Training Under a Hard GPU Memory Ceiling

Conventional training-cost planning treats compute as an arithmetic problem: count the floating-point operations, divide by device throughput, multiply by a price. That arithmetic quietly assumes a fixed-shape batch tensor, and for a raster well-log archive that assumption never holds. The images span roughly 3,200 to 12,800 pixels wide against a narrow 480 to 640 pixel height, in no fixed aspect ratio, so they cannot be stacked, and the first cost that bites is not arithmetic at all; it is the GPU memory committed by the single widest member of a mini-batch. This whitepaper is the first-person engineering and financial account of training VeerNet, our encoder-decoder segmentation network with a transformer attention bottleneck, under exactly that ceiling. The binary segmentation stage was forced to batch size 1, which trained correctly and finished 50 epochs over 2,000 instances in 110 minutes, but starved the optimiser of gradient averaging. The three-class multiclass stage on 15,000 synthetic instances reached an effective batch of 16 through a custom collate function that pads each mini-batch to its widest member on a 32-pixel alignment grid and a validity mask that keeps padded pixels out of the loss; it finished 50 epochs in 550 minutes. We set out the memory inequality that governs the whole problem, the alignment-padding cost that the encoder stride structure imposes, and the two rentable GPU tiers, 750 and 1,800 EUR per month, that the engagement budgeted against. We then build the spend plan an executive can defend in a budget review: how to scope rented hours to a batch-size-1 wall clock, why the widest log, not the average, sizes every line item, and which engineering levers move the bill versus which only move the FLOPs. The deliverable is a method for budgeting compute when the binding constraint is memory, and a worked ledger that an upstream operator or any team training on variable-dimension imagery can reuse.

Tannistha Maiti
Senior AI Researcher

Narendra Patwardhan
Research Collaborator

VEERNET / GPU-MEMORY / COMPUTE-BUDGETING / BATCH-SIZE-1 / COLLATE-FN / ALIGNMENT-PADDING

CONTENTS

There is a number every compute budget begins with, and for most teams it is the wrong number. The reflex is to estimate floating-point operations: count the multiply-adds in a forward and backward pass, multiply by the dataset size and the epoch count, divide by the device's advertised throughput, and read off the hours. It is a clean estimate, it survives a spreadsheet, and on a fixed-resolution benchmark it is roughly right. It was wrong for us by the time we had loaded the first mini-batch, because the dataset we were budgeting against does not have a fixed resolution, and the cost that bit first was not arithmetic. It was memory.

We built VeerNet to digitise a raster archive of scanned well logs: an encoder-decoder segmentation network, residual blocks in the encoder [1], an upsampling decoder in the U-Net lineage [2], and a two-layer transformer attention refinement on the bottleneck [3]. The geoscience that makes this worth doing is upstream of this document; the architecture is its own subject. What this whitepaper is about is the part nobody put on the original plan: that a hard GPU-memory ceiling, not a FLOP count, governed every training decision and every line of the spend, and that budgeting it correctly meant inverting the usual cost model. This is the first-person account of how we did that, what it cost, and how to write a defensible GPU budget when memory is the binding constraint.

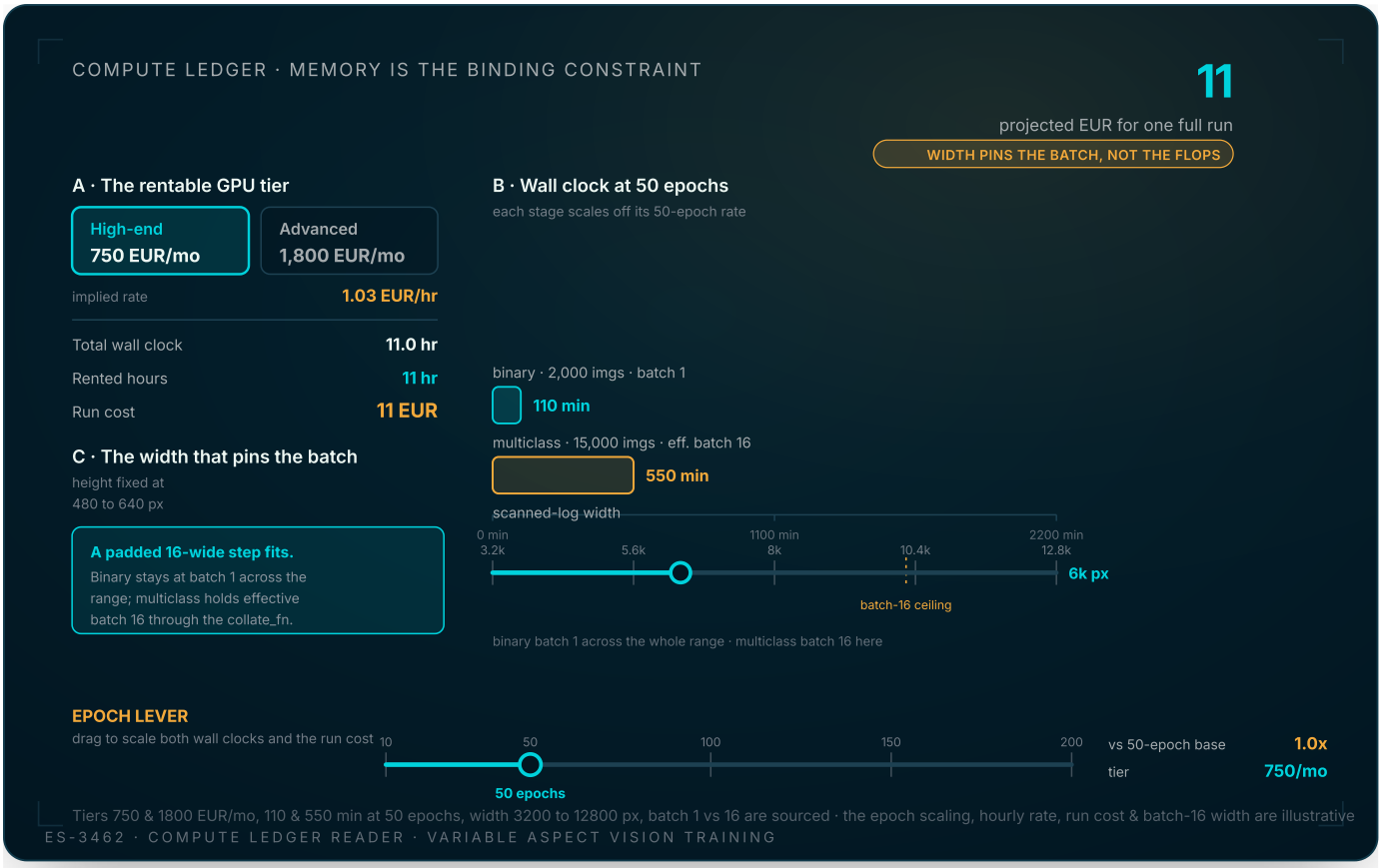
Why this is a memory problem, not a FLOPs problem

A deep-learning training step has a memory footprint with two kinds of terms, and the distinction is the entire planning lever. The fixed terms are the ones that do not change with the input: the model parameters, the gradient buffers, the optimiser state. For a network the size of VeerNet those are real but bounded, and you pay them once regardless of what you feed in [7]. The per-batch term is the activation footprint: every intermediate tensor the forward pass produces and the backward pass has to keep around to compute gradients. That term scales with the size of the input, and for an encoder-decoder with skip connections it is large, because the high-resolution activations from early encoder stages are held alive through the entire decode path to be concatenated back in [2].

For a fixed-resolution benchmark this is a non-issue, because the activation footprint is a constant you measure once. Scanned well logs make it the dominant, variable cost. The raster archive runs from roughly **3,200 to 12,800 pixels wide** and **480 to 640 pixels tall**, in no fixed aspect ratio, because the physical logs were printed at different vertical scales by different service companies across decades, then photographed and scanned by different

operators on different equipment. A 12,800-pixel-wide log is four times the width of a 3,200-pixel one, and its activation footprint through the network is correspondingly larger. The widest image you might sample sets the peak memory of the step, and you have to provision for that peak, not the average, because a batch sampler can pull the widest log at any moment.

That single fact reorders the cost model. The binding constraint is not how many operations the GPU can do per second; it is how many of those activations the GPU can hold at once. When the binding constraint is memory, FLOP-based budgeting answers the wrong question. You can have spare arithmetic throughput and still be unable to fit a second image in the batch, which means the right cost driver is the memory committed by the widest member of any mini-batch, and the right first question for the budget is not "how fast is the card" but "how wide is the widest log, and what does holding it cost". The reader below walks the three coupled exhibits this produces: the GPU tier you rent, the wall clock each training stage actually took, and the input width that decides whether the batch can hold more than one image.



One reader over the compute ledger for variable aspect ratio vision training, where GPU memory, not arithmetic, is the binding constraint. Exhibit A picks a rentable GPU tier, 750 or 1800 EUR per month. Exhibit B shows the training clock: binary segmentation on 2,000 instances ran 50 epochs in 110 minutes, the three-class multiclass set of 15,000 instances ran 50 epochs in 550 minutes, and the epoch lever scales both linearly. Exhibit C is the width ceiling: scanned logs run from 3,200 to 12,800 px wide at a fixed 480 to 640 px height, and that wide variable width is what pins the binary pipeline to batch size 1, with the multiclass collate_fn holding an effective batch of 16 only until the widest members make a padded 16-wide step unaffordable. The tier prices, the 110 and 550 minute endpoints, the width range, and the batch counts are sourced from the engagement archive; the linear epoch scaling, the implied hourly rate, the projected run cost, and the exact width at which a 16-wide step stops fitting are illustrative.

The binary stage: a forced batch of one, and what it cost

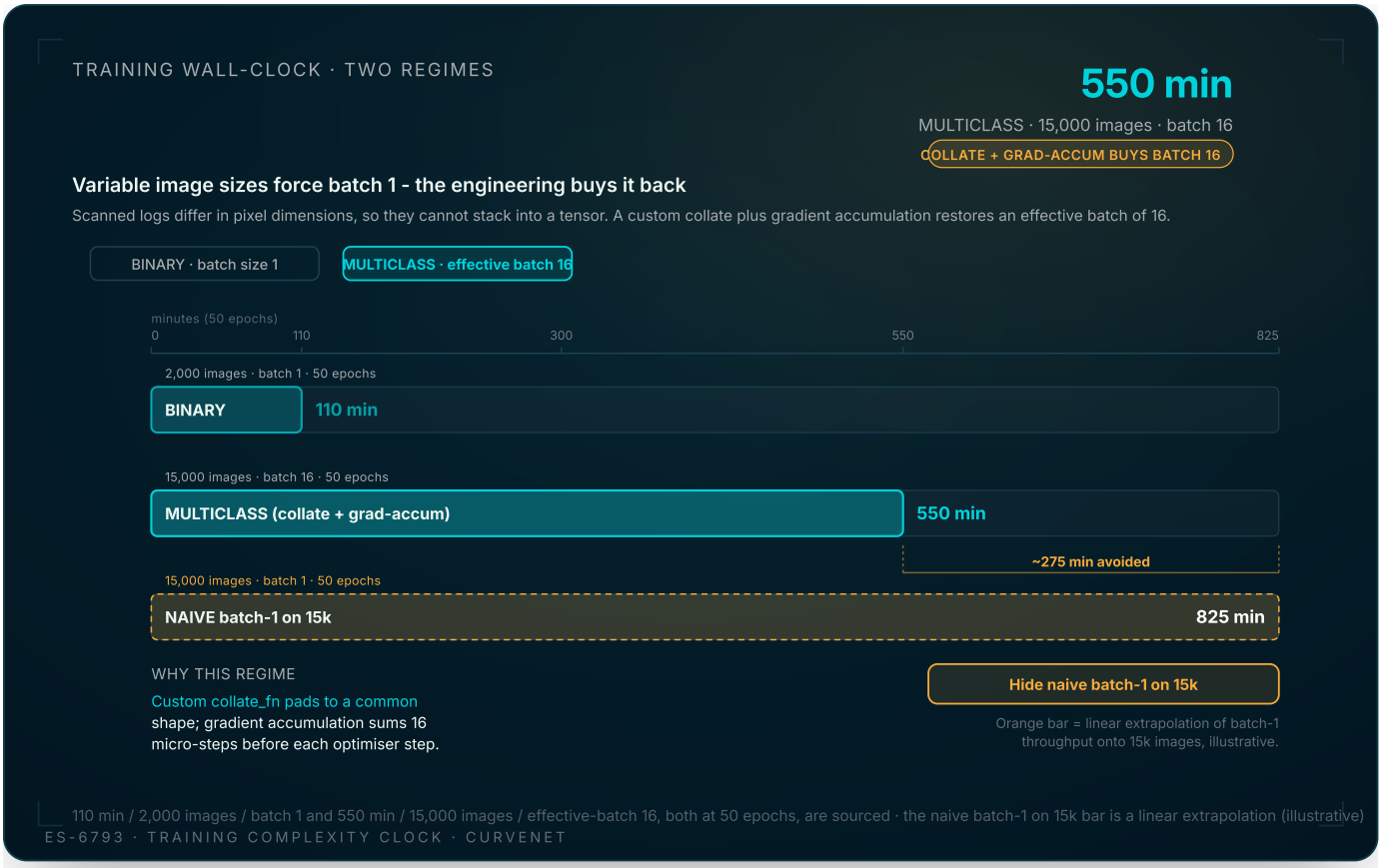
The first stage was binary segmentation, foreground curve trace against background, on an initial dataset of **2,000 instances**. We took the blunt, honest answer to the variable-dimension problem: train at batch size **1**.

A batch of one is the single setting that makes variable dimensions disappear, because one image never needs two shapes to agree. There is nothing to stack, so nothing breaks; you feed one `(1, 1, H, W)` tensor through the network, the grayscale log being a single input channel, compute the loss against one mask, backpropagate, and step. The data path is trivial. Everything you surrender, you surrender at the optimiser. A mini-batch of `N`

produces a gradient that is the mean of `N` per-example gradients, and that averaging is most of why mini-batch training is stable; at batch size 1 every step is driven by one log, the gradient is as noisy as the noisiest image in the set, and the optimiser jitters. We compensated with a lower learning rate and more epochs, which is to say we paid for the memory ceiling in wall-clock time.

Normalisation was the second casualty, and it is worth naming because it is a correctness problem rather than an inefficiency. Batch Normalization estimates per-channel statistics across the batch dimension; with a single image the "batch statistic" is computed over one example, the variance estimate is degenerate, and the normalisation injects noise instead of removing it. Our answer was Group Normalization [4], which computes its statistics across groups of channels inside one image and never touches the batch dimension, so its behaviour is identical whether the batch is 1 or 64. We size the groups with a `half_or_16` rule, taking the larger of half the channel count and a floor of 16 channels per group. With GroupNorm in place, batch size 1 trains correctly; it trains slowly, but the gradient it computes is honest.

The number from that stage is the anchor for everything financial that follows: **110 minutes for 50 epochs** over the 2,000 binary instances. That is a clean, reproducible run, and it is the unit a budget can be built on. It is also a ceiling. Batch size 1 was never going to carry the harder problem, and the wall clock it produced is the slow baseline every later optimisation is measured against.



Scanned well-log images arrive at wildly different pixel dimensions, so they cannot be stacked into a tensor; the naive fix is batch size 1, which under-feeds the GPU and stretches the wall-clock. CurveNet's answer is a custom collate_fn plus gradient accumulation, which buys back an effective batch of 16 without the memory blowup. Pick a regime; toggle the naive batch-1 on 15k bar to see the wall-clock the fix avoided. The 110 min / 2,000 images / batch 1 and 550 min / 15,000 images / effective-batch 16 figures (both at 50 epochs) are the handover's own; the orange naive bar is a linear extrapolation of the batch-1 throughput onto 15k images and is flagged as illustrative.

The multiclass stage: buying back a batch without buying more memory

The multiclass stage is where the constraint stopped being tolerable. The task expanded to **three classes**, background plus two curves, on a synthetic dataset of **15,000 instances** spanning the full dimensional range of the real archive, 3,200 to 12,800 pixels wide. The dataset was larger, harder, and dimensionally wider, and the sparse-foreground imbalance was worse: two thin curves against a vast background means the loss is dominated by background pixels unless the batch is large enough to average over the foreground. At batch size 1 the noisy single-example gradient and the foreground imbalance compound each other. We needed a real batch, and the memory ceiling said we could not have one the obvious way.

You cannot stack a 3,200-wide log and a 12,800-wide log into one tensor, and resizing is off the table because the supervisory signal lives in curve traces that are often one pixel wide; an interpolation kernel smears a one-pixel feature into the background and trains the network on the resize artefact instead of the curve. So we wrote a custom PyTorch collate function. It inspects each sampled mini-batch, finds the maximum height and width across its members, pads every image and mask up to that per-batch maximum, and builds a per-pixel validity mask so that **every padded pixel contributes exactly zero gradient**. The pad makes `torch.stack` legal; the mask keeps the fiction out of the loss. Crucially the padding is not to an arbitrary box: the encoder has five stride-2 stages, a 32-times downsample, so the padded width is rounded up to the nearest 32-pixel boundary or the upsample path does not line back up with the skip connections. That alignment padding is a real, if small, memory cost, and it is one of the few costs that is structural rather than incidental.

That gave us a legal batch of more than 1; gradient accumulation gave us an effective batch of 16 without the memory bill. Instead of forwarding 16 images at once, we forward the small number that fits in memory, call backward to accumulate gradients into the parameter buffers without stepping, repeat until 16 images' worth of gradient has piled up, then take one optimiser step. The optimiser updates on the averaged gradient of 16 examples while the device never holds more than the small physical batch that fits. This is the central move of the whole budget: the effective batch is a software variable the optimiser wants, and the physical batch is a hardware variable the memory ceiling pins, and gradient accumulation decouples them. You buy the gradient quality of a batch of 16 at the memory price of a batch the card can actually hold.

The number from that stage: **550 minutes for 50 epochs** over the 15,000 multiclass instances. It is longer than the binary run by exactly the factors you would predict, more than seven times the data plus the accumulation overhead, and it is a run that was simply not reachable at batch size 1 on a set this large and this wide. Those two wall clocks, 110 and 550 minutes, are the load-bearing inputs to the spend plan.

The memory inequality the budget is built on

Strip the pipeline to the constraint that governs it and the whole thing reduces to one inequality. The peak memory of a training step is approximately the activation footprint of the network evaluated at the largest padded image in the mini-batch, multiplied by the

physical batch size, plus the fixed parameter, gradient, and optimiser-state buffers, plus the backpropagation workspace. Every term except the first is fixed. The first term is set by your widest log times how many of them you try to hold at once.

That observation produces the rules a budget for variable-dimension data has to obey:

- **The widest image, not the average, sizes every line item.** Planning around the mean log width is the classic and expensive mistake. A sampler can pull a 12,800-wide log at any moment, the physical batch has to survive that worst case, and so the physical batch is small. Provision the GPU tier and the hours against the peak, never the mean.
- **Effective batch is free; physical batch is bought.** Gradient accumulation makes the effective batch a parameter you tune for optimisation quality at no memory cost, while the physical batch stays pinned to the ceiling. Decoupling them is what lets a small card train as if it had a large one, and it is the cheapest lever in the budget because it is software.
- **Alignment padding is a structural memory cost.** The 32-pixel rounding from the five stride-2 stages is not optional and not free; it inflates the padded width and therefore the activation footprint. It is small, but it is a real line, and it is the one padding cost you cannot sample your way out of.
- **GroupNorm is what makes the small physical batch harmless.** Because the widest-log ceiling forces a tiny physical batch, a normalisation that degrades at small batch would quietly poison the run. GroupNorm's batch independence [4] is the precondition that lets the memory ceiling be a budget constraint rather than a correctness one.

Hold the whole thing in one sentence: the binary stage paid for the ceiling in optimiser quality, accepting a noisy batch-of-one gradient to keep memory trivial, and the multiclass stage paid for it in engineering and wall clock instead, a collate function and accumulation that bought back a real averaged gradient at a memory price the card could pay.

From wall clock to a defensible GPU spend plan

A budget review does not want a FLOP count; it wants rented hours and a number in euros, with the assumptions visible. The engagement budgeted two GPU tiers: a high-end tier at **750 EUR per month** and an advanced tier at **1,800 EUR per month**. The spend plan is built by composing the two sourced wall clocks with a tier choice and the epoch count, and the reader at the top of this paper is exactly that composition made interactive.

The honest construction is straightforward. The two anchor runs are 110 minutes for the binary stage and 550 minutes for the multiclass stage, both at 50 epochs, which is 11 hours of training time for a single full pass through both stages at the standard epoch budget. A monthly GPU tier converts to an effective hourly rate once you decide a utilisation assumption; against a fully rented month the high-end tier is on the order of a euro an hour and the advanced tier a little over twice that, so a single full run lands in the low double-digit to low triple-digit euros depending on tier and how many times you retrain. The exact arithmetic is illustrative and flagged as such in the reader, but the structure is the point: the bill is dominated by wall clock, the wall clock is set by the batch-size-1 regime the memory ceiling forces, and the tier choice scales a number you have already pinned. The budget you defend is not "we will buy X teraflops"; it is "two anchored runs, scaled by the epoch and retrain count we expect, on the tier whose memory fits our widest log".

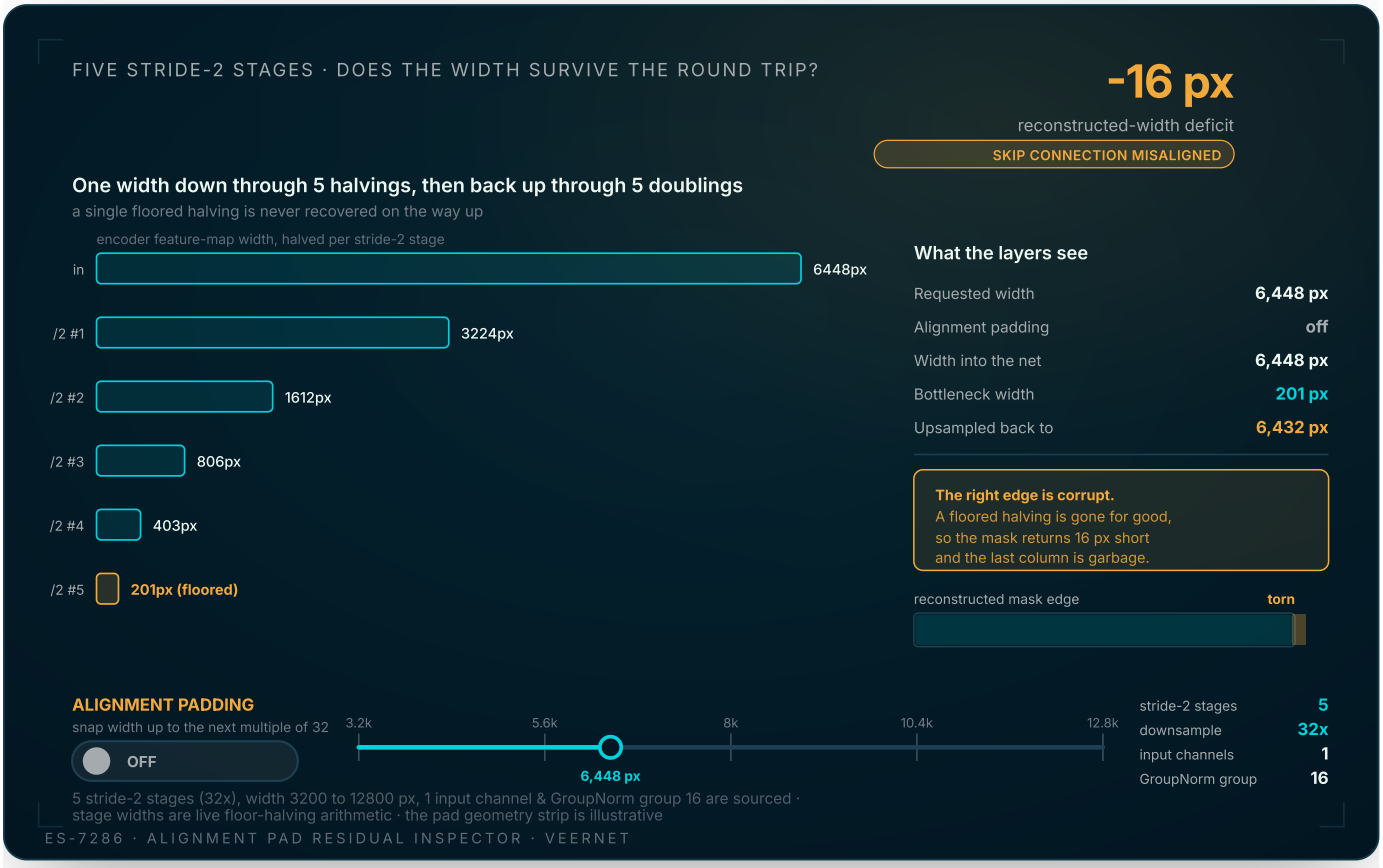
The corollary is the part that surprises a FLOP-trained reviewer: paying for a faster card does not necessarily move this budget. A more expensive tier that adds arithmetic throughput but not memory headroom does not let you increase the physical batch, so it does not shorten the batch-size-1 wall clock that dominates the bill. The tier that helps is the one with more memory, because that is the one that lets the physical batch grow and the accumulation steps shrink. Budgeting by memory rather than FLOPs is what tells you which upgrade is worth paying for and which is wasted.

Methods, in the detail a planner needs to reproduce this

The numbers in this paper come from real training runs, and reproducing the budget means reproducing the conditions that set the wall clocks. The binary stage trained on 2,000 instances at a physical and effective batch of 1, with GroupNorm throughout and the `half_or_16` group-sizing rule, for 50 epochs in 110 minutes. The multiclass stage trained on 15,000 synthetic instances spanning the full 3,200 to 12,800 pixel width range and 480 to 640 pixel height range, three classes, with the custom collate function padding each mini-batch to its widest 32-aligned member, a validity mask zeroing padded pixels in the loss, and gradient accumulation to an effective batch of 16, for 50 epochs in 550 minutes.

The alignment padding deserves its own look, because it is the one memory cost that is dictated by the architecture rather than the data and the only one a planner cannot sample away. The inspector below traces a single input width through the five stride-2 halvings and back up the decode path, with and without the 32-pixel snap, so you can see exactly where

an unaligned width lands the reconstructed mask one or two pixels short of the encoder feature it must concatenate with, and how snapping the width up to the next multiple of 32 makes every stage divide cleanly at the cost of a little more width to hold in memory.



A symmetric encoder-decoder halves the feature-map width once per stride-2 stage and doubles it back up; with 5 stages it downsamples by 32, so an input width that is not a multiple of 32 cannot survive the round trip. Drag the width ruler to choose any input width, then toggle alignment padding. With padding off, watch one of the five integer halvings floor away a fractional pixel that the decoder can never recover, so the reconstructed mask comes back short and its right edge is corrupted. With padding on, the width is first snapped up to the next multiple of 32, every stage divides cleanly, and the round trip is exact. The 5 stride-2 stages, the 32x downsample, the 3200 to 12800 px width range, the single grayscale input channel, and the GroupNorm group size of 16 are sourced from the engagement archive; the stage widths are live floor-halving arithmetic, and the pad-geometry strip is an illustrative schematic.

Three implementation details carry more budget weight than their line count suggests. First, the per-batch padding target is computed from the sampled members, not fixed for the dataset, so a mini-batch of narrow logs pads to a small box and only a batch that catches a wide log pays the wide-box memory; this makes the sampler a budget lever, because bucketed, length-aware sampling that groups similarly-sized logs cuts the masked-pad waste that mixed-width batches incur. Second, the validity mask is multiplied into the loss before any reduction, so the gradient the optimiser sees is computed only over real pixels regardless of how much padding the batch carries; without it the network learns that vast

rectangles of background are the right answer and the run is quietly poisoned. Third, the 32-pixel alignment is dictated by the five stride-2 encoder stages, and it is the one padding cost that is structural; you cannot sample it away, only relax it by changing the downsample factor, which is an architecture decision, not a budget one.

For a planner who wants to relax the ceiling rather than budget around it, the two levers are well established and both trade compute for memory. Mixed-precision training [6] stores activations in half precision and roughly halves the activation footprint, which directly raises the physical batch the memory can hold and shortens the accumulation schedule. Gradient checkpointing [5] discards intermediate activations on the forward pass and recomputes them on the backward, trading extra arithmetic for a lower peak memory, which is precisely the trade you want when arithmetic throughput is the thing you have spare. Neither was needed to ship the runs in this paper, and both are the first places we would spend engineering time if the memory ceiling, rather than the schedule, became the thing we wanted to move.

A worked sensitivity: what actually moves the bill

A budget is only useful if it tells you where to spend the next euro, so it is worth walking the sensitivities one lever at a time, holding the two anchored wall clocks fixed and asking what each change does to the bill.

Start with the epoch count, because it is the lever a reviewer reaches for first and the one whose effect is most nearly linear. The 110-minute binary run and the 550-minute multiclass run are both 50-epoch figures, so doubling the epoch budget roughly doubles the wall clock and therefore the rented hours and the cost, on either tier. The epoch lever in the reader makes this concrete: slide it and both bars and the headline cost move together off their per-epoch rate. The honest caveat is that this linearity is a planning approximation, not a physical law; in practice early epochs do more useful work than late ones, and a real schedule would use early stopping to truncate the multiclass run once the validation curve flattens. For a budget, linear-in-epochs is the conservative assumption, and conservative is what a budget should be.

The retrain count is the lever a FLOP-trained reviewer most often forgets, and it is multiplicative with everything else. A model is not trained once; it is trained, evaluated, adjusted, and trained again, and across a delivery the number of full runs is frequently the

largest single multiplier on the bill. Two anchored runs at 50 epochs is the unit cost; a programme that retrains a dozen times pays that unit a dozen times. This is why the cheapest real saving is usually not a faster card but a tighter experiment loop: fewer, better-chosen retrains move the bill more than any hardware decision, because they attack the multiplier rather than the unit.

The tier choice is the lever whose effect is the most counterintuitive, and it is the one this whole paper is built to clarify. Moving from the 750 to the 1,800 EUR tier roughly multiplies the implied hourly rate, so if nothing else changed the bill would scale with the tier price. But something else can change: a tier with more memory raises the physical batch the card can hold, which shortens the accumulation schedule and therefore the wall clock, which pulls the bill back down. The two effects fight each other, and which one wins depends entirely on whether the more expensive tier buys memory or only arithmetic. A tier that doubles the price and the FLOPs but not the memory makes the bill worse, because it cannot shorten the batch-size-1 wall clock that dominates it. A tier that doubles the price and the memory can make the bill better, because it lets the physical batch grow. The budget tells you which is which only because it is denominated in memory rather than FLOPs.

Finally the width distribution is the lever nobody puts on a budget and everybody should. Because the widest member of a mini-batch sizes the memory, a dataset skewed toward wide logs is more expensive to train per epoch than one skewed toward narrow ones, even at identical instance counts. Bucketed sampling that keeps wide logs together and narrow logs together cuts the masked-pad waste and recovers throughput, which is a software change that moves the wall clock and therefore the bill. It is the rare lever that is free to pull and meaningful in effect, and it exists only because the cost runs through the widest image. A planner who has internalised the memory inequality will price the width distribution of the archive before pricing the card.

What changes once memory is the unit of account

The whole exercise was worth doing because a legacy raster archive is a large, stranded asset, and putting an AI capability against it only pays if the training is affordable and the affordability is honest [8]. Budgeting this kind of work by FLOPs would have produced a confident number that the first mini-batch falsified. Budgeting it by memory produced a

plan that survived contact with the hardware: two anchored wall clocks, a tier chosen for its memory rather than its arithmetic, an effective batch decoupled from the physical one, and a euro figure a reviewer can interrogate line by line.

The reusable result is the inversion itself. When images arrive at a fixed shape, FLOPs are a fine proxy for cost and the usual budget arithmetic holds. When they arrive at every shape, the widest member of a mini-batch sizes the memory, the memory sizes the physical batch, the physical batch sizes the wall clock, and the wall clock sizes the bill, so the chain of cost runs through memory from end to end and FLOPs drop out of the first link entirely. A team that internalises that chain stops asking how fast the card is and starts asking how wide the data is, and it stops paying for throughput it cannot use on a card whose memory was the thing standing in the way. That reordering, more than any single trick in the collate function, is what turned a pile of differently-shaped images into a model on a budget we could sign.

References

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

2. 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 in memory through the decode. https://arxiv.org/abs/1505.04597

3. Vaswani, A., et al. (2017). *Attention Is All You Need.* NeurIPS. The self-attention bottleneck whose memory scales with sequence length, which for a wide log is large. https://arxiv.org/abs/1706.03762

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

5. Chen, T., Xu, B., Zhang, C., Guestrin, C. (2016). *Training Deep Nets with Sublinear Memory Cost.* arXiv. Gradient checkpointing, the canonical lever for trading compute to relax the activation-memory ceiling. https://arxiv.org/abs/1604.06174

6. Micikevicius, P., et al. (2018). *Mixed Precision Training.* ICLR. Halving the activation memory bill by storing activations in half precision. https://arxiv.org/abs/1710.03740

7. Rajbhandari, S., Rajbhandari, S., Ruwase, O., He, Y. (2020). *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models.* SC20. The partitioning view of where training memory goes, useful for separating fixed from per-batch terms. https://arxiv.org/abs/1910.02054

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 a constrained compute 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 includes the per-stage memory profile at the 12,800-pixel worst case, the collate-function and validity-mask reference implementation, the full tier-by-tier spend table across epoch and retrain counts, and the mixed-precision and checkpointing sensitivity analysis for relaxing the ceiling.

EARTHSCAN

Compute Budgeting for Vision Training Under a Hard GPU-Memory Ceiling

Published June 2025. Generated 2026-07-22.
https://www.earthscan.io/whitepapers/compute-budgeting-constrained-gpu-memory-whitepaper