k) \mathbf{e} = \left(1 + \frac{c}{\sigma_k}\right) \mathbf{e},$$

and $H_k = 0$, so R3 is trivial and R1 and R2 hold with constants depending on c, Δ , and d . Let the injected errors be independent and uniform on the sphere of radius q , so the noise model holds with conditional covariance exactly $(q^2/d) \mathbf{I}$ and Lemma 4 is an equality.

In Lemma 3 all terms of $\mathbf{K}_s \mathbf{e}$ carry one sign, so for an early site s ,

$$\|\mathbf{K}_s \mathbf{e}\| \geq T_0 \frac{c}{\sigma_{s+1}} \prod_{\ell=s+2}^{N-1} \left(1 + \frac{c}{\sigma_\ell}\right), \quad \sum_{\ell} \log\left(1 + \frac{c}{\sigma_\ell}\right) \asymp 2c\sqrt{N/\Delta}.$$

Hence

$$\mathbb{E} \|\mathbf{E}_N^{\text{lin}}\|^2 = \frac{q^2}{d} \sum_{s'} \|\mathbf{K}_{s'}\|_F^2 \geq \frac{q^2}{d} \|\mathbf{K}_s \mathbf{e}\|^2,$$

which grows exponentially in $\sqrt{N}$ even after dividing by $d\sigma_N^2$.

It remains to check that the network violates Assumption 1. An injection at site s produces at site k the deviation $\mathbf{v} = \Phi_{k,s+1} \delta_s$ with covariance proportional to $\Phi_{k,s+1} \Phi_{k,s+1}^\top = \mathbf{I} - \mathbf{e} \mathbf{e}^\top + \pi_k^2 \mathbf{e} \mathbf{e}^\top$, where $\pi_k = \prod_{\ell=s+1}^{k-1} (1 + c/\sigma_\ell) \rightarrow \infty$. Since $\mathbf{I} + \mathbf{J}_k$ fixes $\mathbf{e}^\perp$ and stretches $\mathbf{e}$ by $1 + c/\sigma_k$,

$$\frac{\mathbb{E} \|(\mathbf{I} + \mathbf{J}_k) \mathbf{v}\|^2}{\mathbb{E} \|\mathbf{v}\|^2} = \frac{(d-1) + \pi_k^2 (1 + c/\sigma_k)^2}{(d-1) + \pi_k^2} \geq 1 + \frac{2c}{\sigma_k} \cdot \frac{\pi_k^2}{(d-1) + \pi_k^2} \geq 1 + \frac{c}{\sigma_k}$$

once $\pi_k^2 \geq d$. The assumption then requires $(1 + \varepsilon_k)(1 + \Delta/\sigma_k^2) \geq 1 + c/\sigma_k$, so at every site with $\pi_k^2 \geq d$ and $\sigma_k \geq \max(2\Delta/c, \sqrt{\Delta})$,

$$\varepsilon_k \geq \frac{c/\sigma_k - \Delta/\sigma_k^2}{1 + \Delta/\sigma_k^2} \geq \frac{c}{4\sigma_k}.$$

Both conditions hold from a fixed site on, with $\pi_k \rightarrow \infty$ at $s = 0$ and $\sigma_k \rightarrow \infty$, and the sum over the remaining sites, $\sum_k c/(4\sigma_k) \asymp (c/2)\sqrt{N/\Delta}$, grows as $\sqrt{N}$, so no Λ independent of N exists. Conversely, $(\mathbf{I} + \mathbf{J}_k)^\top (\mathbf{I} + \mathbf{J}_k) \preceq (1 + c/\sigma_k)^2 \mathbf{I}$ gives $\varepsilon_k \leq 2c/\sigma_k + c^2/\sigma_k^2$, so the budget grows exactly as $\sqrt{N}$. $\square$

Proof of (b). Take orthonormal $\mathbf{e}_0, \mathbf{f}$, $\mathbf{\Pi} = \mathbf{I} - \mathbf{e}_0 \mathbf{e}_0^\top$, $n(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|$, a constant $h > 0$, and the 0-homogeneous blocks

$$F_k(\mathbf{x}) = b_k n(\mathbf{x}) + \frac{1}{2} h \sqrt{d} \|\mathbf{\Pi} n(\mathbf{x})\|^2 \mathbf{f}, \quad b_k = \sqrt{d} (\sigma_{k+1} - \sigma_k),$$

on the clean ray $\mathbf{z}_k = \sqrt{d} \sigma_k \mathbf{e}_0$, $\sigma_k^2 = \sigma_0^2 + k\Delta$. At the clean point the quadratic term has zero first derivative, and

$$\mathbf{J}_k = \alpha_k \mathbf{\Pi}, \quad 1 + \alpha_k = \frac{\sigma_{k+1}}{\sigma_k}, \quad (\mathbf{I} + \mathbf{J}_k)^\top (\mathbf{I} + \mathbf{J}_k) = \mathbf{I} + \frac{\Delta}{\sigma_k^2} \mathbf{\Pi} \preceq \frac{\sigma_{k+1}^2}{\sigma_k^2} \mathbf{I},$$

so Eqn. (24) holds with $\varepsilon_k = 0$ for every positive semidefinite Σ , and R1–R3 hold.

Inject one error at site 0, uniform on the sphere of radius q , and none later. Its tangential part is carried at the critical gain, $\Phi_{n,k}|_{\mathbf{\Pi}} = (\sigma_n/\sigma_k) \mathbf{\Pi}$, so $\mathbf{X}_k := \mathbf{\Pi} \mathbf{E}_k^{\text{lin}}/\sigma_k$ satisfies $\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{\Pi} \rho_k/\sigma_{k+1}$. For a single injection Eqn. (12) gives $\rho_0 = \delta_0$ and $\rho_k = -w_{k-1} \delta_0$ for $k \geq 1$, so $\mathbf{X}_k = g_k \mathbf{\Pi} \delta_0$ with

$$g_k = \frac{1}{\sigma_1} - \sum_{t=1}^{k-1} \frac{w_{t-1}}{\sigma_{t+1}} = \frac{T_{k-1}}{\sigma_k} + \sum_{t=0}^{k-2} T_t \left(\frac{1}{\sigma_{t+1}} - \frac{1}{\sigma_{t+2}} \right),$$

where the second form follows by summation by parts. Since $g_{k+1} - g_k = -w_{k-1}/\sigma_{k+1} \leq 0$, the sequence decreases toward $g_\infty = \sum_{t \geq 0} T_t(1/\sigma_{t+1} - 1/\sigma_{t+2}) > 0$, so $g_k \geq g_\infty$ for every k . Along $\mathbf{e}_0$ nothing is propagated, so only the tail of the injected error remains, $\langle \mathbf{e}_0, \mathbf{E}_k^{\text{lin}} \rangle = T_{k-1} \langle \mathbf{e}_0, \boldsymbol{\delta}_0 \rangle$.

The $\mathbf{f}$ -component of the second-order response at site k collects two curvatures. The quadratic term of the block gives

$$\frac{h}{2\sigma_k^2 \sqrt{d}} \|\Pi \mathbf{E}_k^{\text{lin}}\|^2 (1 + O(q)) = \frac{h}{2\sqrt{d}} g_k^2 \|\Pi \boldsymbol{\delta}_0\|^2 (1 + O(q)),$$

terms of one sign carrying the same random factor $\|\Pi \boldsymbol{\delta}_0\|^2$, of mean $q^2(d-1)/d$. The curvature of $n(\mathbf{x})$ in the $b_k n(\mathbf{x})$ term adds the cross term

$$-\frac{b_k}{d\sigma_k^2} \langle \mathbf{f}, \mathbf{E}_k^{\text{lin}} \rangle \langle \mathbf{e}_0, \mathbf{E}_k^{\text{lin}} \rangle (1 + O(q)),$$

of the same order in q but carrying the factor T_{k-1} , which decays geometrically in k , and of mean zero. In the sums below the cross terms therefore contribute $O(1)$ against $\asymp \sqrt{N}$ one-sign terms, and the quadratic term dominates. The remaining second-order components of $\mathbf{r}_k$ carry an extra factor $1/\sigma_k$ or T_{k-1} . Those along $\mathbf{e}_0$ are orthogonal to $\mathbf{f}$, the $\mathbf{f}$ -part of those along $\Pi \boldsymbol{\delta}_0$ is the cross term above, and neither changes the sums. For each N take q small enough that the deviation stays within its linear bound, so the truncation is never active and $\mathbf{r}_n^\tau = \mathbf{r}_n$. This is possible because $\mathbf{E}_n^{\text{lin}}$ and the threshold Θ_n both scale linearly in q while the remainder is quadratic. The propagated responses then add coherently along $\mathbf{f}$,

$$\sum_n \frac{(\mathbb{E} \|\mathbf{r}_n^\tau\|^2)^{1/2}}{\sqrt{d} \sigma_{n+1}} \asymp \bar{q}^2 \sqrt{N}, \quad \frac{1}{d\sigma_N^2} \mathbb{E} \left\| \sum_n \Phi_{N,n+1} \mathbf{r}_n^\tau \right\|^2 \asymp \bar{q}^4 N.$$

The network satisfies every hypothesis of Theorem 2 with $\varepsilon_k = 0$, so no argument from those hypotheses alone can remove the $\sqrt{N}$ of Proposition 1. $\square$

A.7 The backward source bound

On the backward pass the compensator runs the same recurrence along the backward site sequence (Sec. 5.4), so at gain one the correction applied at a site is again built from the errors injected at the sites visited before it, with the weights of Eqn. (12). We keep the forward index throughout. The backward pass visits the sites in decreasing order of k , so the error injected at site s is corrected at the sites $k = s-1, s-2, \dots$, where $\sigma_k \leq \sigma_s$. What changes is the point of entry. An error or a correction handled at site k reaches the gradient of the hidden state through the Jacobian of the normalization at that site (Sec. 3.2). Write this operator at site k as $\mathbf{A}_k = \mathbf{D}_k/\sigma_k$ with $\|\mathbf{D}_k\| \leq \bar{\gamma}$, where $\bar{\gamma}$ is a constant of the order of the largest magnitude among the normalization weights. The error $\boldsymbol{\delta}_s$ then enters through $\mathbf{A}_s$ at its own site, while the fraction $w_{s-1-k} \boldsymbol{\delta}_s$ subtracted at a later visited site $k < s$ enters through $\mathbf{A}_k$. Once the backward pass has reached site t , the compensated partial sum is

$$\mathbf{S}_t^{\text{bwd}} = \sum_{s \geq t} \mathbf{B}_{s,t} \boldsymbol{\delta}_s, \quad \mathbf{B}_{s,t} = \mathbf{A}_s - \sum_{k=t}^{s-1} w_{s-1-k} \mathbf{A}_k.$$

A correction now leaves through a different operator than the one its error entered through, so the cancellation is no longer exact, and the bound depends on how fast the operators vary along the trajectory.

Proposition 3 (backward source). *Suppose, in addition to R1, that the backward injections satisfy the noise model of Eqn. (22) with the history taken in the order in which the backward pass visits the sites and with q their bound in gradient units, and that the operators vary smoothly along the trajectory, $\|\mathbf{D}_s - \mathbf{D}_k\| \leq c_D (s-k) \sqrt{\Delta_{\max}}/\sigma_s$ for $k \leq s$. Then*

$$\max_t \mathbb{E} \|\mathbf{S}_t^{\text{bwd}}\|^2 \leq C_{\text{bwd}} \frac{q^2}{\sigma_1^2},$$

with C_{bwd} independent of N , built from $\bar{\gamma}$, c_D , the kernel moments μ_w and τ_2 , and the scale constants $\sigma_0, \sigma_1, \Delta_{\min}, \Delta_{\max}$.

Proof. Since $\sum_r w_r = 1$,

$$\mathbf{B}_{s,t} = \mathbf{A}_s T_{s-t} + \sum_{k=t}^{s-1} w_{s-1-k} (\mathbf{A}_s - \mathbf{A}_k).$$

For $k < s$, R1 gives $\sigma_s^2 - \sigma_k^2 \leq (s-k)\Delta_{\max}$, hence $\sigma_s - \sigma_k \leq (s-k)\Delta_{\max}/\sigma_s$, and with $\sigma_k \geq \sigma_0$ and the smoothness hypothesis,

$$\|\mathbf{A}_s - \mathbf{A}_k\| \leq \frac{\|\mathbf{D}_s - \mathbf{D}_k\|}{\sigma_s} + \bar{\gamma} \frac{\sigma_s - \sigma_k}{\sigma_s \sigma_k} \leq \frac{C_A (s-k)}{\sigma_s^2}, \quad C_A = c_D \sqrt{\Delta_{\max}} + \frac{\bar{\gamma} \Delta_{\max}}{\sigma_0}.$$

The kernel has $\sum_r w_r (r+1) = 1 + \mu_w$, the Frobenius norms carry a factor $\sqrt{d}$, and Lemma 4 applies with $\mathbf{B}_{s,t}$ in place of $\mathbf{K}_s$ and the history taken in visiting order, so

$$\begin{aligned} \mathbb{E} \|\mathbf{S}_t^{\text{bwd}}\|^2 &\leq \frac{q^2}{d} \sum_{s \geq t} \|\mathbf{B}_{s,t}\|_F^2 \leq 2q^2 \bar{\gamma}^2 \sum_{s \geq t} \frac{T_{s-t}^2}{\sigma_s^2} + 2q^2 C_A^2 (1 + \mu_w)^2 \sum_{s \geq t} \frac{1}{\sigma_s^4} \\ &\leq \frac{2q^2}{\sigma_1^2} \left[\bar{\gamma}^2 c_\Delta + C_A^2 (1 + \mu_w)^2 \frac{c'_\Delta}{\sigma_1^2} \right], \end{aligned}$$

where $c_\Delta := \sup_t \sum_{s \geq t} T_{s-t}^2 \sigma_1^2 / \sigma_s^2$ and $c'_\Delta := \sum_s \sigma_1^4 / \sigma_s^4$. Both are finite with explicit bounds: $\sigma_s \geq \sigma_0$ gives $c_\Delta \leq (\sigma_1^2 / \sigma_0^2) \tau_2$, and $\sigma_s^4 \geq (\sigma_0^2 + s\Delta_{\min})^2$ gives $c'_\Delta \leq \sigma_1^4 / \sigma_0^4 + \sigma_1^4 / (\Delta_{\min} \sigma_0^2)$ by comparison with the integral. Neither depends on N . $\square$

The backward pass therefore has the analogous source guarantee, in mean square and under a smoothness condition on the normalization Jacobians. The guarantee follows from the same cancellation mechanism, with constants degraded by the operator mismatch between a site and the sites that correct it.

B Training Configuration

The training runs of Sec. 7.3 use the NeMo Qwen3-0.6B configuration, trained from scratch on Dolma C4: 28 layers, hidden size 1,024, FFN size 3,072, 16 attention heads with 8 query groups, RMSNorm, QK layer normalization, RoPE base 10^6 , and no dropout. The 151,669-token vocabulary of the Qwen tokenizer is padded to 151,936. Sequences have 2,048 tokens. The main run uses a micro batch of 24 and a global batch of 384 sequences, 786,432 tokens per step, for 12,716 steps, about 10.0B tokens in total. Parallelism is $P = 4$ without pipeline, context, data, or sequence parallelism. Training uses BF16 mixed precision with FP32 master weights, gradients, and optimizer states. The optimizer is AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-5}$, weight decay 0.1, and gradient clipping at 1.0. The learning rate warms up over 100 steps to its peak, 3.0×10^{-4} for the 16-bit run and 3.5×10^{-4} for the 4-bit run, and follows a cosine decay to 3×10^{-5} . The learning-rate ablation of Table 7b uses the same configuration with a global batch of 32 sequences for 4,096 steps, about 0.27B tokens. The reported loss is the training cross-entropy averaged over tokens.

C Implementation Details

The pipeline of Sec. 4.2 keeps four stages busy at once on each of the P GPUs, and a tile has to move from one stage to the next without the host taking part. Three things have to be in place beforehand. Each GPU has to know where to write inside its peers. The stages within a GPU have to overlap across tiles while the stages of any one tile stay in order. Finally, a GPU has to detect when the data it needs from a peer has arrived. The rest of this appendix describes these in turn, then the cache that keeps their setup out of the per-call cost, and finally the exactness of the backward rescaling of Sec. 5.4.

Workspace layout. Before a GPU can write into a peer, it must hold that peer’s addresses. A call uses three kinds of memory on every GPU: a buffer that receives the b -bit codes its peers send, a buffer that receives the scale and zero-point that decode those codes (Sec. 4.3), and an array of signal words that peers write to announce that the data has arrived. Each GPU allocates its own three and shares their addresses with the others when the tensor-parallel group is created, which gives one table of $3P+1$ addresses: the P code buffers, the P parameter buffers, the P signal arrays, and one flag base. The table is built once and stays fixed, and every later call reads the same entries, so a kernel reaches any peer by indexing the table with its rank. The compensator state $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ of Sec. 5.3 is kept out

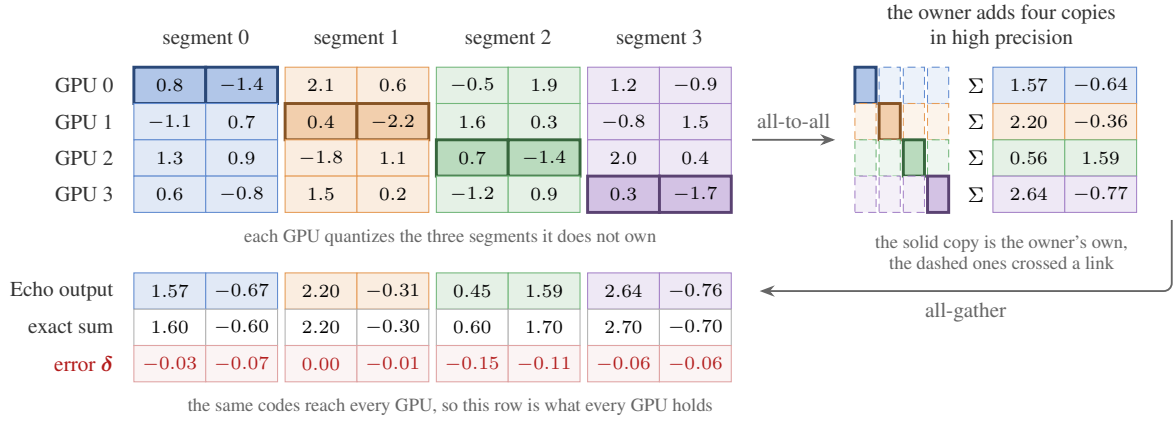


Figure 4. Worked example at $P = 4$. The quantizer is Eqn. (8) with $b = 4$ and a group equal to the two values of one segment. Top left: each GPU quantizes the three segments it does not own and keeps its own segment in full precision. Top right: the owner adds the four copies in high precision, three of them dequantized from b -bit codes and one its own. Bottom: the all-gather sends one quantized copy of each segment sum, so every GPU holds the same values. The last row is the error this all-reduce injects, the δ_k of Fig. 1. Inputs are shown to one decimal and sums to two.

of this table and owned by the compensation path, so the interface of the collective is the same with and without compensation.

Stream and event schedule. Push, reduce, gather, and tail each get their own CUDA stream. Work issued to different streams runs concurrently, so putting the stages on separate streams lets push quantize tile 3 while gather is still sending tile 1. Streams do not enforce the order within a tile, and tile 2 must not be reduced before it has been pushed. CUDA events enforce this order. When push finishes tile 2 it records an event, and reduce waits on that event before it starts tile 2. Since a stage waits only for the stage directly before it, the event of a tile is no longer needed once the next stage has started on the tile, and two event slots used in turn suffice for the whole pipeline (Fig. 2(b)). The caller is synchronized at the two ends. It records an event before the first push, so the pipeline starts only once the input is ready, and waits on one after the last tail, so its next kernel starts only once the output is complete. The compensator update is issued on a fifth stream after the last tail and runs alongside the caller’s next matrix multiplication, and the entry stage of the next call waits on an event recorded at the end of the update. The host launches the kernels and returns. It performs no synchronization while the tiles are in flight.

Flag values. Events do not cross GPUs. A stage that needs a segment from a peer therefore polls the flag of Sec. 4.2 and waits until it reads the value that stands for the data it needs. Each stage publishes a value one higher than the stage before it. Push publishes the base value, reduce publishes the base plus one, and gather publishes the base plus two, which is the value tail polls for. The host raises the base by three at every call, so no value is used twice in the life of the group, and no flag needs to be cleared. A consumer that arrives early reads a smaller number and keeps polling, and a flag still holding a value from the previous call is below anything the current call polls for, so it cannot release a consumer by mistake. No slot is cleared between calls and no CPU round trip takes place on the readiness path.

Schedule cache. The tiling, the launch order, and the streams and events above depend only on the shape of the call, so they are computed once and reused, with a cache keyed by the data type, bit width, policy, group size, and segment size. The cache stores the tiling and launch table together with the streams and events for each device. Only the first call for a new shape, or the first after a rebuild, pays this build and warm-up cost, which the latency measurements of Sec. 7 exclude, so all reported numbers are steady state.

Backward rescaling. Section 5.4 multiplies the gradient by a shared power-of-two factor before the quantized all-reduce and divides the result by the same factor after it. Three choices make the scaling exact. The factor is a power of two, so multiplying and dividing shift only the floating-point exponent and add no rounding of their own. Its exponent is derived from the largest gradient magnitude across the tensor-parallel group, obtained with a max all-reduce of a single scalar, so every GPU applies the same factor and the all-reduce computes the same sum, only rescaled. Finally, the two steps of the compensator run outside the scaling. The compensator forms $\mathbf{x}' = \mathbf{x} + \mathbf{c}$ before the multiplication and $\mathbf{u} = \mathbf{x}' - \hat{\mathbf{y}}/P$ after the division, so $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$ stay in original gradient units and the analysis of Secs. 5.1 and 6 applies unchanged.