Images

A training step has one cost that scales with the size of the input and dominates everything else on an encoder-decoder with skip connections: the activation memory held alive from the forward pass for the backward pass to consume. For a raster well-log archive whose widest image is 12,800 pixels across by up to 640 tall, that single term is what breaches the GPU memory ceiling, and it does so at batch size 1, where the usual relief valves of smaller batches and gradient accumulation have nothing left to give. This whitepaper is the first-person methodology account of recovering the headroom from the numerics rather than from the batch size or the wall clock. The central move is to store activations in 16 bits instead of 32 under automatic mixed-precision autocast, which halves the dominant memory term at 2 bytes per value against 4, while keeping the loss, the normalisation statistics, and the master weights in fp32 so accuracy does not move. We set out the activation-memory inequality the whole problem is governed by, the two half-precision formats the loop can choose between, and the reason the choice between them is a numerics decision and not a convenience: bfloat16 keeps the full fp32 exponent range with 8 exponent bits and needs no gradient rescue, while fp16 keeps only 5 exponent bits, underflows small gradients to zero, and must be paired with a dynamic loss scaler that multiplies the loss by a power of two before the backward pass and divides it out before the optimiser step. We walk three instruments that argue the budget, the underflow mechanism, and the format anatomy, then state the operating posture we adopted for the 12,800-pixel worst case at batch size 1, the GroupNorm group-size floor of 16 that keeps normalisation valid at that tiny physical batch, and the failure modes that send a half-precision run divergent. The deliverable is a method for buying back activation headroom with numerics that any team training on variable-dimension imagery under a hard memory ceiling can reuse.

Narendra Patwardhan
Research Collaborator

Tannistha Maiti
Senior AI Researcher

Tarry Singh
Founder & CEO

VEERNET / MIXED-PRECISION / FP16 / BF16 / AUTOCAST / LOSS-SCALING

CONTENTS

> "
>
> The widest log in the archive does not care about the batch size. It is 12,800 pixels across, and the activation memory of one pass over it is the number that touches the ceiling. We bought the headroom back from the numerics.
>
> "

I

THE LEVER

When the input is too wide, change the bytes, not the batch

What we were actually short of

We built VeerNet, our encoder-decoder segmentation network with a transformer attention refinement on the bottleneck, to digitise a raster archive of scanned well logs. The geoscience that makes the work worth doing, and the choice to grade the digitiser on the reconstructed curve rather than the segmentation mask, are subjects of their own. This document is about a single engineering constraint that shaped every training decision and that we solved with numerics: the widest image in the archive is roughly 12,800 pixels across, against a height that runs only 480 to 640 pixels, and the memory that a single forward and backward pass commits over an image that wide is what breaches the device ceiling.

The reflex relief valves do not apply here. When a model runs out of memory, the standard answers are to shrink the batch, accumulate gradients across several small steps, or shrink the model. We had already been pushed to the floor on the first two: the images have no fixed aspect ratio, so they cannot be stacked into a batch tensor at all, and the binary segmentation stage was therefore forced to batch size 1. There is no smaller batch than one. Gradient accumulation changes how many steps you average over; it does nothing for the memory of the single step that is already too big. And shrinking the model was off the table because the architecture was doing its job on the curve metrics. The thing we were short of was the activation memory of one step over the widest log, and the only lever still untouched was how many bytes each activation occupies.

That lever is mixed precision. Store the activations in 16 bits instead of 32 and the dominant memory term halves, because every value that used to take 4 bytes now takes 2. Keep a full-precision master copy of the weights and keep the loss in fp32, and the accuracy does not move [1]. This is not a clever trick we invented; it is the standard mixed-precision recipe,

and the contribution of this whitepaper is the discipline of applying it as the primary lever for a memory ceiling rather than the throwaway speed optimisation it is usually treated as, plus the format decision that the ceiling forces you to make consciously.

THE SINGLE STEP, BEFORE AND AFTER THE NUMERICS CHANGE

12,800 px
Widest scanned log, the worst-case input

1
Batch size the variable widths already force

4 to 2
the lever
Bytes per activation, fp32 to half precision

fp32
accuracy held
Master weights and loss stay full precision

The rest of this document treats numerics as a budget. We define the memory inequality the budget has to satisfy, show how half precision moves the binding term, and then make the case that the choice of half-precision format is not interchangeable: one of the two formats needs a loss scaler and one does not, for a reason that is visible in the bits.

Why this is the right lever and not just a faster one

Mixed precision is most often sold as a speed feature, because 16-bit matrix multiplies run faster on hardware with the right units. Speed was welcome, but it was not why we reached for it. We reached for it because it is the cheapest lever that moves the binding constraint

without touching the model or the data. Gradient checkpointing also relaxes the activation ceiling, by recomputing activations in the backward pass instead of storing them, and we hold it in reserve [7]; but it trades compute for memory and slows the step, whereas halving the byte width buys memory and speed at once and only asks you to be careful about numerics. When the constraint is memory and the architecture is fixed, the order of levers we would reach for is: cast the activations to 16 bits first, then add checkpointing if that is still not enough. This piece is about getting the first lever right.

II

THE BUDGET

The activation term is the one that breaches the ceiling

The memory inequality the whole problem obeys

A training step has a memory footprint with two kinds of terms, and only one of them is the problem. The fixed terms do not change with the input: the model parameters, the gradient buffers, the optimiser state. You pay them once, they are bounded by the size of the network, and on a model the size of VeerNet they are real but not what breaches the ceiling. The per-batch term is the activation footprint, every intermediate tensor the forward pass produces and the backward pass keeps alive to compute gradients, and on an encoder-decoder with skip connections it is large, because the high-resolution activations from the early encoder stages are held alive through the entire decode path to be concatenated back in [4]. The constraint that has to hold is simply that the sum fits.

THE TRAINING STEP HAS TO FIT UNDER THE DEVICE CEILING

Formula LaTeX

Copy LaTeX

$$M_{\text{params}} + M_{\text{grads}} + M_{\text{opt}} + M_{\text{act}}(W, H, b) \;\leq\; M_{\text{device}}$$

The first three terms are constants for a given network. The fourth, the activation memory, is a function of the image width, the image height, and the batch size, and for our archive the width runs to 12,800 pixels. At batch size 1, the only free variables left in that term are the geometry, which we cannot change without throwing away the data, and the bytes per activation value, which we can. Writing the activation term explicitly as a byte width times a count of activation values makes the lever obvious.

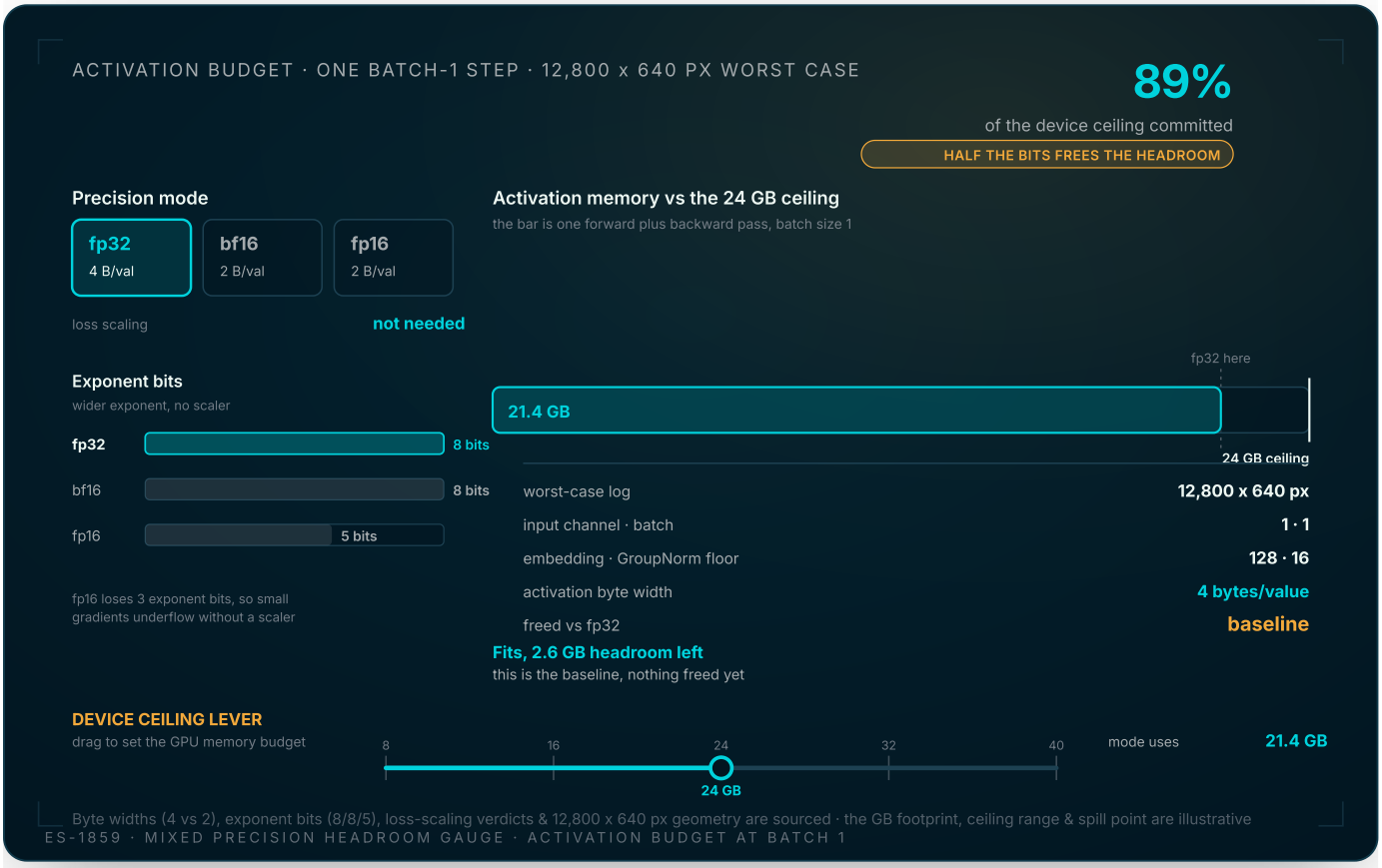
ACTIVATION MEMORY FACTORS INTO A BYTE WIDTH AND AN ACTIVATION COUNT

Formula | LaTeX

Copy LaTeX

$$M_{\text{act}} = s \cdot A(W, H, b), \qquad s = \begin{cases} 4 & \text{fp32} \\ 2 & \text{fp16 or bf16} \end{cases}$$

The activation count is fixed by the architecture and the input. The byte width is a numerics decision. Moving it from 4 to 2 halves the dominant term in the inequality, and for the worst-case log that is the difference between a step that fits and a step that does not. The first instrument reads that budget directly.



An activation-memory budget bar for one forward plus backward pass on the worst-case scanned log, 12,800 by 640 px at batch size 1, drawn as the precision mode flips fp32 to bf16 to fp16. Pick a mode and the bar shows how much of the chosen device ceiling the step commits: fp32 stores activations at 4 bytes per value, bf16 and fp16 at 2, so half-precision frees the headroom the worst-case log needs. Each mode is paired with its loss-scaling verdict and exponent-bit width: fp32 and bf16 carry 8 exponent bits and need no scaler, while fp16 carries only 5 and underflows small gradients without a dynamic loss scaler. Drag the ceiling lever to read the same footprint against a different GPU memory budget. The byte widths, exponent-bit counts, loss-scaling verdicts, and log geometry are sourced from the engagement archive; the absolute gigabyte footprint, the ceiling range, and the exact spill point are illustrative schematics of the budget.

The gauge makes the lever concrete. Pick fp32 and the activation bar for one batch-size-1 step over the 12,800 by 640 worst case sits high against the device ceiling. Flip to bf16 or fp16 and the same step drops to half the footprint, because the byte width halved, and the headroom it frees is the headroom the worst-case width needs. The fp32 ghost marker stays on the bar so the freed amount is always legible as a distance, not just a smaller number, and the ceiling lever lets you read the same footprint against whatever device budget you actually have to live under. The two half-precision modes free the same memory; what separates them is not in this picture, it is in the gradients, and that is the next instrument.

> **THE POSTURE WE ADOPTED ON THE BUDGET**
>
> At batch size 1 there is no batch lever left, so the activation byte width becomes the primary memory control. Cast the activations to 16 bits through autocast first; reserve gradient checkpointing for the case where half precision alone still does not fit. Read every memory decision against the widest log in the archive, not the average, because the widest member is what sizes the step.

III

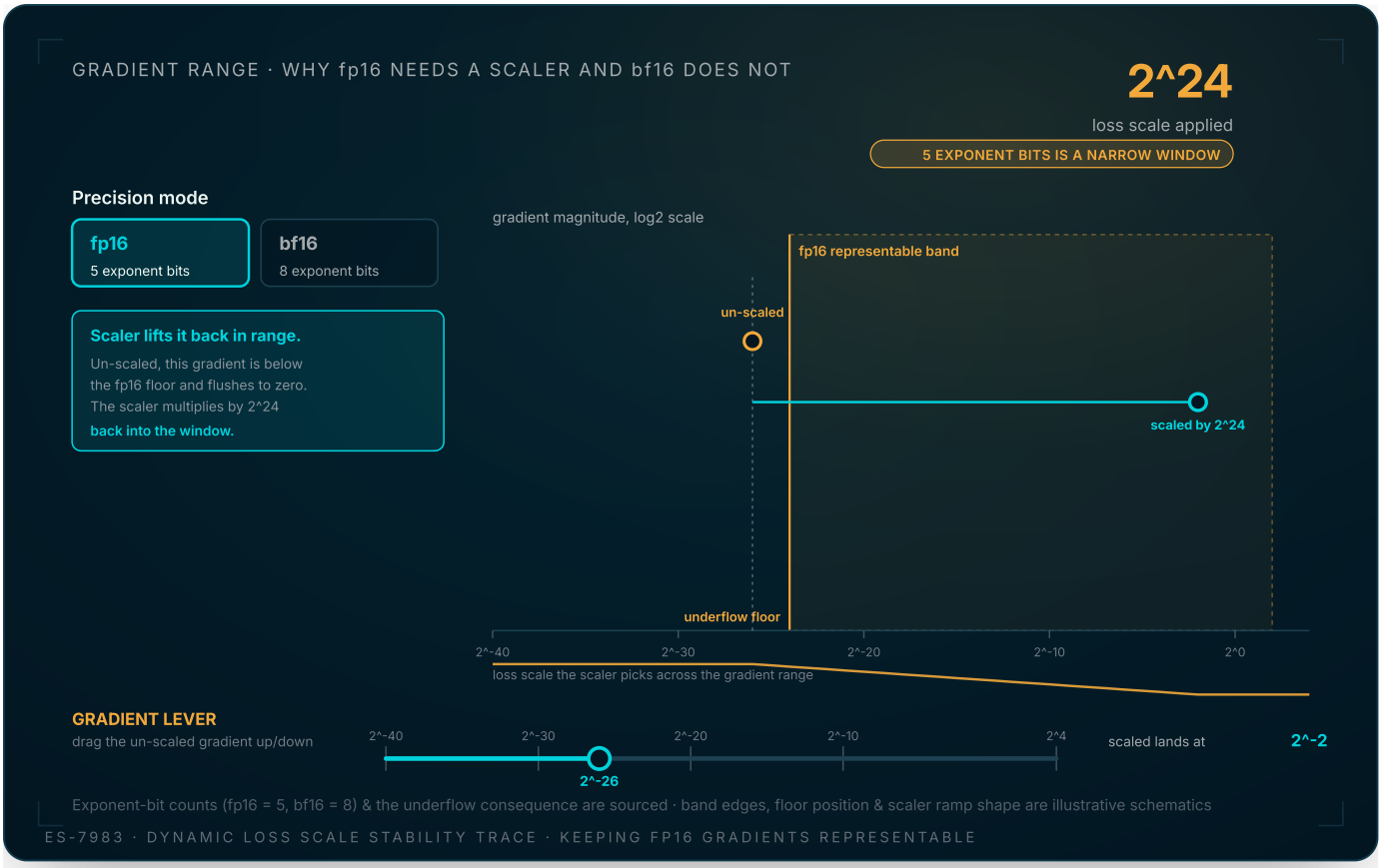
THE MECHANISM

Half precision is two different bargains, and only one needs a scaler

Both 16-bit formats halve the bytes; they do not halve the same way

Once the decision is to store activations in 16 bits, there are two formats to store them in, and they are not interchangeable. The novice reading is that 16 bits is 16 bits, so pick either. The correct reading is that the two formats spend their 16 bits on different things, and the difference decides whether your training loop has to run a loss scaler and whether it will diverge if it does not. The format question is therefore a first-class numerics decision, not a flag you set and forget.

The failure both formats are exposed to is underflow in the backward pass. Gradients, especially deep in a network and late in training, can be very small in magnitude. If a gradient is smaller than the smallest number a format can represent, it is flushed to zero, and a gradient that has become zero by accident stops that parameter learning. Whether this happens depends entirely on the exponent range of the format, which is set by the number of exponent bits, and that is exactly where the two 16-bit formats differ. The second instrument traces this.



A trace of where gradient magnitudes land inside the representable range of each half-precision mode, on a log2 axis. fp16 carries only 5 exponent bits, so its representable band is narrow and small gradients fall below an underflow floor and flush to zero; bf16 carries 8 exponent bits, the same as fp32, so the un-scaled gradient is already inside the band. Drag the gradient lever toward the floor: in fp16 the dynamic loss scaler multiplies the gradient by a power of two to lift it back into the window, while in bf16 the scaler stays flat at one because no rescue is needed. The exponent-bit counts and the qualitative consequence (fp16 underflows, bf16 does not, dynamic scaling rescues fp16) are sourced numerics; the exact band edges, the underflow-floor position, and the scaler-ramp shape are illustrative schematics of the mechanism.

The trace puts gradient magnitude on a log2 axis and draws the representable band of each format on it. In fp16, the band is narrow, because fp16 spends only 5 bits on the exponent, and there is an underflow floor that small gradients fall below. Drag the gradient lever down toward that floor and watch the dynamic loss scaler respond: it multiplies the gradient by a power of two to lift it back into the representable window before the values are stored, then that factor is divided out of the gradients before the optimiser updates the weights, so the update itself is unbiased. In bf16, the band is wide, because bf16 keeps all 8 exponent bits, and the un-scaled gradient is already inside it, so the scaler line stays flat at one. The same lever, the same gradient, two completely different verdicts. The mechanism is worth stating as the equation the loss scaler implements, because it is small and exact.

DYNAMIC LOSS SCALING LIFTS THE GRADIENT, THEN REMOVES THE FACTOR BEFORE THE STEP

Formula | LaTeX

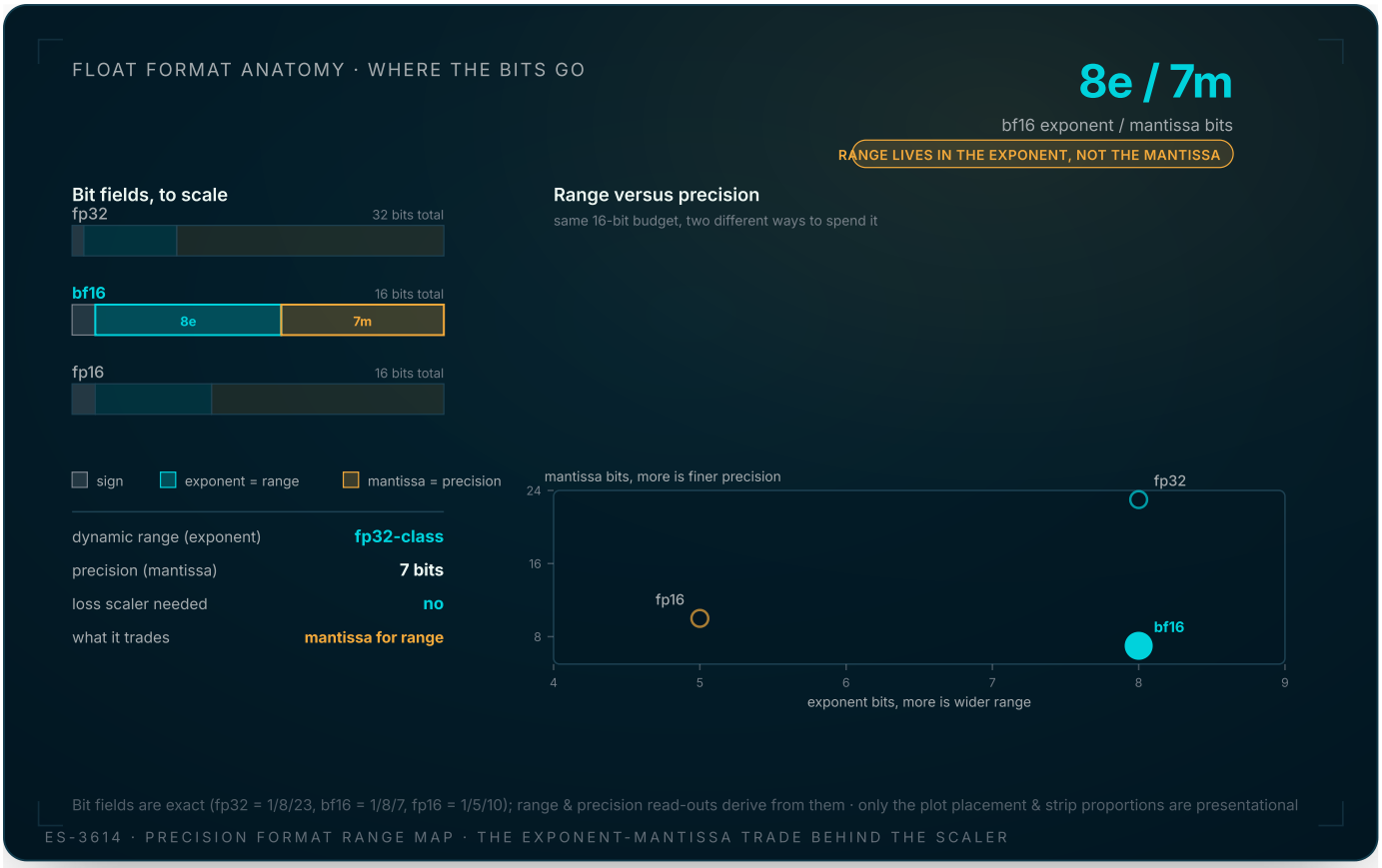
Copy LaTeX

$$g_{\text{stored}} = \nabla_\theta \left(\alpha \cdot \mathcal{L}\right) = \alpha \cdot \nabla_\theta \mathcal{L}, \qquad \theta \leftarrow \theta - \eta \cdot \frac{g_{\text{stored}}}{\alpha}$$

The scaling factor alpha is a power of two so that the multiply and divide are exact shifts of the exponent and introduce no rounding of their own. A dynamic scaler raises alpha after a run of clean steps and halves it the moment a step overflows, skipping that step's update, so the factor tracks the largest value that keeps small gradients representable without pushing the large ones past the overflow edge [1]. That whole apparatus exists only because fp16 has a narrow exponent range. In bf16 there is no apparatus, because there is nothing to rescue [2].

Why the format anatomy decides the bargain

The reason one format needs a scaler and the other does not is not a property you have to take on faith; it is visible in how each format partitions its 16 bits. Dynamic range lives in the exponent, and precision lives in the mantissa, and the three formats in play spend their bits on those two things very differently. The third instrument lays the anatomy out.



The bit-field anatomy of the three float formats the training loop chooses between. The strips split each format into sign, exponent, and mantissa fields to scale: fp32 is 1, 8, and 23 bits; bf16 is 1, 8, and 7; fp16 is 1, 5, and 10. Dynamic range lives in the exponent and precision lives in the mantissa, so the range-versus-precision plot shows the single trade that decides whether a loss scaler is needed. bf16 keeps fp32's 8 exponent bits, inheriting fp32-class dynamic range and needing no scaler, and pays in mantissa precision; fp16 spends 3 of those exponent bits to buy mantissa precision and inherits a narrow range that a dynamic loss scaler then has to manage. The bit-field widths are exact IEEE 754 and bfloat16 definitions and the range and precision read-outs derive directly from them; only the plot placement and the strip proportions are presentational.

Read across the bit-field strips and the trade is plain. fp32 has 8 exponent bits and 23 mantissa bits: wide range and high precision, at 32 bits. bf16 keeps the same 8 exponent bits and cuts the mantissa to 7: it preserves the fp32 dynamic range in half the bits, and it pays for that with coarser precision per value. fp16 keeps a larger 10-bit mantissa but spends only 5 bits on the exponent: more precision than bf16, but a dynamic range so much narrower that small gradients fall off the bottom of it. The range-versus-precision plot places the three formats by their real field widths and shows bf16 and fp32 sharing the wide-range right edge while fp16 sits to the left. That single picture is the whole argument for why bf16 trains like fp32 without a scaler and fp16 does not: the range a gradient needs lives in the exponent, and bf16 kept the exponent.

> **“bf16 did not buy us better numbers than fp16. It bought us a training loop with no loss scaler in it, which is one fewer thing to tune and one fewer way to diverge.”**

— FROM OUR OWN TRAINING NOTES

This is why we treat the format as a deliberate choice rather than a default. On hardware with native bf16 support, bf16 is the lower-risk way to spend the 16 bits: same memory saving, fp32-class range, no scaler to schedule. Where only fp16 is available, the saving is identical but it comes with the obligation to run and tune a dynamic loss scaler, and to watch for the divergence modes a mis-scaled run produces. Either way the headroom is bought; the format decides what else you have to manage to keep it.

IV

THE RESULT

The worst-case log fits, and the curve metrics do not move

What the numerics change actually delivered

The point of the exercise was never a benchmark number; it was to make the widest log in the archive trainable at all under the device ceiling, without paying for it in accuracy. On that test the result is the one mixed precision is designed to give: the activation memory of the binary stage at batch size 1 drops to roughly half once activations are stored in 16 bits, which is what lets the 12,800-pixel worst case fit, and the master-weights-in-fp32 discipline means the reconstructed-curve metrics the digitiser is actually graded on are unchanged. The multiclass stage, which reaches an effective batch of 16 through a custom collate function on the 15,000-instance set, gets the same per-step relief, which is what leaves room for the padded batch in the first place.

THE TRAINING CONFIGURATION THE NUMERICS HAD TO FIT INSIDE

12,800 x 640
Worst-case log width and height in pixels

1 channel
Grayscale input, batch size 1 on the binary stage

128
Embedding dimension at the bottleneck

16
GroupNorm group-size floor at the tiny physical batch

Two configuration choices deserve their own line, because they are the ones that interact with the precision decision rather than sitting beside it.

The first is normalisation. At batch size 1, batch normalisation is a non-starter, because its statistics would be estimated from a single example and would be meaningless. We use group normalisation instead, which normalises over channel groups inside one example and is therefore independent of the batch dimension [5]. We hold the group size at a floor of 16, the half-or-16 rule, so that with the 128-dimensional embedding the groups stay large enough to give stable statistics rather than degenerating toward instance normalisation. Mixed precision does not change this choice, but it makes it more important: the normalisation statistics are among the values we deliberately keep in fp32 inside the autocast region, because computing a mean and variance over half-precision values is exactly the kind of reduction that loses accuracy in 16 bits.

The second is the autocast boundary itself. Autocast is not a global cast of every tensor to 16 bits; it is a context around the forward pass that runs eligible operations, the convolutions and the matrix multiplies, in half precision while leaving numerically sensitive operations, the loss computation and the normalisation reductions, in fp32 [1]. Getting that boundary right is the difference between a stable mixed-precision run and a subtly degraded one, and it is the part that rewards reading the autocast operation list rather than trusting that wrapping the whole step is safe.

Where a half-precision run goes wrong

Half precision is not free of failure modes; it relocates them. The ones we watched for, in the order they tend to bite, are worth naming because they are the cost of the lever.

A loss scale set too high overflows on the large gradients, the dynamic scaler then halves it and skips the step, and if this happens every step the run makes no progress while looking like it is training. A loss scale set too low leaves the small gradients underflowing exactly as if there were no scaler at all, so the symptom is a model that quietly stops improving on the parameters whose gradients are smallest. A reduction left inside the autocast region in 16 bits, a sum or a mean over many values, accumulates rounding error and shows up as a slow accuracy gap against the fp32 baseline that is easy to misattribute to the model. And bf16, for all that it removes the scaler, has the coarsest mantissa of the three formats, so a computation that genuinely needs fine precision, rather than wide range, is the one place bf16 can underperform fp16. None of these is a reason not to use the lever. They are the reason to treat the format choice and the autocast boundary as decisions to get right rather than defaults to accept.

V

THE METHOD

Reproducing the precision posture

Methods, in the detail a practitioner needs

The recipe is small enough to state completely, which is part of why it is the first lever to reach for. Keep a master copy of the model weights in fp32; this is the copy the optimiser updates, so that weight updates too small to be representable in 16 bits still accumulate over many steps. Run the forward pass inside an autocast context so the eligible operations execute in 16 bits and produce 16-bit activations, which is where the memory saving lives, while the loss and the normalisation statistics stay in fp32. On fp16, wrap the backward pass with a dynamic loss scaler that multiplies the loss before backprop and unscales the gradients before the optimiser step, raising the scale after clean steps and halving it on an overflow. On bf16, omit the scaler entirely. Hold normalisation as group normalisation with a group-size floor of 16 so it is valid at batch size 1. Read the activation budget against the widest image in the corpus, because the widest member sizes the step and therefore the whole inequality.

The discipline that makes this trustworthy rather than hopeful is to validate that the deliverable metrics, the reconstructed-curve agreement against ground truth, are unchanged between the fp32 baseline and the mixed-precision run before adopting it. Mixed precision should buy memory and speed and cost nothing in accuracy; if a curve metric moves, the autocast boundary or the loss scale is wrong, and that regression is the signal to fix it rather than to ship.

A roadmap once precision is spent

Halving the byte width is the first lever, not the last. Once activations are in 16 bits and the worst-case log fits, the next increments of headroom come from levers that compose with precision rather than replace it. Gradient checkpointing recomputes activations in the

backward pass instead of storing them, trading step time for a further drop in the activation term, and it is the natural second move when an even wider scan or a deeper model pushes back against the ceiling [7]. Tiling the widest images along the depth axis, processing a tall image in overlapping vertical strips, attacks the activation count directly rather than its byte width, and is the move when no precision or recomputation trick is enough. The order is deliberate: spend the cheap, accuracy-neutral lever first, then the compute-for-memory lever, then the ones that touch the data layout. Precision is where the budget starts because it is the only one of the three that costs nothing in accuracy and gives back speed while it gives back memory.

Activations in 16 bits

- The dominant per-step memory term, stored at 2 bytes per value instead of 4
- Entered through autocast around the forward pass, not a manual cast of every tensor
- This is the line item that actually frees the headroom the 12,800-pixel log needs
- Halving it is what lets the worst-case width fit at batch size 1

Master weights in fp32

- The optimiser keeps a full-precision master copy of the weights
- Tiny weight updates that would vanish in 16 bits still accumulate correctly
- Accuracy does not move: the deliverable curve metrics are unchanged
- The half-precision copy is used only for the forward and backward maths

Loss scaling, format-dependent

- fp16: a dynamic loss scaler multiplies the loss by a power of two before backprop
- The scaler backs off on an overflowing step and ramps up after a clean run
- bf16: no scaler at all, its 8 exponent bits already cover the fp32 range
- Choosing the format is choosing whether you run a scaler

WHAT TO CARRY OUT OF THIS

1. At batch size **1** the activation byte width is the primary memory lever, because there is no batch left to shrink. Storing activations in 16 bits halves the dominant term, at **2** bytes per value against **4**, which is what lets the **12,800**-pixel worst-case log fit.
2. Keep the master weights and the loss in **fp32** inside an autocast context. The 16-bit copy does the forward and backward maths; the fp32 copy takes the update, so tiny weight changes still accumulate and the curve metrics do not move.
3. The two 16-bit formats are not interchangeable. **bf16** keeps fp32 dynamic range with **8** exponent bits and needs no loss scaler; **fp16** has only **5** exponent bits, underflows small gradients, and must run a dynamic loss scaler.
4. Loss scaling multiplies the loss by a power of two before backprop and divides it out before the step, so the update is unbiased while small gradients stay representable. A dynamic scaler ramps up after clean steps and halves on overflow.
5. Hold normalisation as group normalisation with a group-size floor of **16** so it is valid at the tiny physical batch, and keep the normalisation reductions in fp32 inside autocast.

Limitations

The absolute memory figures in the budget gauge are illustrative schematics of the activation term, not measured allocations from a profiler; the byte widths, the geometry, the batch size, the embedding dimension, and the GroupNorm floor that drive them are the sourced quantities, and the gauge is built to argue the proportional effect of halving the byte width rather than to predict a device-specific gigabyte total. The representable-range band edges, the underflow floor, and the loss-scaler ramp shape in the stability trace are schematics of the mechanism on a log2 axis; the exponent-bit counts that motivate them and the qualitative underflow behaviour are real, but the exact positions are presentational. The format-anatomy map uses exact IEEE 754 and bfloat16 field widths, and the range and precision read-outs derive from them, but the plot placement is presentational. The accuracy-neutrality claim is specific to this network and this deliverable metric, the reconstructed-curve agreement against ground truth, and to keeping the loss, the master weights, and the normalisation statistics in fp32; a different architecture, a different metric, or a careless autocast boundary can move accuracy, and the only safe posture is to validate the deliverable metric between the fp32 and mixed-precision runs before adopting the change. Finally, the relative merits of the two formats depend on the hardware: bf16 is the lower-risk choice only where it is natively supported, and on devices without it the fp16-plus-scaler path is the one that buys the same memory.

References

1 Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H. (2017). *Mixed Precision Training.* arXiv (ICLR 2018). The reference recipe: fp16 storage and compute, an fp32 master copy of the weights, and dynamic loss scaling to keep small gradients representable. https://arxiv.org/abs/1710.03740

2 Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., et al. (2019). *A Study of BFLOAT16 for Deep Learning Training.* arXiv. The format that keeps the fp32 exponent range in 16 bits, so training matches fp32 without a loss scaler. https://arxiv.org/abs/1905.12322

3 He, K., Zhang, X., Ren, S., Sun, J. (2016). *Deep Residual Learning for Image Recognition.* CVPR. The residual-block encoder whose activation footprint is the dominant per-step memory term. https://arxiv.org/abs/1512.03385

4 Ronneberger, O., Fischer, P., Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* MICCAI. The encoder-decoder shape whose skip connections hold

high-resolution activations alive through the entire decode path. https://arxiv.org/abs/1505.04597

5 Wu, Y., He, K. (2018). *Group Normalization.* ECCV. The normalisation scheme that stays valid at the small physical batch a memory ceiling forces. https://arxiv.org/abs/1803.08494

6 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017). *Attention Is All You Need.* NeurIPS. The self-attention refinement on the bottleneck whose activation memory scales with sequence length. https://arxiv.org/abs/1706.03762

7 Chen, T., Xu, B., Zhang, C., Guestrin, C. (2016). *Training Deep Nets with Sublinear Memory Cost.* arXiv. Gradient checkpointing, the complementary lever that trades recomputation for activation memory once precision has been spent. https://arxiv.org/abs/1604.06174

8 Koroteev, D., Tekic, Z. (2021). *Artificial intelligence in oil and gas upstream: Trends, challenges, and scenarios for the future.* Energy and AI. Context for why digitising a legacy raster archive is worth training under a constrained memory budget at all. https://www.sciencedirect.com/science/article/pii/S2666546820300033

Get the full whitepaper

This page is the long-form summary. The complete whitepaper adds the per-stage activation profile at the 12,800-pixel worst case under fp32 and half precision, the autocast operation list and the exact fp32 fallbacks we kept, the dynamic-loss-scaler schedule and the overflow-skip logic, and the worked accuracy-parity check between the fp32 baseline and the mixed-precision run on the held-out curves.

EARTHSCAN

Half the Bits, Twice the Headroom: Mixed-Precision Training for 12800-Pixel Log Images

Published December 2024. Generated 2026-07-22.
https://www.earthscan.io/whitepapers/mixed-precision-training-under-a-hard-gpu-memory-ceiling