WHERE THE FP4 PRETRAINING GAP SITS, AND WHY THAT DECIDES WHAT TRANSFERS ACROSS SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

Training a language model in 4-bit floating point (FP4) is faster than in BF16 but ends at a higher loss. Recipes narrow this gap by keeping some computations, or the end of training, in higher precision, and these choices are usually tuned on small models. We ask where the gap comes from in one FP4 recipe at three model sizes, and find that the answer changes with scale. In a small model nearly all of the gap comes from rounding in the forward pass; as the model grows this share falls, and rounding the gradients, alone and together with the forward pass, takes over. At every size a similar share of the gap is the cost of the rounding still in progress, which a switch to BF16 removes at once, while the rest stays in the weights. Since higher precision in one part of training removes at most the share of the gap that arises there, where the gap sits decides what transfers across scale: a higher-precision forward pass removes most of the gap in a small model but less than half in a large one, while a short switch to BF16 at the end removes the same share at every size. A few runs at the target scale measure these shares before any recipe is trained.

1 INTRODUCTION

Training a language model in a 4-bit floating-point (FP4) format such as NVFP4 or MXFP4 runs its matrix multiplications at twice the throughput of 8-bit formats, but it ends at a higher training loss than training in BF16. Recipes narrow this gap by keeping some of the computation in higher precision: the inputs of the forward matrix multiplication (GEMM), the last blocks of the model, or the last part of training (NVIDIA et al., 2025; Tseng et al., 2025; Chmiel et al., 2025; Castro et al., 2025). We call such a choice a precision allocation. Each allocation costs throughput, and each is tuned by ablations at one scale. Whether it pays off on a larger model is the question a practitioner faces.

One might expect the published accounts of where the gap comes from to answer this, but they disagree. At 12B, switching the forward matrix multiplications to higher precision late in training recovers most of the gap and switching the backward ones recovers nothing (NVIDIA et al., 2025); at 7B, switching the backward ones closes the gap (Chmiel et al., 2025). Each account comes from one recipe at one scale, and the recipes, formats and references differ at the same time.

In this work we measure where the gap of one NVFP4 recipe sits at 60M, 130M and 1B parameters. We split it by tensor, with runs that round only the activations and weights or only the gradients, and by time, with runs that switch to BF16 partway through training, and we compare what the allocations in use remove with what the split measures. Our key findings are as follows.

An allocation removes at most the fraction of the gap it acts on. Across the allocations we ran, running the forward GEMM in MXFP8 removes at most the fraction of the gap that rounding activations and weights produces on its own, and switching to BF16 for the last steps of training, a BF16 tail, removes at most the gap still recoverable when it starts (Figure 1). A tail removes all of that within about a thousand steps, and a longer tail removes no more. (Section 3)

The split of the gap changes with scale. Rounding activations and weights produces 89% of the gap at 60M and 42% at 1B. At 1B the gap from rounding gradients, constant through training at the two smaller scales, grows through training, and a quarter of the gap appears only when activations,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

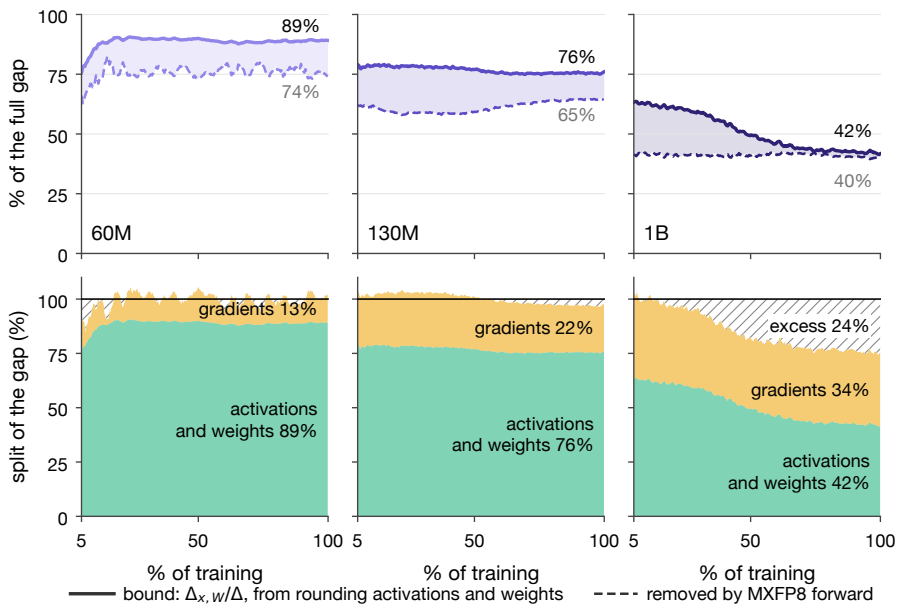

Figure 1: An allocation removes at most the fraction of the gap it acts on, and that fraction moves with scale. Top, solid: the fraction of the full gap that rounding activations and weights produces on its own, $\Delta_{x,W}(t)/\Delta(t)$, over training at each scale. Dashed: the fraction of the gap that running the forward GEMM in MXFP8 removes. The dashed curve stays under the solid one through training at all three scales, except by 0.2 points (0.0002 nats) at 1B; the solid curve falls from 89% at 60M to 42% at 1B, and the removed fraction falls with it, from 74% to 40%. Bottom: the full gap split into the gap from rounding activations and weights, the gap from rounding gradients and the excess (hatched). Two seeds per curve at 60M and 130M, one at 1B.

weights and gradients are rounded together. The fraction of the gap that a BF16 tail removes stays near a quarter at every scale. (Section 4)

The bound transfers across scale; the numbers do not. Running the forward GEMM in MXFP8 removes 74% of the gap at 60M and 40% at 1B, following its bound, while tails of equal length remove the same fraction at the scales we compared. Two runs at the target scale, one fully in NVFP4 and one that rounds only the activations and weights, give the bound there before any allocation is trained. (Section 5)

The gaps have simple causes. The gap from rounding activations and weights is rounding error on the activations entering the forward GEMM; its cost is set by the size of that error, not by clipping or outliers, and it grows because the loss becomes more sensitive to the same error as the model converges. The gap from rounding gradients is the cost of unbiased noise in the updates, which does not grow at small scale. A switch to BF16 removes the cost of the rounding in progress and not what earlier rounding has left in the weights. (Section 6)

2 Setup

Models and training. We train Llama-style decoders (Touvron et al., 2023) at three scales, 60M, 130M and 1B parameters, on FineWeb-Edu (Penedo et al., 2024) with the GPT-2 tokenizer (Radford et al., 2019), for 1.2B, 2.6B and 20B tokens, with AdamW (Loshchilov & Hutter, 2019) and a learning rate that warms up over the first 1% of steps and then falls along a cosine curve to 5% of its peak (Appendix A lists the model shape and hyperparameters of each scale). Each model is trained with the linear layers of its transformer blocks in simulated NVFP4 and in BF16 as the reference. The difference between their training losses, which we call the gap, is our principal measure.

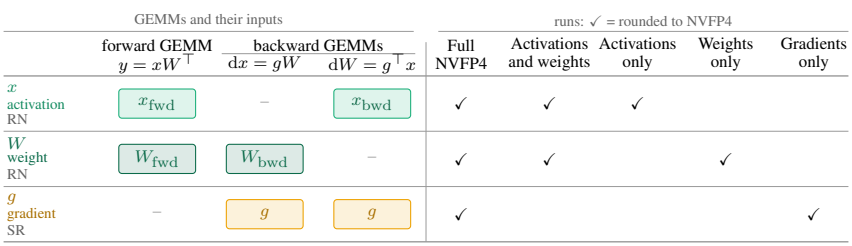

Figure 2: The three tensors of a linear layer, activation x, weight W and gradient g, and the three GEMMs of training. Each tensor enters two GEMMs and is quantized separately for each, so it has two FP4 copies. The right part lists the runs; each rounds both copies of the ticked tensors to NVFP4.

FP4 recipe. NVFP4 (NVIDIA et al., 2025) stores each value in 4 bits. We do not run native NVFP4 GEMMs but simulate them (Appendix A gives the reason): before a matrix multiplication, each input that is rounded is quantized to NVFP4 and dequantized back to BF16, and the multiplication runs in BF16. Every linear layer inside the transformer blocks is simulated, while embeddings, the output layer, normalization and the two attention products stay in BF16.

Tensors and GEMMs. The linear layers we round are the attention projections and the MLP of each transformer block. Each takes an activation x, the output of the layer before it, and multiplies it by its weight W to produce its output $y = xW^\top$. Training runs three matrix multiplications (GEMMs) per layer (Figure 2): the forward GEMM computes the output $y = xW^\top$, and two backward GEMMs compute the gradients, $\mathrm{d}x = gW$ with respect to the input and $\mathrm{d}W = g^\top x$ with respect to the weight, where $g = \partial L/\partial y$ is the gradient of the loss L with respect to the output. Each of the three tensors enters two GEMMs, and each GEMM quantizes its inputs afresh from their BF16 values (Appendix A explains why). Each tensor therefore has two FP4 copies, which we call its GEMM inputs and mark by where they are used: x_{fwd} is the copy of x in the forward GEMM and x_{bwd} its copy in the backward GEMM for $\mathrm{d}W$, and likewise W_{fwd} and W_{bwd}; both copies of g are in backward GEMMs, and the main text does not tell them apart. Activations and weights are rounded to nearest (RN), and gradients use stochastic rounding (SR) (Gupta et al., 2015), whose expected value is the unrounded value, so that the rounding errors do not add up to a bias as the updates accumulate over steps; Appendix B explains why the two use different rounding.

Two kinds of runs. We train two kinds of FP4 runs. The first kind is for analysis. Such a run rounds a chosen group of tensors, both copies of each, in every simulated layer and leaves the rest in BF16, which shows how much of the gap that group produces on its own. Besides full NVFP4, which rounds all three tensors, these are the runs that round only the activations and weights, only the gradients, only the activations and only the weights (Figure 2), each named after the tensors it rounds; a run that switches from NVFP4 to BF16 partway through training is of this kind as well. The second kind is a precision allocation, a scheme one would actually train with. An allocation decides where NVFP4 is used and where higher precision is: which GEMM inputs, in which layers, and during which steps. Since higher precision costs throughput, an allocation spends a limited budget. Running the forward GEMM in a higher-precision format and switching the whole model to BF16 for the last steps are the two we study most; Appendix G lists every allocation we ran and its cost. Section 3 relates the two kinds of runs: for the allocations we ran, what an allocation removes is bounded by what the analysis runs measure.

The gap and how we split it. The gap is the training loss of an FP4 run minus the training loss of the BF16 reference. We use Δ for the gap of full NVFP4, the run that rounds all three tensors, and $\Delta(t)$ for its value after a fraction t of the training steps, so that $t = 1$ is the end of training; the gaps of the other runs carry a subscript that names what they round. We use the fraction rather than the step so that the three scales, which train for different numbers of steps, can be compared at the same point. Both losses are smoothed with a 1,000-step running mean, because the loss of a single step is

too noisy to compare (Appendix C gives the smoothing and the seed variation). We split the gap in two ways, by tensor and by time.

By tensor: rounding only the activations and weights, with the gradients kept in BF16, gives a gap $\Delta_{x,W}$, and rounding only the gradients gives Δ_g. We group the activations with the weights because they are the two inputs of the forward GEMM and are rounded the same way, while the gradient enters only the backward GEMMs and is rounded stochastically; this is also the line along which the allocations of Section 3 divide precision. If the two sources of rounding error affected the loss independently, rounding all three tensors would give $\Delta_{x,W} + \Delta_g$. Whether they do is an empirical question, and Section 4 finds that the answer changes with scale. To measure the departure, we define Δ_{ex} as the difference,

$$\Delta_{\mathrm{ex}} = \Delta - \Delta_{x,W} - \Delta_g. \tag{1}$$

It is zero when the two errors add, positive when rounding all three tensors together costs more than the sum, and negative when it costs less; $\Delta = \Delta_{x,W} + \Delta_g + \Delta_{\mathrm{ex}}$ holds by definition. We call Δ_{ex} the excess because wherever it is not zero in our runs it is positive.

By time: we train a run in NVFP4 up to a fraction t of training, save it, and continue it from that point in two branches, one that stays in NVFP4 and one that switches to BF16, so that both share the same NVFP4 training up to t. The branch that stays in NVFP4 is the full-NVFP4 run, with gap Δ at the end. The branch switched to BF16 still ends with a gap against the BF16 reference, and we call this remaining gap the persistent gap $\Delta_{\mathrm{pers}}(t)$: it is what the NVFP4 training before t left behind and the BF16 training after t did not remove. The difference between the two branches at the end, $\Delta_{\mathrm{rec}}(t) = \Delta - \Delta_{\mathrm{pers}}(t)$, is what the switch at t did remove, and we call it the recoverable gap. Section 6 identifies the two: the recoverable gap is the cost of the rounding still in progress at t, and the persistent gap is what the rounding before t has left in the weights and optimizer state.

We report each of these gaps in nats and also as a fraction of the full gap, $\Delta_{x,W}/\Delta$ for example, because the full gap itself differs between scales and the fractions can be compared across them. For an allocation we report the reduction of the gap, the full gap minus the gap of the allocation's run, likewise as a fraction of the full gap.

3 An allocation removes at most the fraction of the gap it acts on

An allocation that raises the precision of some GEMM inputs, or of some steps, acts on the gap that those inputs or steps produce. We study two existing allocations, one that divides precision by tensor and one that divides it by time. The first keeps the forward GEMM in higher precision (Tseng et al., 2025): we run the forward GEMM in MXFP8 (Rouhani et al., 2023), a block-scaled 8-bit format, and keep the backward GEMMs in NVFP4, so that this allocation, MXFP8 forward, acts on the gap from rounding activations and weights. The second runs the tail of training, the last $p\%$ of steps, with the whole model switched to BF16 (Chmiel et al., 2025; NVIDIA et al., 2025); this BF16 tail acts on the gap that is still recoverable when it starts.

We find that, across the allocations we ran, an allocation removes at most the fraction of the gap that it acts on: MXFP8 forward removes at most $\Delta_{x,W}/\Delta$, the fraction of the full gap that rounding activations and weights produces on its own, and a BF16 tail switched on at a fraction t of training removes at most $\Delta_{\mathrm{rec}}(t)/\Delta$, the fraction that a switch to BF16 at t recovers. By tensor, the gap an allocation acts on is measured by the analysis run that rounds only those tensors, so the bound can be tested directly, and the question of how much the allocation removes reduces to that one run. By time, what a tail removes is set when it starts: the BF16 steps after the first thousand remove nothing more.

By tensor. MXFP8 forward acts on the gap from rounding activations and weights, $\Delta_{x,W}$, since its two MXFP8 inputs, x_{fwd} and W_{fwd}, are copies of these two tensors, while their other copies, x_{bwd} and W_{bwd}, stay in NVFP4; it can therefore remove at most the fraction $\Delta_{x,W}/\Delta$ of the gap. Figure 1 shows the fraction it removes and $\Delta_{x,W}/\Delta$ over training at the three scales (Appendix G tabulates the end-of-training gap and cost of every allocation). The removed fraction stays under $\Delta_{x,W}/\Delta$ throughout training at every scale. If the forward GEMM runs in BF16 instead of MXFP8, which we tried at 60M, it removes slightly more but still stays under $\Delta_{x,W}/\Delta$. The one allocation we ran

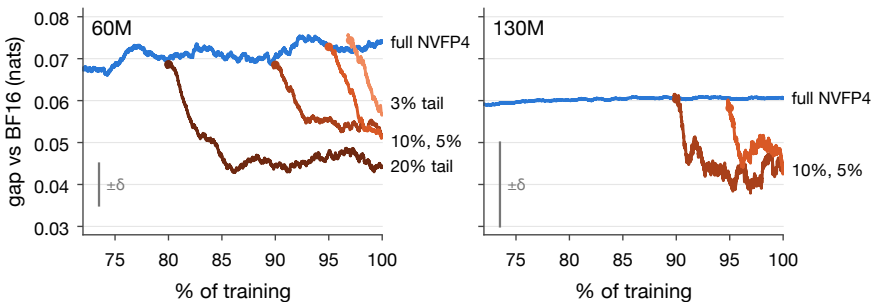

Figure 3: Switching the whole model from NVFP4 to BF16 for the last $p\%$ of steps, a BF16 tail, removes most of what it removes within a thousand steps of the switch, and a longer tail removes no more: the 5% and 10% tails end at the same gap at both scales, and the 20% tail, which switches earlier, ends lower. Each orange curve is a tail; the dot marks the switch. The blue curve stays in NVFP4 throughout. The bar marked $\pm\delta$ shows the typical difference between the final losses of two BF16 runs that differ only in the seed, measured in Appendix C.

on the gradients, the backward GEMMs of the last block in BF16 at 60M, removes 3% of the gap against a bound of 13%.

By time. A BF16 tail removes most of what it will ever remove within the first thousand steps after the switch, and the gap then stays constant for the rest of training (Appendix G gives the time course of each tail); the activations-only run switched to BF16 at the midpoint of training keeps a constant gap for the remaining half. The gap that remains after a switch, the persistent gap $\Delta_{\text{pers}}(t)$, therefore does not shrink with further BF16 training. A tail switched at a fraction t of training removes the recoverable gap $\Delta_{\text{rec}}(t)$, all of it within about a thousand steps, and no tail removes more than the recoverable gap at its switch, however long it runs; this is the bound of the opening paragraph for allocations by time. A longer tail costs more, since every BF16 step gives up the throughput of NVFP4, and it helps only by switching earlier, when less of the gap has become persistent: the 20% tail, switched at 80% of training, ends at a smaller gap than the 10% tail, but the 10% tail ends at the same gap as the 5% tail at both scales (Figure 3), because little of the gap becomes persistent between 90% and 95%. Late in training, where the persistent gap has stopped growing, a tail needs only about a thousand steps, and a short tail does what a longer one does at a fraction of the cost.

4 The split of the gap changes with scale

At 60M, rounding activations and weights alone produces 89% of the full gap; at 1B it produces 42% (Figure 1). To see why, we follow the gaps of the analysis runs over training at each scale (Figure 4). The fraction falls because the gap from rounding activations and weights, $\Delta_{x,W}$, shrinks with scale while the rest of the gap grows. The gap from rounding gradients, Δ_g, stays constant through training at the two smaller scales but grows at 1B. At 1B the two gaps also no longer add up to the full gap: rounding all three tensors together costs more than their sum, and the excess Δ_{ex} reaches a quarter of the gap. The fraction of the gap that a BF16 tail removes, in contrast, is about the same at every scale.

The gap from rounding activations and weights shrinks. The gap from rounding activations and weights, $\Delta_{x,W}$, is smaller at each larger scale and grows less over training: the 1B curve ends lowest and is the flattest of the three (Figure 4, second panel). Of its two tensors, the activations carry the growth. At every scale the gap from rounding activations alone, Δ_x, grows through training and is smaller at each larger scale, while the gap from rounding weights alone, Δ_W, stays constant through training at 0.01 to 0.02 nats (Appendix D gives both runs at each scale); Section 6 traces Δ_x to the rounding error on the activations that enter the forward GEMM.

The gap from rounding gradients grows at 1B. At 60M and 130M, the gap from rounding gradients, Δ_g, stays constant from early in training. At 1B it grows through training and ends

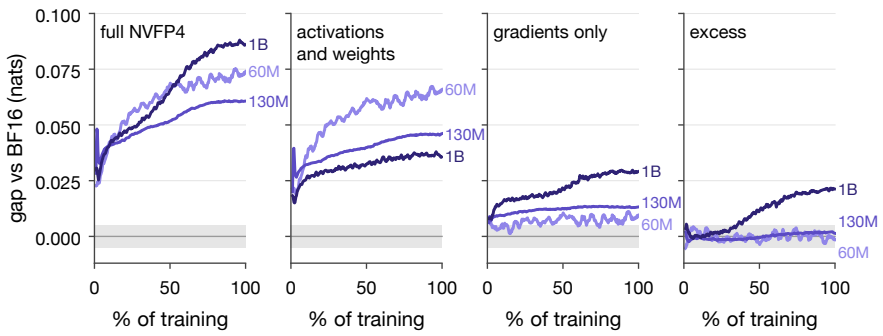

Figure 4: The gap from rounding activations and weights shrinks with scale, while the gap from rounding gradients, constant through training at 60M and 130M, and the excess, zero there, both grow through training at 1B. Each panel follows one gap over training, one curve per scale; the grey band is $\pm\delta$, the seed variation of the final loss (Appendix C).

about twice as large (Figure 4, third panel; Appendix F gives the two 1B seeds and the per-tensor measurements of rounding error). Chmiel et al. (2025) report the same at 7B: once the gradients fall below the quantization noise late in training, switching the backward GEMMs to higher precision closes the gap. NVIDIA et al. (2025) find no such gain at 12B, but their reference is FP8, which rounds the gradients itself (Appendix G reads their switches in our terms).

At 1B the gap exceeds the sum of the two. At 60M and 130M, $\Delta_{x,W}$ and Δ_g add up to the full gap throughout training, so the excess Δ_{ex} is zero (Figure 4, fourth panel). At 1B it is not zero: from the middle of training the full gap grows faster than the sum, and the excess reaches a quarter of the gap by the end. The variation in the final loss between two BF16 runs that differ only in the seed is about 0.005 nats at 1B (Appendix C), and the excess is four times that. Inside $\Delta_{x,W}$, the gaps from rounding activations alone and weights alone still add up at 1B (Appendix D), so the excess arises between the activations and weights on one side and the gradients on the other.

At 1B most of the gap is in the gradients and the excess. At 1B the gap from rounding gradients and the excess together make up more than half of the gap. MXFP8 forward leaves x_{bwd} and W_{bwd} in NVFP4, so it still rounds activations and weights: its bound stays $\Delta_{x,W}/\Delta = 42\%$, it removes 40%, and a better format for the forward GEMM alone has little left to gain. Keeping both copies of the activations and weights in higher precision leaves only the gradients rounded, and the run that rounds only the gradients has the gap Δ_g. Such an allocation can therefore remove up to $1 - \Delta_g/\Delta = 66\%$ of the gap, $\Delta_{x,W}$ and the excess together. At 60M and 130M, where the gradients carry a fifth of the gap or less and there is no excess, keeping the backward GEMMs of the last block in BF16 removes 3% of the gap (Appendix G); a study of backward precision at small scale would find nothing there, while at 1B the next gain lies in the backward copies of the activations and weights.

The fraction a BF16 tail removes stays near a quarter. The 3–10% tails of Section 3 remove 23–30% of the gap at every scale (Appendix G lists each tail). A switch of the activations-only run to BF16 at the midpoint of training shows the same: it removes more than half of that run's gap at both 60M and 1B (Appendix G gives both switches). Each of these removes what the bound of Section 3 allows it, the recoverable gap at its switch, so the bound by time, unlike the bound by tensor, does not move with scale.

5 The bound transfers across scale; the numbers do not

The bound of Section 3 holds for our allocations at all three scales, while the fraction it allows changes with scale (Section 4). What carries over from a small model to a large one is therefore the bound. It can be measured at the target scale before any allocation is tried: two runs give the bound by tensor, and one switch to BF16 gives the bound by time. Figure 5 shows the two cases.

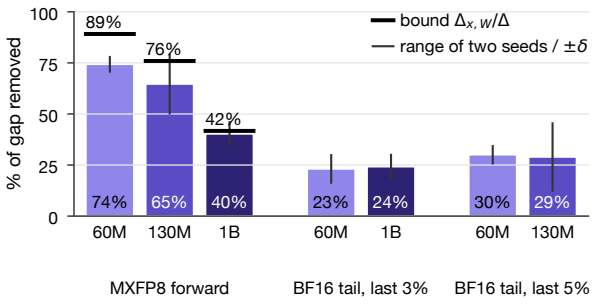

Figure 5: What an allocation removes follows its bound at each scale. Bars: the fraction of the full gap removed at the end of training. MXFP8 forward is bounded by the fraction from rounding activations and weights (black tick), which falls with scale; a BF16 tail by the recoverable gap, whose fraction does not. Error bars: range of two seeds, or $\pm\delta$ for one run.

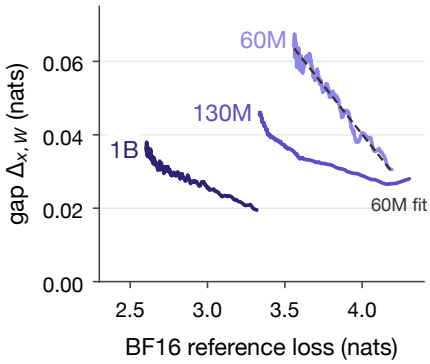

Figure 6: Within a run the gap from rounding activations and weights falls on a straight line against the BF16 reference loss, and each scale lies on its own line: a larger model reaches a lower loss with a smaller gap. Each curve follows one scale over training after the first 3,000 steps, from right (early) to left (late); the dashed line is the 60M fit.

Allocations on the forward GEMM, whose bound falls. Allocations that raise the precision of the forward GEMM act on the gap from rounding activations and weights, whose fraction of the full gap falls with scale (Section 4). MXFP8 forward removes 74% of the gap at 60M and 40% at 1B, following its bound from 89% to 42% (Figure 5); an allocation tuned at 60M would promise three quarters of the gap at 1B and return two fifths, and the 42% bound from two runs at 1B says so in advance. Keeping one layer in BF16, the one with the largest rounding error at 60M, removes 18% of the gap there and 13% at 130M (Appendix G).

BF16 tails, whose bound stays. A BF16 tail is a switch to BF16 at a fraction t of training, so what it can remove is the recoverable gap at t, the cost of the rounding still in progress when it switches (Section 6). That cost is a quarter of the full gap at every scale, and tails of equal length remove the same fraction at 60M and 1B and at 60M and 130M (Figure 5), so the length of a tail can be tuned on a small model. Why the fraction is a quarter we cannot show. It would follow if what a switch recovers and what it leaves behind both grow with the model's sensitivity to the rounding error (Section 6): their ratio would then be set by the training schedule, which our three scales share, and not by the model. A tail on the forward GEMM alone recovers as much as a full tail at 60M and 130M at a third of the cost (Appendix G lists both); it acts on the activations and weights, whose fraction moves, so its benefit has to be measured at the target scale.

6 Why the gaps behave this way

Sections 3 to 5 measured the gaps. This section asks what each one is, using runs that change one thing at a time (Table 5 lists them): where the gap from rounding activations arises and why it grows through training, why the gap from rounding gradients stays constant at small scale, and why a switch to BF16 removes part of the gap at once and never the rest.

The growing cost comes from the activations, through the forward GEMM. At all three scales the gap from rounding activations alone grows through training, while the gap from rounding weights alone stays constant from early on (Section 4; Appendix D gives both runs at each scale), so the growth of the gap from rounding activations and weights comes from the activations at every scale. The activation is rounded twice in every step, once as an input of the forward GEMM and once as an input of the backward GEMM for $\mathrm{d}W$, in the same way both times. At 60M, where we separated the two, rounding x_{fwd} alone gives 0.046 nats, the whole gap from rounding activations,

while rounding x_{bwd} alone gives 0.006 and no growth. The cost that the next four paragraphs trace is therefore that of a rounded input to the forward GEMM.

The cost is set by the size of the rounding error. A rounding error has a size, measured by its mean square, and a bias, the amount by which it leans one way on average. The gap follows the size. Stochastic rounding has no bias but twice the mean squared error of rounding to nearest (Appendix B describes the two), and used on the activations and weights it makes the gap from rounding them 3.3 times larger. MXFP4, whose block scales are restricted to powers of two and so are coarser than those of NVFP4, makes it 1.6 times larger. MXFP8 has a mean squared error 6 to 10 times smaller than NVFP4 on the same tensors, and MXFP8 forward removes 83%, 85% and 96% of what its bound allows at 60M, 130M and 1B (Appendix E gives the error of each format and the gap of each run).

Clipping and outliers are not the cause. Removing the clipping of NVFP4, which affects about 3% of the elements of every GEMM input, leaves the gap unchanged, and a Hadamard rotation that spreads the outliers of a block, its few largest elements, over the others makes the gap larger (Appendix E defines both and gives the runs). Among the changes we tried, only the mean squared error moves this gap.

The cost grows because the sensitivity to the error grows. The rounding error does not grow. Relative to the values it rounds, the error on the weights stays near 0.10 through training, the relative mean squared error on the forward activations in the last part of training is 0.94 times its value in the first, and activation outliers do not become more frequent (Appendix E). The gap grows nonetheless, so what grows is the sensitivity of the loss to a rounding error of fixed relative size.

The growth follows convergence, not the learning rate. The learning rate falls through training under our cosine schedule, the rule that sets the learning rate at each step, so the gap could be growing with the falling learning rate. It is not: under a constant learning rate the gap still grows, from 0.032 nats at 10% of training to 0.059 at the end; halving the final learning rate of the cosine schedule leaves the gap unchanged; and doubling the tokens levels it off, the last fifth of the longer run adding 0.001 (Appendix E gives the schedule runs). Within one run the gap falls on a straight line against the BF16 reference loss (Figure 6; Appendix E gives the fit), so it tracks how far the model has converged rather than the step or the learning rate.

A larger model is less sensitive to the same rounding error. Within a run, a lower loss brings a larger gap, yet a larger model reaches a lower loss with a smaller gap: each scale lies on its own line (Figure 6). At the 1B reference loss of 2.61, the 60M line would give 0.106 nats, while the 1B gap is 0.036. The format, and so the relative rounding error, is the same; the 1B model is about a third as sensitive to it.

Rounding the gradients adds unbiased noise, whose cost does not grow. Stochastic rounding makes the rounded gradient an unbiased estimate of the exact one, so its effect on training is that of noise in the updates. We added unbiased noise to the gradients of a BF16 run, with the same mean square as the error of stochastic rounding. It gives a constant gap of 0.009 nats, the same as rounding the gradients at 60M, and noise four times as large gives a constant 0.105. Rounding the gradients to nearest instead adds a bias, and the gap then grows through training to 0.057 (Appendix E gives these runs). This is why the gap from rounding gradients is a constant offset at 60M and 130M. At 1B it grows through training although the measured rounding error of the gradients rises no more than at 130M (Appendix F): the same noise costs more at 1B, and more as training proceeds.

A switch removes the cost of the rounding in progress, not what earlier rounding left in the weights. When the activations-only run at 60M switches from BF16 into NVFP4 at the midpoint of training, a gap of 0.027 nats appears within 100 steps; when a run switches the other way, the recoverable gap leaves as fast and the rest never leaves (Section 3). The rest sits in the weights and optimizer state. The run switched into NVFP4 at the midpoint ends 0.012 nats below the run that was in NVFP4 throughout, and most of this difference is what the first half in NVFP4 left in the weights and optimizer state rather than the different batches the two runs see after the switch

(Appendix C gives the test). This is why a tail does its work within its first thousand steps and never reaches the persistent gap.

7 Related work

FP4 pretraining recipes combine block scaling, stochastic rounding of the gradients, Hadamard rotations and layers or phases kept in higher precision (NVIDIA et al., 2025; Tseng et al., 2025; Castro et al., 2025; Chmiel et al., 2025; Panferov et al., 2026; Chen et al., 2026b; Rahimifar et al., 2026; Cim et al., 2026; Hu et al., 2026), and earlier FP4 training scaled whole vectors rather than blocks (Wang et al., 2025); the space of such choices has been ablated at small scale (Hu et al., 2025). Each recipe reports its gap to a higher-precision reference at the scales it trains. We take one of these recipes and measure where its gap sits.

Published accounts of where the gap comes from disagree. At 12B, most of the gap is placed on the forward pass: switching the forward GEMMs to higher precision late in training recovers most of it and switching the backward GEMMs recovers nothing, against an FP8 reference that itself rounds the gradients (NVIDIA et al., 2025). At 7B, a switch of the backward GEMMs closes the gap once the gradients fall below about $\sqrt{3}$ times the quantization noise (Chmiel et al., 2025). Other studies place the damage on the GEMM that computes the weight gradient (Cim et al., 2026), on the block scales of a tensor differing from those of its transpose (NVIDIA et al., 2025; Rahimifar et al., 2026), or on an activation mean bias that strengthens during training (Cao et al., 2026). These accounts come from different recipes, formats and scales. We find that the split moves with scale within one recipe, so accounts from different scales can disagree even when nothing else differs, and the bound of Section 3 accommodates both switch results: a switch removes at most the fraction of the gap that its side produces, and that fraction depends on the recipe and the scale. Note, however, that Cao et al. (2026) attribute the forward cost to a bias that grows, whereas in our runs the forward gap follows the mean squared rounding error and not its bias, and the relative error does not grow through training (Section 6).

Studies that vary which tensors are quantized, precision scaling laws, the allocations in use and the prior work behind our mechanisms are discussed in Appendix H.

8 Discussion

We have measured where the gap of one NVFP4 recipe sits at 60M, 130M and 1B and found that an allocation removes at most the fraction of the gap it acts on, a bound that carries over from a small model to a large one while the fractions it allows do not.

The bound is worth measuring because one measurement covers a class of allocations. Every allocation on the forward GEMM, whatever format or layers it chooses, removes at most $\Delta_{x,W}/\Delta$, so the run that rounds only the activations and weights, together with the full-NVFP4 run that any comparison needs anyway, bounds all of them at once. At 1B it says that no forward allocation removes more than 42% of the gap and that MXFP8 forward, at 40%, has taken nearly all of it, so the next gain lies elsewhere. A switch to BF16 bounds every tail in the same way and costs only the steps after the switch. The fractions have to be read at the end of training, since they move through it (Figure 1).

The mechanisms of Section 6 explain what happens within a run and leave the cross-scale facts open: why a larger model is less sensitive to the same rounding error, which makes the forward fraction fall with scale; whether the growth of the gradient gap at 1B and the excess are one fact, the same noise costing more late in a large run; and why the recoverable gap is a quarter at every scale, which a run with another learning-rate schedule would test (Section 5).

The bound by tensor rests on the forward side: the one allocation we ran on the backward GEMMs is a single run at 60M, where it removes 3% against a bound of 13%. At 1B, where more than half of the gap lies in the gradients and the excess, allocations on the backward GEMMs are the runs that would test the bound where it matters most, and if the trends of Section 4 continue beyond 1B, they are where the gap will be. Our measurements are of one recipe, simulated; the same two runs would show whether other recipes behave alike.

REPRODUCIBILITY STATEMENT

The training setup, the FP4 simulation and its validation are in Appendix A; the smoothing, seed floors and every run used are in Appendix C. Every number in the paper is computed from the logged training losses and the per-tensor and probe logs of the runs listed there. We will release the simulation code, the run configurations and the loss logs.

AI USE STATEMENT

Large language models were used to revise and polish the writing of this paper. The authors take full responsibility for all content, including any text produced with their assistance.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Hengjie Cao, Zhendong Huang, Mengyi Chen, Yifeng Yang, Fang Dong, Anrui Chen, Ruijun Huang, Xin Zhang, Mingzhi Dong, Yujiang Wang, et al. The curse and blessing of mean bias in FP4-quantized LLM training. *arXiv preprint arXiv:2603.10444*, 2026.

Roberto L. Castro, Andrei Panferov, Soroush Tabesh, Oliver Sieberling, Jiale Chen, Mahdi Nikdan, Saleh Ashkboos, and Dan Alistarh. Quartet: Native FP4 training can be optimal for large language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Mengzhao Chen, Chaoyi Zhang, Jing Liu, Yutao Zeng, Zeyue Xue, Zhiheng Liu, Yunshui Li, Jin Ma, Jie Huang, Xun Zhou, and Ping Luo. Scaling law for quantization-aware training. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. arXiv:2505.14302.

Yuxiang Chen, Yifan Liu, Xiaoming Xu, Pengle Zhang, Michael Beyer, Martin Rapp, Jun Zhu, and Jianfei Chen. TetraJet-v2: Accurate NVFP4 training for large language models with oscillation suppression and outlier control. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026b.

Brian Chmiel, Maxim Fishman, Ron Banner, and Daniel Soudry. FP4 all the way: Fully quantized training of large language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Musa Cim, Sarthak Arora, Poovaiah Palangappa, Miro Hodak, Ravi Dwivedula, Meena Arunachalam, and Mahmut Taylan Kandemir. Pretraining large language models with MXFP4 on native FP4 hardware. *arXiv preprint arXiv:2605.09825*, 2026.

Enea Monzio Compagnoni, Tianlin Liu, Rustem Islamov, Frank Norbert Proske, Antonio Orvieto, and Aurelien Lucchi. Adaptive methods through the lens of SDEs: Theoretical insights on the role of noise. In *International Conference on Learning Representations*, 2025.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1737–1746, 2015.

Robert Hu, Carlo Luschi, and Paul Balanca. Elucidating the design space of FP4 training. *arXiv preprint arXiv:2509.17791*, 2025.

Robert Hu, Carlo Luschi, and Paul Balanca. UE5M3 FP4 block scaling for stable language model pretraining. *arXiv preprint arXiv:2609.02846*, 2026.

Tanishq Kumar, Zachary Ankner, Benjamin F. Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. In *International Conference on Learning Representations*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Marin Community. [precision] native FP4 (NVFP4/MXFP4) pretraining for the EOY GB200 hero run. GitHub issue #7403, marin-community/marin, `https://github.com/marin-community/marin/issues/7403`, 2026. Opened 2026-07-19.

NVIDIA, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, et al. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025.

Open Compute Project. OCP microscaling formats (MX) specification, version 1.0. `https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf`, September 2023.

Andrei Panferov, Erik Schultheis, Soroush Tabesh, and Dan Alistarh. Quartet II: Accurate LLM pre-training in NVFP4 by improved unbiased gradient estimation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2601.22813.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Mehdi Rahimifar, Amin Darabi, Mehran Taghian Jazi, Xing Huang, Yao Wang, Zhijun Tu, Yufei Cui, Yunke Peng, and Hongliang Li. Stable FP4 training via transposition-invariant block quantization. *arXiv preprint arXiv:2607.24953*, 2026.

Tanzila Rahman, Mehran Taghian Jazi, Yunke Peng, Zhuang Ma, Anandharaju Durai Raju, Yao Wang, Xing Huang, Hei Yi Mak, Shadan Golestan, Hoang Le, Yonghan Dong, Wei Guo, and Yaoyuan Wang. Activation outliers matter: Robust recovery for quantized multimodal LLMs. *arXiv preprint arXiv:2608.26581*, 2026.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Xingwu Sun, Shuaipeng Li, Ruobing Xie, Weidong Han, Kan Wu, Zhen Yang, Yixing Li, An Wang, Shuai Li, Jinbao Xue, Yu Cheng, Yangyu Tao, Zhanhui Kang, Chengzhong Xu, Di Wang, and Jie Jiang. Scaling laws for floating-point quantization training. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57544–57570, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Albert Tseng, Tao Yu, and Youngsuk Park. Training LLMs with MXFP4. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 1630–1638, 2025.

Aditya Vavre and Nima Tajbakhsh. Low-precision training with Megatron-Bridge: FP8 (current scaling), MXFP8, NVFP4 vs BF16. GitHub discussion #3460, NVIDIA-NeMo/Megatron-Bridge, `https://github.com/NVIDIA-NeMo/Megatron-Bridge/discussions/3460`, 2026. Posted 2026-04-21.

Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zhengjun Zha, and Peng Cheng. Optimizing large language model training using FP4 quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 62937–62957, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

A TRAINING SETUP AND FP4 SIMULATION

Models. All models are Llama-style decoder-only transformers with pre-normalization. Each block has RMSNorm (Zhang & Sennrich, 2019) (FP32 gain, $\epsilon = 10^{-6}$) before attention and before the MLP, rotary position embeddings (Su et al., 2024) (base 10,000) on queries and keys, causal multi-head attention, and a SwiGLU MLP (Shazeer, 2020) with hidden size $\lfloor \frac{8}{3}d/64 \rfloor \cdot 64$. No linear layer has a bias. A final RMSNorm precedes the output layer, whose weight is tied to the token embedding. The token embedding is initialized with standard deviation 0.02. Linear weights are initialized with standard deviation $d_{\text{in}}^{-1/2}$, except the attention output and MLP down-projections, which use $0.02/\sqrt{2L}$ for L layers. Blocks are numbered from 0, so the last block is block 7 at 60M, 11 at 130M and 21 at 1B. Table 1 lists the sizes. The scale labels are nominal. The counted parameters, including the tied embedding, are 50.7M, 123.6M and 1,207.5M. The token budgets are 20 tokens per nominal parameter, which is 23.7, 21.0 and 16.6 tokens per counted parameter.

Table 1: Model and training settings per scale. Parameter counts include the tied embedding; the count inside the transformer blocks is in parentheses.

	60M	130M	1B
Width d	512	768	2048
Layers L	8	12	22
Attention heads (head dimension)	8 (64)	12 (64)	16 (128)
MLP hidden size	1344	2048	5440
Parameters	50.7M (24.9M)	123.6M (85.0M)	1,207.5M (1,104.5M)
Training tokens	1.2B	2.6B	20B
Sequences per step (tokens per step)	32 (32,768)	24 (24,576)	256 (262,144)
Training steps	36,622	105,795	76,294
Warmup steps (1%)	366	1,057	762
Peak learning rate	1×10^{-3}	6×10^{-4}	3×10^{-4}
BF16 reference seeds	8	2	2
GPUs (data-parallel)	1	1	4

Numerics outside the quantized GEMMs. Linear and embedding weights and their AdamW states are stored in BF16. There is no FP32 master copy. Matrix products run in BF16. The cross-entropy loss is computed in FP32. The BF16 reference runs the same code with no tensor rounded.

Data. The data are FineWeb-Edu documents (Penedo et al., 2024) tokenized with the GPT-2 BPE tokenizer (Radford et al., 2019). An end-of-text token follows every document. The embedding table has 50,304 rows (the tokenizer has 50,257 tokens). Each training sequence is a 1,024-token window drawn at a uniformly random position from a fixed tokenized subset of FineWeb-Edu (about 4B tokens at 60M and 130M, 21.1B at 1B), so sequences can cross document boundaries. The seed sets both the initialization and the order of the drawn windows.

Optimization. All runs use AdamW with $\beta = (0.9, 0.95)$, $\epsilon = 10^{-8}$ and weight decay 0.1 on all parameters. The global gradient norm is clipped to 1.0. The learning rate warms up linearly over the first 1% of steps and then follows a cosine decay to 5% of the peak at the last step. The number of steps is the token budget divided by the tokens per step, rounded up.

Schedule variants at 60M. Four 60M variants change only the schedule (Appendix E). Each has its own one-seed BF16 reference with the same schedule. (i) Constant: the learning rate stays at the

peak after warmup. (ii) Warmup–stable–decay: the learning rate stays at the peak for the first 80% of the post-warmup steps and then decays linearly to zero. (iii) Halved final learning rate: the cosine decays to 2.5% of the peak instead of 5%. (iv) Twice the tokens: a cosine schedule over 2.4B tokens (73,243 steps) with the same batch and peak learning rate.

GPUs and the three 1B windows. Since NVFP4 is simulated in BF16 (below), no run needs FP4 hardware. Each 60M and 130M run used one GPU. Each 1B run used four GPUs, data-parallel over four ranks of 64 sequences each. The 1B runs were done in three training windows. Window 1 ran the two BF16 seeds, full NVFP4, activations and weights, gradients only (seed 0), activations only and weights only. Window 2 ran gradients only (seed 1), MXFP8 forward, the 3% BF16 tail, the switch to BF16 at 50% and the last-block-in-BF16 chain (seed 0). Window 3 ran the last-block-in-BF16 chain with seed 1. Window 1 ran each rank's 64 sequences as one micro-batch; windows 2 and 3 split them into eight micro-batches of 8 sequences. The global batch, the token stream and the gradient are the same in both cases; only the order of floating-point summation differs. The resulting offset between windows is in Appendix C. We compared the model, quantizer and rounding code of windows 1 and 2 file by file and found them identical. The window-3 configuration matches except for the micro-batch count.

Why NVFP4 is simulated. A native NVFP4 GEMM takes both of its inputs in NVFP4, while most of our runs round only some GEMM inputs: the activations-only run, for example, rounds x_{fwd} and keeps W_{fwd} in BF16. Simulation rounds each GEMM input on its own and runs every GEMM in BF16, so two runs differ only in which inputs are rounded.

Where rounding is applied. Every linear layer inside the transformer blocks is simulated: the query, key, value and output projections and the three MLP projections (7 layers per block). The token embedding, the tied output layer, all RMSNorm layers, the rotary embedding and the two attention products (query–key scores and score–value) stay in BF16. Each linear layer runs its three GEMMs through one autograd function. Before each GEMM, each of its two inputs can be replaced by its quantize–dequantize image. The dequantized input is stored in BF16 and the GEMM runs in BF16. A run is defined by the set of GEMM inputs that are rounded, and the same set is used in every simulated layer (Table 2).

Why each GEMM quantizes its own copy. A GEMM can use an NVFP4 input only if the input's 16-value blocks run along the dimension the GEMM sums over, because each block scale must factor out of the sum over its block. The forward GEMM sums x over features (d_{in}) and the backward GEMM for $\mathrm{d}W$ sums it over tokens (BT), so the two GEMMs need blocks along different dimensions, and each quantizes x afresh from its BF16 values; the backward GEMM does not reuse the forward copy. The same holds for W (the forward GEMM sums over d_{in}, the backward GEMM for $\mathrm{d}x$ over d_{out}) and for g (the backward GEMM for $\mathrm{d}x$ sums over d_{out}, the one for $\mathrm{d}W$ over BT). Table 2 lists the block dimension of each copy.

Table 2: The six GEMM inputs of a linear layer with activation $x \in \mathbb{R}^{BT\times d_{\mathrm{in}}}$, weight $W \in \mathbb{R}^{d_{\mathrm{out}}\times d_{\mathrm{in}}}$ and gradient $g \in \mathbb{R}^{BT\times d_{\mathrm{out}}}$ (BT = tokens in the batch). The blocks run along the dimension the GEMM sums over. The two copies of g are told apart only here, by the gradient they compute. RN = round-to-nearest; SR = stochastic rounding (Appendix B).

GEMM	GEMM input	Tensor	16-value blocks along	Rounding
Forward, $y = xW^\top$	x_{fwd}	x	d_{in} (last dim.)	RN
	W_{fwd}	W	d_{in} (last dim.)	RN
Backward, $\mathrm{d}x = gW$	$g_{\mathrm{d}x}$	g	d_{out} (last dim.)	SR
	W_{bwd}	W	d_{out} (first dim.)	RN
Backward, $\mathrm{d}W = g^\top x$	$g_{\mathrm{d}W}$	g	BT (first dim.)	SR
	x_{bwd}	x	BT (first dim.)	RN

NVFP4 quantizer. Table 3 summarizes the recipe. Elements are E2M1 numbers with magnitudes $\{0, 0.5, 1, 1.5, 2, 3, 4, 6\}$. Each rounded tensor T first gets an FP32 per-tensor scale

Table 3: The FP4 recipe.

Format	NVFP4: E2M1 values, one E4M3 scale per 16-value block, one FP32 scale per tensor
Rounded	every linear layer inside the transformer blocks (7 per block)
Kept in BF16	embeddings, output layer, normalization, the two attention products
Rounding	RN for activations and weights; SR for gradients
Simulation	quantize, dequantize to BF16, GEMM in BF16
Not used	random Hadamard transform, 16×16 weight blocks, blocks kept in higher precision

$s_T = \max|T|/(6 \cdot 448)$. Each block b of 16 elements along the block axis gets the scale $s_b = \mathrm{E4M3}\big(\max_{i \in b}|T_i|/(6\,s_T)\big) \cdot s_T$, where $\mathrm{E4M3}(\cdot)$ rounds to the nearest E4M3 value (ties to even) after clamping to $[2^{-9}, 448]$. Each element is divided by s_b, its magnitude is clamped to 6, and it is rounded onto the E2M1 grid. The result is multiplied by s_b and cast to BF16. RN rounds to the nearest grid point, with ties going to the even E2M1 encoding, as in torchao. SR rounds a value v between grid points $\ell < v < h$ up to h with probability $(v-\ell)/(h-\ell)$ and down to ℓ otherwise. When E4M3 rounding lowers s_b, the block maximum exceeds 6 after division and is clipped. The inputs of the backward GEMM for $\mathrm{d}W$ are blocked along the BT axis, which requires BT to be a multiple of the block size; it is at every scale.

Variants used by single runs. (i) SR on activations and weights. (ii) Scale round-up: s_b is rounded up to the next E4M3 value instead of to the nearest, so no element clips. (iii) Hadamard rotation: the forward inputs x_{fwd} and W_{fwd} of every MLP down-projection are multiplied by a fixed 16×16 orthonormal Hadamard matrix along the reduction dimension before rounding; $Q(xH)\,Q(WH)^\top$ equals $xW^\top$ apart from rounding. (iv) RN on both copies of the gradient, $g_{\mathrm{d}x}$ and $g_{\mathrm{d}W}$.

MXFP4 and MXFP8. MXFP4 uses E2M1 elements in blocks of 32 along the same axes, with a power-of-two (E8M0) block scale $2^{\lfloor \log_2 \max|T_b| \rfloor - 2}$ (the OCP floor rule (Open Compute Project, 2023)) and no per-tensor scale. Elements above 6 after division clip to 6. MXFP8 uses the same block size and scale rule with exponent offset 8 and E4M3 elements. The E4M3 cast rounds to nearest (ties to even) and saturates at 448. MXFP8 is used only with RN. With MXFP8 forward, x_{fwd} and W_{fwd} are MXFP8 and the four inputs of the backward GEMMs keep the NVFP4 recipe. On synthetic data, the mean squared rounding error of NVFP4 is 10.5 times that of MXFP8 for Gaussian inputs and 5.6 times for heavy-tailed inputs.

Validation. (1) Parity with torchao (0.18.0, CPU): our MXFP4 quantizer is bitwise identical to torchao's. Our NVFP4 quantizer agrees up to a 0.05% residual mismatch, which comes from one-unit differences in the scale path. (2) SR is unbiased: over 400 draws the mean rounding error is 8% of the RN residual, and it falls as the number of draws grows. (3) Each GEMM input acts only on its GEMM, by unit test with a fixed upstream gradient: rounding x_{fwd} and W_{fwd} changes y and leaves $\mathrm{d}x$ and $\mathrm{d}W$ bitwise unchanged; rounding W_{bwd} changes only $\mathrm{d}x$; rounding x_{bwd} changes only $\mathrm{d}W$. With no tensor rounded, the layer is bitwise a plain BF16 matrix product. (4) Block axes: the recipe blocks W_{bwd}, $g_{\mathrm{d}W}$ and x_{bwd} along their first dimension, the dimension their GEMMs sum over. (5) Logging does not change training: the 60M activations-only run with per-layer error logging ends at $\Delta = 0.0466$, against 0.0485 for the same seed without logging, a difference below the seed variation δ.

B Rounding modes of the three tensors

The two roundings. Let v be a value, after division by its block scale, that lies between two adjacent points $\ell < v < h$ of the E2M1 grid. Rounding to nearest (RN) returns whichever of ℓ and h is closer. Stochastic rounding (SR) returns h with probability $(v-\ell)/(h-\ell)$ and ℓ otherwise, so its expected value is v itself. Both keep the block scale, the clipping at 6 and the grid of Appendix A; they differ only in how the last step onto the grid is taken.

Bias. RN is a fixed function of v: a value at a given position between two grid points is rounded the same way, with the same error, every time it occurs. When the same or similar values are rounded in many steps, these errors lean the same way and do not average out. SR has no bias: its error has mean zero for every v, so errors from different steps average out as they accumulate. Appendix A (validation (2)) checks this for our quantizer: the mean error over 400 SR draws is 8% of the RN error and falls as the number of draws grows.

Mean squared error. SR pays for the absence of bias with a larger error. For a value at distance $a = v - \ell$ above the lower grid point in a step of size $h - \ell$, the squared error is $\min(a, h - \ell - a)^2$ under RN and has expected value $a\,(h - \ell - a)$ under SR. If a is spread uniformly over the step, the mean squared error is $(h - \ell)^2/12$ under RN and $(h - \ell)^2/6$ under SR, twice as large. With our NVFP4 quantizer on synthetic tensors the ratio is 2.1 for Gaussian values and 2.2 for heavy-tailed values (Student's t with three degrees of freedom).

Why activations and weights use RN. The activations and weights enter the forward GEMM, and the gap they produce follows the mean squared error of their rounding, not its bias (Section 6). RN has the smaller error, and SR on the activations and weights makes their gap 3.3 times larger at 60M (0.221 against 0.066 nats; Table 5).

Why gradients use SR. The gradients enter only the backward GEMMs, whose outputs are summed into the weights over all later steps. An error that leans the same way in every step therefore adds up in the weights, while an unbiased error averages out. With RN on both copies of g, the gap from rounding gradients at 60M grows through training to 0.057 nats; with SR it stays constant, at 0.010 nats at the end (Tables 4 and 5). This split, RN for activations and weights and SR for gradients, is also the one used by published NVFP4 training recipes (NVIDIA et al., 2025; Chmiel et al., 2025).

C Measuring the gap, and all runs

Smoothing. Every run logs the training loss of each step's global batch. We smooth each run's loss with a trailing 1,000-step running mean; over the first 1,000 steps the window is all steps so far. The gap at step t is the smoothed loss of the run minus the mean of the smoothed losses of the BF16 reference seeds at the same step. For a run with several seeds we average the smoothed losses of its seeds. Runs of one scale that share a schedule have the same number of steps, so a fraction of training is the same step in each of them. We chose the window on the eight 60M BF16 seeds. The across-seed standard deviation of the final loss is 0.0593 nats for the raw last step, 0.0038 for a 100-step mean, 0.0033 for 200 steps, 0.0032 for 500 and 1,000 steps, and 0.0040 for 2,000 steps. The raw last step is about 18 times noisier than the 1,000-step mean. The learning rate changes by 3.6% across the last 1,000 steps.

Seed variation. At 60M, eight BF16 seeds end at a smoothed loss of 3.5637 with standard deviation 0.0032 nats (range 0.0105). The standard deviation of the difference between two seeds is $\sqrt{2} \times 0.00323 = 0.0046$; we round it to $\delta = 0.005$. The band does not widen during training: its standard deviation is 0.0045–0.0065 in the first half and 0.0032–0.0044 in the last quarter. No seed is an outlier (Grubbs test, $n = 8$, $\alpha = 0.05$). All eight seeds ran on one machine. Repeating an identical configuration on another machine changed the gap by 0.001–0.002 at 60M and by 0.004 at 130M. At 130M, the two BF16 seeds differ by 0.0107, 0.0088 and 0.0097 at 10%, 50% and 100% of training; we use $\delta = 0.010$. At 1B, the two BF16 seeds differ by 0.0025 to 0.0051 on 500-step windows between 10% and 100% of training; we use $\delta = 0.005$. The seed changes the initialization and the data order together. Seed spreads of single runs can exceed δ: the 60M weights-only run has a standard deviation of 0.0050 over three seeds, and the two 130M activations-only seeds are 0.0135 apart. A difference between two gaps smaller than δ is not read as an effect.

Fractions of the gap. The fraction of the gap that a run produces and the fraction that an allocation removes both divide by the full-NVFP4 gap of the same scale at the same step, averaged over its seeds (two at 60M and 130M, one at 1B). For a single-run allocation, the error bar $\pm\delta/\Delta$ in Figure 5 is the seed variation expressed as a fraction of the gap: 7, 16 and 6 points at 60M, 130M and 1B.

Chained runs. Precision switches and BF16 tails are chained runs. A prefix run trains to a fraction of training and saves the model, optimizer and random-number states. A successor run loads that checkpoint, changes which tensors are rounded, and continues on the same learning-rate schedule. Before the resume, a chained run reproduces the continuous run of the same configuration. At 60M, re-run prefixes differ from the original runs by at most 0.005 at the switch step (0.001 or less for the full-NVFP4 seed-0 prefixes at 80% and 90%). At 130M, re-run prefixes differ from the original runs by 0.004. After the resume, the data sampler is reseeded, so a successor sees different batches from the continuous run. In the one direct test (the 60M last-block-in-BF16 run, seed 1), a continuous run and a run resumed at 97% agree within 0.0003 at the checkpoints through 97% and end 0.0047 apart. Two successors resumed from the same checkpoint see the same batches, so a BF16 tail and its NVFP4 continuation form a paired comparison. At 1B the shared steps of every prefix and successor are byte-identical.

Offset between 1B training windows. Window-2 and window-3 runs used eight micro-batches per GPU, window-1 runs one (Appendix A). On the same seed and data, window-2 runs sit below window-1 runs: by 0.0031 for the activations-only prefix (steps 20,147–38,146) and by 0.0016 for full NVFP4 (steps 56,005–74,004). The window-3 offset is not measured, since no window-1 run has the same configuration and seed. We do not correct for the offset. It makes the window-2 allocations (MXFP8 forward, the 3% tail, the 50% switch) read up to 0.003 nats better against window-1 references. For MXFP8 forward at 1B this is 4 points of removed fraction, which moves it further under its upper bound. Two window-2 loss streams each miss 200 logged steps (steps 9,801–10,000 of the BF16-tail chain and 65,501–65,700 of the seed-0 last-block chain); the running mean uses the logged steps of each window.

All runs. BF16 references: 8 seeds at 60M, 2 at 130M, 2 at 1B (cosine), and one seed for each 60M schedule variant. The analysis runs that split the gap by tensor are in Table 4. The single-change runs are in Table 5, the schedule runs in Table 7, and the allocation runs in Table 8. Two 130M full-NVFP4 runs with per-tensor error logging supply the clipping rate in Section 6: a 750M-token run and a 1,017-step continuation of it with denser logging. No gap is reported for them. G denotes the growth $\Delta(100\%) - \Delta(20\%)$; for runs with several seeds it is computed on the mean over seeds, and it is not given for runs whose precision changes during training.

Table 4: Analysis runs that split the gap by tensor. Each run rounds the listed tensors (both copies) or GEMM inputs in every simulated layer with the recipe's rounding (RN for activations and weights, SR for gradients) and leaves the others in BF16. At 1B, the second gradients-only seed is from training window 2; all other 1B runs are from window 1.

Run	Rounded	Scale	Seeds	Δ per seed	Mean	G
Full NVFP4	x, W, g	60M	2	0.0742, 0.0738	0.0740	0.0230
		130M	2	0.0557, 0.0656	0.0607	0.0177
		1B	1	0.0863		0.0390
Activations and weights	x, W	60M	2	0.0666, 0.0653	0.0660	0.0201
		130M	2	0.0459, 0.0463	0.0461	0.0124
		1B	1	0.0360		0.0069
Gradients only	g	60M	2	0.0125, 0.0068	0.0097	0.0042
		130M	2	0.0074, 0.0190	0.0132	0.0027
		1B	2	0.0307, 0.0277	0.0292	0.0126
Activations only	x	60M	2	0.0485, 0.0428	0.0457	0.0200
		130M	2	0.0273, 0.0408	0.0341	0.0126
		1B	1	0.0254		0.0078
Weights only	W	60M	3	0.0253, 0.0157, 0.0183	0.0198	-0.0013
		130M	2	0.0128, 0.0098	0.0113	-0.0014
		1B	1	0.0140		-0.0018
x_{fwd} only	x_{fwd}	60M	1	0.0464		0.0198
x_{bwd} only	x_{bwd}	60M	1	0.0055		0.0011

Table 5: Single-change runs. All are 60M with one seed, except the last switch row (1B, training window 2). Each changes one thing relative to the run in its first column.

Run	Change	Δ	G
Rounding and format			
Activations and weights	SR instead of RN	0.2206	0.1167
Activations and weights	MXFP4 instead of NVFP4	0.1036	0.0366
Full NVFP4	block scale rounded up (no clipping)	0.0767	0.0204
Full NVFP4	Hadamard rotation in down-projections	0.0937	0.0368
Gradients only	RN instead of SR	0.0573	0.0465
Precision switches and logging			
Activations only	per-layer error logging	0.0466	0.0204
Activations only	BF16 from 50% of training	0.0168	–
Activations only	BF16 until 50%, activations only after	0.0350	–
Activations only (1B)	BF16 from 50% of training	0.0088	–
Error injected into a BF16 run			
Unbiased noise on g (both copies)	$1\times$ measured size, cosine	0.0089	0.0026
	$1\times$, constant learning rate	0.0024	-0.0025
	$1\times$, warmup–stable–decay	0.0083	0.0004
	$2\times$, constant learning rate	0.0246	0.0002
	$2\times$, warmup–stable–decay	0.0252	0.0017
	$4\times$, cosine	0.1048	-0.0014

D THE SPLIT BY TENSOR AT EACH SCALE

This appendix gives the analysis runs that split the gap by tensor, behind Section 4 and the first paragraph of Section 6. Figure 7 shows the activations-only and weights-only runs over training and all runs at the end of training. Table 6 gives the seed-mean gaps at five points of training.

Table 6: Seed-mean gap (nats) at five points of training. "Sum" is the arithmetic sum of the activations-and-weights and gradients-only gaps.

Scale	Run	10%	20%	50%	80%	100%
60M	Full NVFP4	0.0419	0.0510	0.0686	0.0696	0.0740
	Activations and weights	0.0359	0.0458	0.0618	0.0615	0.0660
	Gradients only	0.0050	0.0054	0.0105	0.0062	0.0097
	Sum	0.0408	0.0513	0.0723	0.0678	0.0756
	Activations only	0.0197	0.0257	0.0385	0.0426	0.0457
	Weights only	0.0185	0.0211	0.0250	0.0190	0.0198
130M	Full NVFP4	0.0411	0.0430	0.0518	0.0600	0.0607
	Activations and weights	0.0322	0.0336	0.0398	0.0454	0.0461
	Gradients only	0.0094	0.0105	0.0125	0.0131	0.0132
	Sum	0.0416	0.0441	0.0523	0.0585	0.0592
	Activations only	0.0203	0.0215	0.0272	0.0332	0.0341
	Weights only	0.0145	0.0127	0.0118	0.0114	0.0113
1B	Full NVFP4	0.0410	0.0473	0.0652	0.0840	0.0863
	Activations and weights	0.0257	0.0291	0.0322	0.0360	0.0360
	Gradients only	0.0151	0.0167	0.0213	0.0284	0.0292
	Sum	0.0408	0.0458	0.0535	0.0645	0.0652
	Activations only	0.0143	0.0177	0.0216	0.0253	0.0254
	Weights only	0.0150	0.0159	0.0145	0.0144	0.0140

Activations and weights plus gradients. At 60M and 130M the sum of the activations-and-weights and gradients-only gaps matches full NVFP4 within δ at all five points (largest difference 0.0037 at 60M and 0.0015 at 130M). At 1B the two agree at 10% and 20% of training and separate afterwards, to 0.0117 at 50% and 0.0211 at the end. Full NVFP4 rises gradually over this stretch:

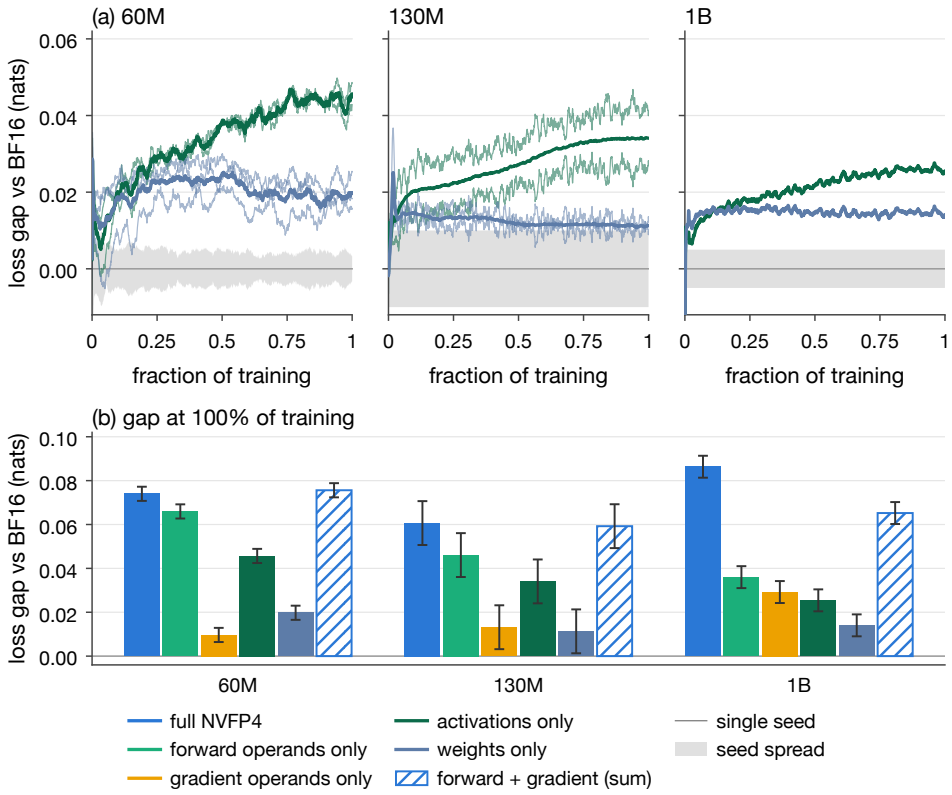

Figure 7: (a) Activations only and weights only over training; thick lines are seed means, thin lines single seeds. (b) All these runs at the end of training, with the arithmetic sum of activations and weights and gradients only (hatched). Bands and error bars: at 60M the across-seed standard deviation of the eight BF16 seeds; at 130M and 1B the seed variation $\pm\delta$.

between 38% and 63% of training, its largest step between adjacent 1% points is 0.003 (500-step windows), so no single event produces the separation.

Activations plus weights. The activations-only and weights-only gaps add up to the activations-and-weights gap at every scale: 0.0655 against 0.0660 at 60M, 0.0454 against 0.0461 at 130M and 0.0394 against 0.0360 at 1B. The weights-only gap does not grow at any scale ($G = -0.0013$, -0.0014, -0.0018), and it does not grow on any single seed. The activations-only gap carries the growth of the gap from rounding activations and weights at every scale.

The forward and the backward copy of the activations. At 60M, rounding only x_{fwd}, the copy in the forward GEMM, gives 0.0464 with growth 0.0198. From 5% of training on it stays within 0.0046 of the activations-only run with the same seed (seed 0), below δ. Rounding only x_{bwd}, the copy in the backward GEMM for $\mathrm{d}W$, gives 0.0055 with growth 0.0011 (one seed each). The gap from rounding activations is therefore the cost of rounding the input of the forward GEMM.

E MECHANISM RUNS

This appendix gives the runs behind Section 6. All are 60M with one seed unless stated. Per-run end values are in Table 5.

The size of the rounding error, not its bias. With stochastic rounding on the activations and weights, the gap of the run that rounds only these two tensors is 0.0842 at 10% of training, 0.1674 at 50% and 0.2206 at the end, 3.3 times the run with rounding to nearest, with about 6 times its

growth G. Stochastic rounding is unbiased and has about twice the mean squared error of rounding to nearest (Appendix B). With MXFP4 instead of NVFP4 on the activations and weights, the gap is 0.0560, 0.0905 and 0.1036 at the same points, 1.57 times the NVFP4 run. The three roundings rank in the same order by gap and by mean squared error: NVFP4 with nearest rounding, then MXFP4, then NVFP4 with stochastic rounding. Before the run, an account in which the gap came from the bias of rounding to nearest predicted that stochastic rounding would shrink it; it grew instead. MXFP8, used by MXFP8 forward, has a mean squared error 10.5 times smaller than NVFP4 on Gaussian values and 5.6 times smaller on heavy-tailed values (Appendix A).

Clipping and rotation. In the last 3.3% of a 130M full-NVFP4 run, about 3.4% of the elements of every GEMM input clip, all in blocks whose E4M3 scale rounded down. Rounding the block scale up removes all clipping; the 60M full-NVFP4 gap is then 0.0767 against 0.0740, a difference below δ. A 16×16 Hadamard rotation of the forward inputs x_{fwd} and W_{fwd} of every down-projection raises the gap to 0.0937, as predicted before the run.

Relative error and outliers over training. The relative rounding error (root mean square of the error divided by that of the values) of the 56 quantized weight matrices of the 60M full-NVFP4 run is 0.098 at step 5,000 and 0.105 at the end. In the 60M activations-only run, the relative mean squared error of x_{fwd} (the squared rounding error summed over all layers, divided by the summed squared values) is 0.0079 early (steps $\leq$ 5,475) and 0.0074 late (steps $\geq$ 31,025), a ratio of 0.94; for x_{bwd} the ratio is 1.01. Over the same span the activations-only gap grows 1.6- to 2.2-fold. The kurtosis of the down-projection inputs is 40–150 throughout and does not rise, and the fraction of block maxima that are outliers stays at or below 1%.

Schedules. Table 7 gives the schedule runs. Each is compared with a one-seed BF16 reference of the same schedule. The gap from rounding activations and weights grows under a constant learning rate, grows during the stable phase of a warmup–stable–decay schedule, and levels off when training runs twice as long. Halving the final learning rate leaves the full gap unchanged: at a matched BF16 loss of 3.6 the fitted gaps are 0.0685 and 0.0682.

Table 7: Schedule runs at 60M (one seed each). Gap in nats at four points of training.

Run	Schedule	10%	50%	80%	100%
Activations and weights	cosine, 1.2B tokens (seed 0)	0.040	–	–	0.067
Activations and weights	constant	0.0322	0.0486	–	0.0588
Activations and weights	warmup–stable–decay	0.0351	0.0515	0.0574	0.0641
Activations and weights	cosine, 2.4B tokens	0.0405	0.0613	0.0697	0.0705
Full NVFP4	cosine, final learning rate halved	0.0405	–	–	0.0719

The affine relation within a run. Over the steps after the first 3,000, the gap of each 60M run is an affine function of the BF16 reference loss L of its own schedule (Figure 6). For the run that rounds activations and weights,

$$\Delta_{x,W} = 0.222 - 0.0446\,L, \tag{2}$$

with rms residual 0.0018 nats. For full NVFP4 the fit is $\Delta = 0.256 - 0.0523\,L$ (rms 0.0019); for the run that rounds only the gradients the slope is -0.0028 (rms 0.0017); for full NVFP4 with a halved final learning rate, $\Delta = 0.291 - 0.0617\,L$ (rms 0.0012). Each fit uses one run (seed 0). The slopes of the activations-and-weights run and the gradients-only run add to -0.047, against -0.052 for full NVFP4. At the 1B reference loss of 2.61, Equation (2) gives 0.106 nats, against the 1B gap $\Delta_{x,W} = 0.036$.

The gradients. Unbiased noise added to both copies of the gradient in a BF16 run, with the mean squared error that stochastic rounding produces on them in the NVFP4 recipe, gives a gap of 0.002–0.009 that does not grow under the cosine, constant or warmup–stable–decay schedule, including through the decay phase. At twice the size the gap is about 0.025 and flat; at four times it is 0.105 and flat. Rounding the gradients to nearest produces a growing gap: 0.0081 at 10% of training, 0.0295 at 50% and 0.0573 at the end.

Clipping and outliers, in full. Two features of block formats are commonly held responsible for a low-precision gap: clipping, where a value beyond the largest the format can represent is set to that largest value, and outliers, the few elements of a block far larger than the rest, which set the block's scale and so coarsen the rounding of the others. Neither is the cause here. NVFP4 clips about 3% of the elements of every GEMM input, all in blocks whose scale, an 8-bit E4M3 number shared by the block (Appendix A gives the format), rounds down; rounding the scale up instead removes the clipping and leaves the gap unchanged. A Hadamard rotation, which spreads the outliers of a block over all of its elements before rounding, applied to the forward inputs of the down-projections makes the gap larger (the runs are given above). Among the changes we tried, only the mean squared error moves this gap.

F THE 1B RUNS

This appendix gives the 1B measurements behind Sections 4 and 5 and the gradient paragraph of Section 6. The training windows and their offset are described in Appendices A and C.

The gap from rounding gradients on two seeds. The gradients-only run has two seeds at 1B, seed 0 from window 1 and seed 1 from window 2. Seed 0 goes from 0.0188 at 20% of training to 0.0307 at the end and seed 1 from 0.0146 to 0.0277, a growth of 0.012 and 0.013. The seeds are offset by about 0.004, the size of the seed variation δ, and part of that offset is the window offset. At 60M and 130M the gradients-only gap grows by 0.004 and 0.003 over the same span.

The per-tensor probes do not explain the growth. Probes log the relative mean squared error of every rounded GEMM input during training. For the 1B gradients-only run (seed 0), the median over tensors grows 1.63-fold from 10% to 100% of training, against 1.82- and 1.85-fold for the two 130M gradients-only seeds. The aggregate weighted by each tensor's squared values falls at 1B (0.038 to 0.032), so the two aggregations disagree in sign there. The probe also grows in runs whose loss gap is flat (1.82-fold for 130M gradients only, 1.50-fold for 1B weights only), so its growth does not track the gap. The ratio of the gap to the median relative mean squared error rises at 1B (0.46 to 0.51 for gradients only) and falls at 130M (0.18 to 0.13 and 0.42 to 0.29). A given gradient error costs more loss late in the 1B run. The 1B probes use an 8-row batch, which changes the levels but not the growth ratios, so we compare only growth ratios across scales. We did not analyse the probes of the second 1B seed.

The excess. Full NVFP4 exceeds the sum of activations and weights and gradients only by more than δ from 35% of training on (Table 6); at the end the excess is 0.021, about four times the 1B seed variation. The full and activations-and-weights runs have one seed each. Using the window-2 gradient seed lowers the gradient mean by about 0.0015 through the window offset, small against 0.021.

The last block in BF16 is unstable at 1B. With block 21 in BF16 and the other 21 blocks in NVFP4, both seeds show a gradient-norm event starting at 12,000–14,000 steps (16–18% of training, near the peak learning rate). The global gradient norm reaches the clipping threshold of 1.0 for several thousand steps and returns below 0.5 at about 22,000 steps. The event is confined to the down-projection of block 20 (the last NVFP4 GEMM before the BF16 block) and the query, key and output projections of block 21. The value projection and the MLP of block 21 take no material part (on seed 1 the MLP down-projection briefly reaches 3 times its baseline gradient norm, against about 18 times for the block-20 down-projection). Both seeds end with a gap of 0.18 (500-step means at the end), twice that of full NVFP4. The same setting has no such event at 60M or 130M. We have not established the cause. Attention-logit growth in the BF16 block, a regularizing effect of rounding noise on the attention inputs, and a width or learning-rate threshold are untested hypotheses.

G ALLOCATIONS, TAILS AND SWITCHES

This appendix gives the allocation runs behind Sections 3 and 5, the time course of the BF16 tails, the precision switches and the probe maps used by the depth rules.

All allocation runs. Table 8 lists every allocation run. Cost is the fraction of training GEMM FLOPs that leaves NVFP4. A block's forward GEMMs are one third of its GEMM FLOPs, a tail over $p\%$ of steps is $p\%$ of the FLOPs, and a tail on the forward GEMM only is $p\%/3$. MXFP8 forward runs no BF16 and puts one third of the GEMM FLOPs in MXFP8. The removed fraction uses the full-NVFP4 mean of the same scale (0.0740, 0.0607, 0.0863). The two 1B last-block rows are 500-step means at the end of training; every other value is the 1,000-step running mean at the last step. [a]The 60M last-block row has two seeds; seed 1 was run twice (continuous, and resumed at 97%), and its two runs are averaged before the seed mean.

Table 8: All allocation runs. Gap in nats at the end of training; removed = fraction of the full-NVFP4 gap removed (%).

Scale	Allocation	Cost	Δ per run	Mean	Removed
60M	MXFP8 forward	33% MXFP8	0.0217, 0.0164	0.0190	74
	Forward GEMM in BF16 (all blocks)	33.3%	0.0167		77
	BF16 tail, last 1%	1%	0.0650		12
	BF16 tail, last 3%	3%	0.0569		23
	BF16 tail, last 5%	5%	0.0549, 0.0486	0.0518	30
	BF16 tail, last 10%	10%	0.0545, 0.0493	0.0519	30
	BF16 tail, last 20%	20%	0.0448, 0.0440	0.0444	40
	BF16 tail on the forward GEMM only, last 10%	3.3%	0.0550, 0.0488	0.0519	30
	Last block in BF16	12.5%	0.0515; 0.0494 / 0.0447[a]	0.0493	33
	First block in BF16	12.5%	0.0588		21
	Last block in BF16 from 50%	6.25%	0.0592		20
	Forward GEMM of the last block in BF16	4.2%	0.0571		23
	Backward GEMMs of the last block in BF16	8.3%	0.0719		3
	Forward GEMM of the last 3 blocks in BF16	12.5%	0.0425, 0.0414	0.0420	43
	One linear layer in BF16 (last down-projection)	2.8%	0.0589, 0.0621	0.0605	18
	Last block in BF16 + 3% tail	15.1%	0.0392, 0.0350	0.0371	50
	Forward GEMM of last 3 blocks + forward tail	14.6%	0.0344, 0.0287	0.0315	57
130M	MXFP8 forward	33% MXFP8	0.0127, 0.0302	0.0214	65
	BF16 tail, last 5%	5%	0.0431		29
	BF16 tail, last 10%	10%	0.0438		28
	BF16 tail on the forward GEMM only, last 10%	3.3%	0.0441		27
	Last block in BF16	8.3%	0.0416		31
	Forward GEMM of the last 3 blocks in BF16	8.3%	0.0411		32
	One linear layer in BF16 (probe's top layer)	<2%	0.0530		13
	Last block in BF16 + 3% tail	11.1%	0.0237		61
1B	MXFP8 forward	33% MXFP8	0.0516		40
	BF16 tail, last 3%	3%	0.0655		24
	Last block in BF16 (unstable)	4.5%	0.182, 0.180		–
	Last block in BF16 + 3% tail (unstable)	7.4%	0.139, 0.131		–

How fast a tail recovers. A BF16 tail and its NVFP4 continuation resumed from the same checkpoint see the same batches, so their difference has a window standard error of at most 0.0003. At 60M (20% tail, seed 1) the tail's advantage is 0.011 in the first 100 steps, 0.018 at 200–400 steps, 0.021 at 800–1,000 steps and 0.024 at 3,200–3,662 steps, the last window of its NVFP4 continuation. At 130M (10% tail) it is 0.0075, 0.012 and 0.014 over the same windows and 0.017 at 3,662–5,290 steps, again the continuation's last window. Relative to these last windows, about half of the advantage arrives in the first 100–200 steps and 82–91% by 1,000 steps. The advantage still rises slowly in the last windows; part of that may be the NVFP4 continuation's own growth, which these pairs cannot separate. The 1B 3% tail (2,289 steps) comes from a window-2 prefix; against that prefix's own gap at 97% (0.0854) it removes 0.020, which is 23% of the gap.

Precision switches. Figure 8 shows the switches of the activations-only run at 50% of training. At 60M, switching to BF16 lowers the paired gap from 0.039 in the 500 steps before the switch to 0.016 in the first 100 steps after it. From 2,000 steps after the switch to the end it stays at 0.014–0.016. The mirror run, BF16 until 50% and activations-only after, reaches 0.027 in the first 100 steps and ends at 0.035, 0.012 below the run that was rounded throughout. For full NVFP4, the gap left after a tail is 0.044 for a switch at 80% and 0.052 for switches at 90% and 95% (two-seed means, Table 8). The 0.017 left by the activations-only switch at 50% belongs to a different run and is not compared with these. At 1B the activations-only switch at 50% lowers the gap from 0.0188 in the 500 steps before the switch to 0.0084 in the 500 steps after it, a 55% drop on the same run; against the continuous activations-only run the drop is 62% over the first 500 steps. At the end of training, on the 1,000-step running mean, the switched run is 65% below the continuous one (0.0088 against

0.0254). The continuous run is from window 1, so the window offset makes both comparisons with it read slightly large.

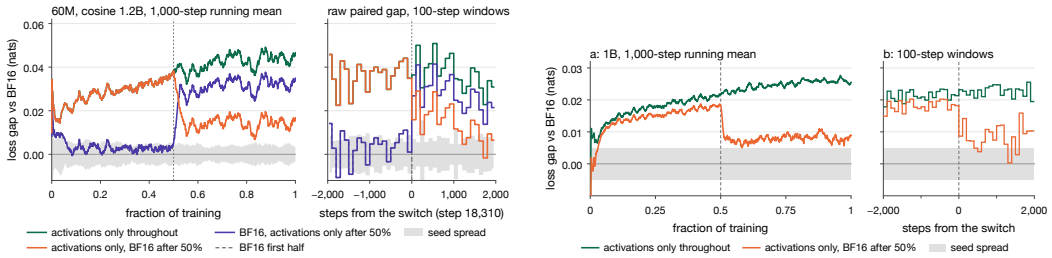

Figure 8: Switches of the activations-only run at 50% of training. Left: 60M, both directions (smoothed gap, and raw paired gap in 100-step windows around the switch). Right: 1B, switch to BF16 (smoothed gap, and 100-step windows). One seed per run.

Probe maps across depth. An evaluation-time probe rounds the forward inputs of one linear layer at a time on a checkpoint and measures the loss change. At 60M the per-layer changes add up to 90–94% of the change with all 56 layers rounded, the top layer is the last block's down-projection (0.010 nats, 13–17% of the total), and the ranking is stable between 50% and 100% of training (Spearman 0.965). In the training-time error log of the activations-only run, the same layer carries 63% of the late squared rounding error of the activations, up from 20% early. At 130M the map is flat: the per-layer changes add up to 0–60% of the all-rounded change on four evaluation sets, the top single change is 0.0006, and rounding the last block's down-projection alone lowers the loss.

The published switch results in our terms. NVIDIA et al. (2025) switch a 12B model from NVFP4 to higher precision at 8.2T of 10T tokens and compare it with an FP8 reference (their Appendix D). The relative loss error is 1.5% before the switch; switching the forward GEMMs alone brings it to 0.5%, switching the backward GEMMs alone leaves it unchanged, and switching both matches the reference. Read with the gaps of Section 2, the forward switch leaves the cost of rounding only the gradients, so $\Delta_g \approx \Delta/3$; the backward switch leaves the cost of rounding only the activations and weights, so $\Delta_{x,W} \approx \Delta$; the excess is then about $-\Delta/3$. Rounding the gradients costs nothing there while the forward GEMM is still rounded and a third of the gap once it is not, a floor set by the larger of the two errors. Since the switches come late in training, these are the recoverable parts of each gap at 82% of training rather than the gaps themselves. Our 1B run has the opposite sign: the excess is $+24\%$ of the gap, and rounding all three tensors costs more than the sum. Three differences could account for the disagreement, and the published data do not separate them. First, the FP8 reference rounds the gradients itself, so the gradient cost that FP8 and NVFP4 share is not in their gap, while ours is measured against BF16. Second, their recipe treats the gradient GEMMs specially: random Hadamard transforms on the inputs of the weight-gradient GEMM, two-dimensional block scaling of the weights so that a tensor and its transpose share their scales, and 16% of the linear layers kept in BF16, all of which lower the gradient error relative to our recipe. Third, the model, a 12B hybrid of Mamba and Transformer blocks, and the amount of training differ from ours. Chmiel et al. (2025), at 7B against a BF16 reference, find the opposite of NVIDIA et al. (2025): the gap closes when the backward GEMMs switch to BF16 late in training, which in our terms is a recoverable gradient gap that is most of the gap at that point. Both results fit the bound of Section 3: each switch removes at most the fraction of the gap that its side produces, and the fractions differ between the two settings.

H Further related work

Studies that vary which tensors are quantized exist across runs. Sun et al. (2025) vary which GEMM inputs are quantized in a scaling law, Chen et al. (2026a) split the error of quantization-aware training into weight and activation parts, Kumar et al. (2025) fit separate sensitivities for weights, activations and the attention cache, and activation quantization was found to cost more than weight quantization in quantization-aware fine-tuning (Rahman et al., 2026). None of them follows the split over the

course of an FP4 pretraining run or switches precision partway through training in both directions, which is what separates the recoverable gap from the persistent gap here.

Precision scaling laws fit the loss penalty of low-precision training as a function of the precision, the model size and the data (Kumar et al., 2025; Sun et al., 2025; Castro et al., 2025; Chen et al., 2026a). Quartet uses one factor for the forward pass and one for the backward pass, both independent of model size (Castro et al., 2025). Our fractions are what such factors have to reproduce, and between 130M and 1B the gap from rounding activations and weights and the gap from rounding gradients move in opposite directions.

The allocations we test are in use. Practitioners keep the last blocks in BF16 (NVIDIA et al., 2025; Vavre & Tajbakhsh, 2026), run a higher-precision phase at the end of training (Chmiel et al., 2025; NVIDIA et al., 2025; Marin Community, 2026), and pair 8-bit forward GEMMs with 4-bit backward GEMMs (Tseng et al., 2025). Each is chosen from ablations at one scale; Section 3 says what each can remove and Section 4 which of these bounds moves with scale.

Two of our mechanisms have counterparts in the literature. That unbiased noise in the updates costs a constant offset in the loss follows the stationary-loss analysis of adaptive optimizers under gradient noise (Compagnoni et al., 2025) and the noise threshold of Chmiel et al. (2025); stochastic rounding itself goes back to Gupta et al. (2015). Hadamard rotations were introduced to remove outliers before 4-bit inference (Ashkboos et al., 2024) and are part of several recipes (NVIDIA et al., 2025; Castro et al., 2025; Cim et al., 2026); on the forward inputs of our down-projections a rotation makes the gap larger, which is consistent with the gap following the mean squared error rather than the outliers. We know of no published account of why the loss becomes more sensitive to a forward rounding error of fixed size as training converges, or less sensitive as the model grows.