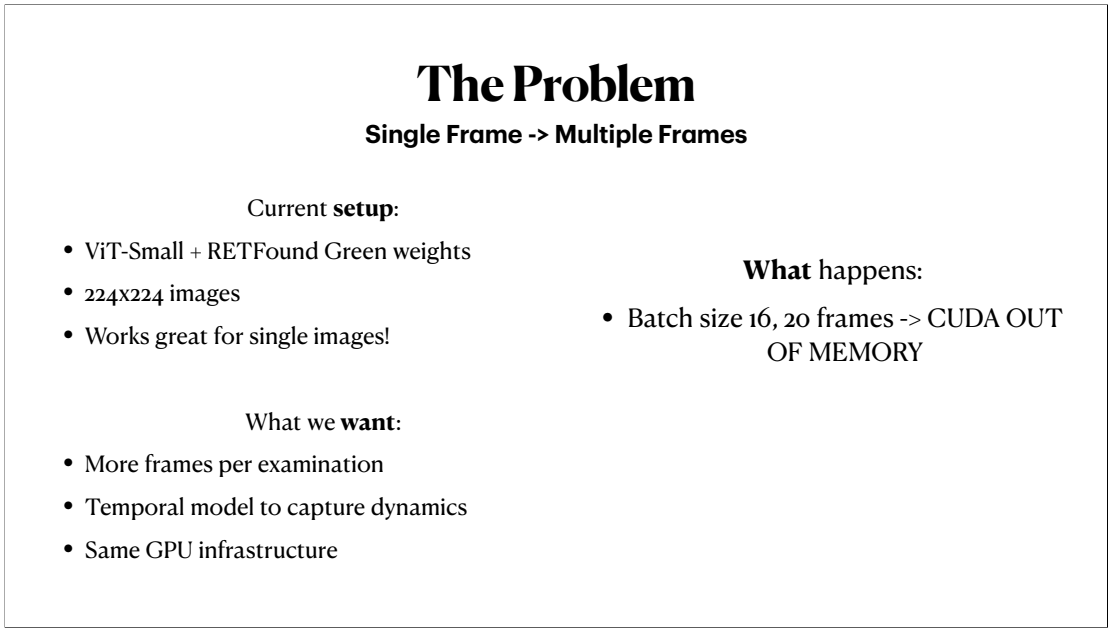
Making MedNet Ready for Temporal Data

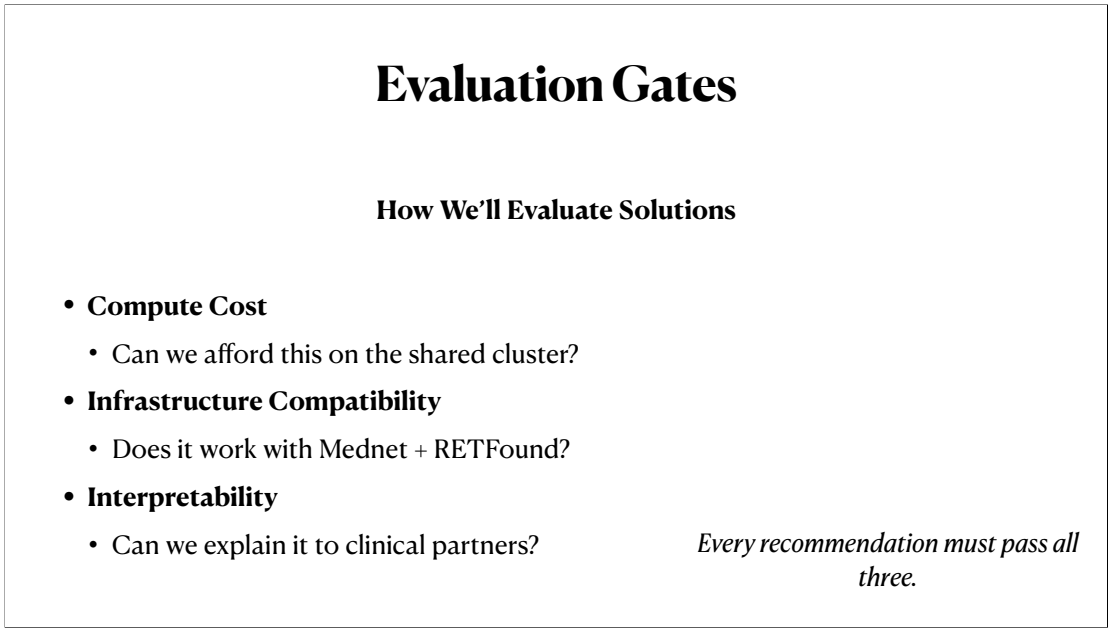
Performance Analysis & Recommendations for the MedAI Team

Tymo van Rijn | Idiap Research Institute

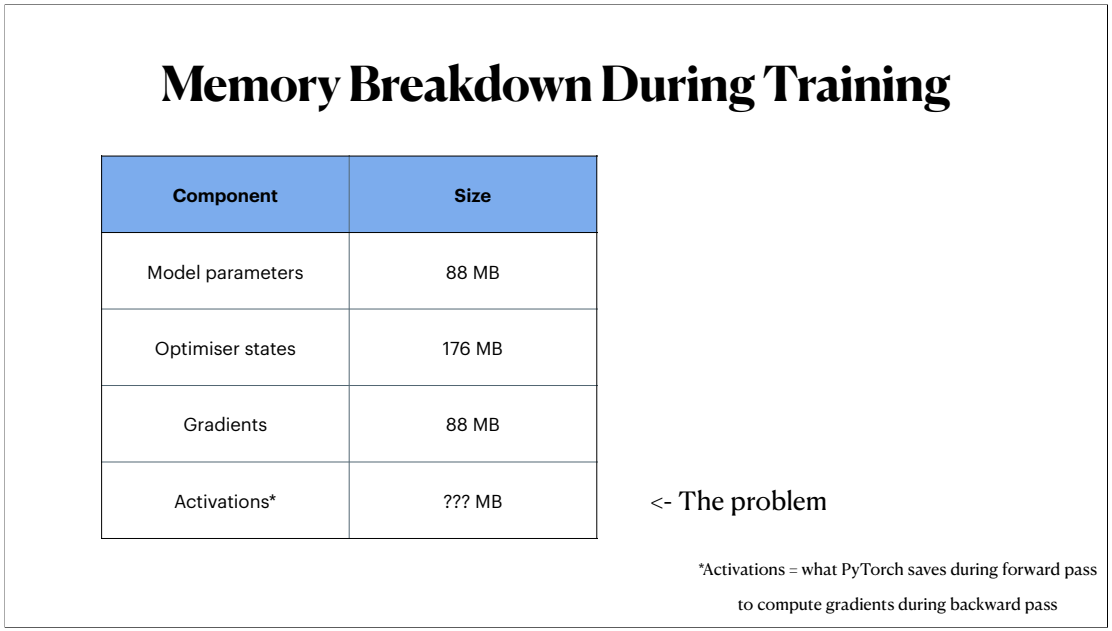
Today I want to share some findings about MedNet's performance when we try to use multiple frames instead of single images. This is relevant for anyone working with temporal medical data—like angiography sequences. I'll show you where the memory actually goes, why we hit out-of-memory errors, and what we can realistically do about it



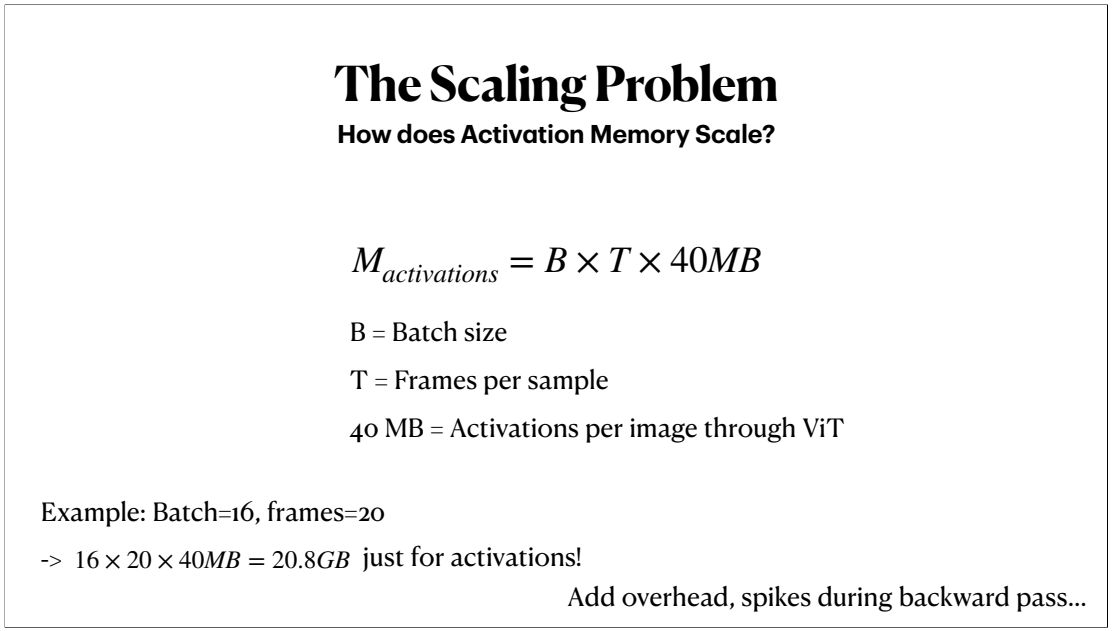
Here's the situation. MedNet works well for single-frame classification. But medical diagnoses often depend on change over time—blood flow, contrast agent progression, that kind of thing. When we try to scale to 12 or 20 frames per sample, we hit memory limits. The question is: why, and what can we do?



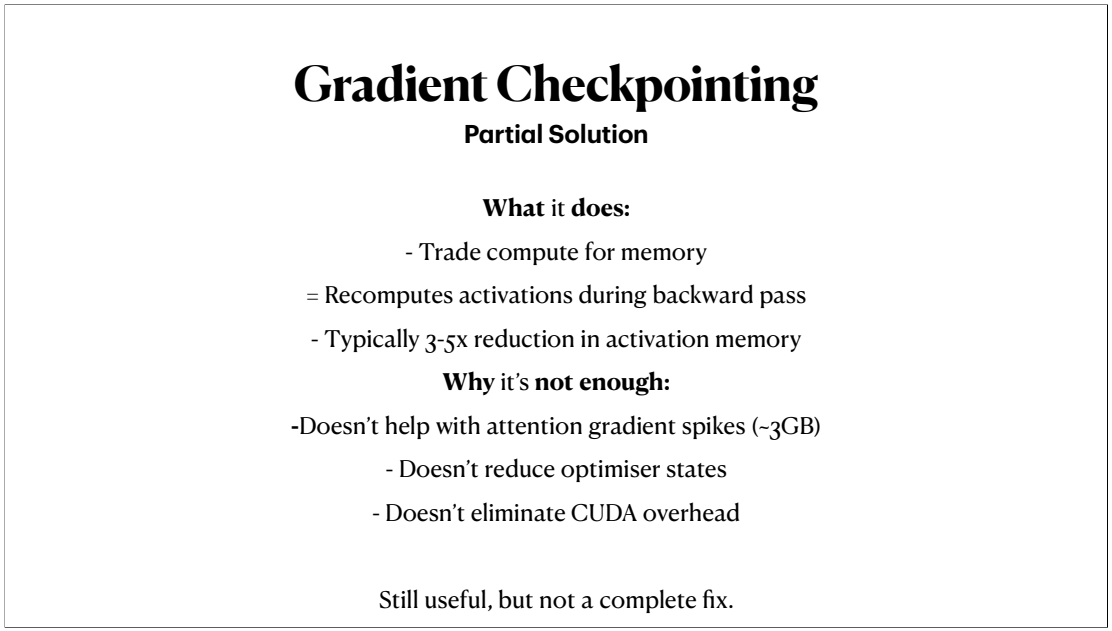
Before jumping to solutions, let's be clear about our constraints. First: compute cost—we share the cluster, so anything that multiplies training time needs a good reason. Second: it has to work with MedNet and the RetFound backbone we're already using. Third: we need to be able to explain what the model is doing to clinical partners. Any solution that fails one of these gates is not practical for us.



Let's look at where memory actually goes. The model itself—ViT-Small—is only 88 megabytes. Optimizer states double that, gradients add another 88 megs. So far we're at about 350 MB total. That's nothing on a 24 GB GPU. So where does all the memory go? Activations. These are the intermediate values PyTorch has to save during the forward pass so it can compute gradients later



Here's the key insight. Activation memory grows linearly with both batch size AND number of frames. Each image through ViT needs about 40 megabytes of activation storage. With batch size 16 and 20 frames, that's 320 images, times 40 megs, equals 12.8 gigabytes. Add the fixed costs, add the memory spikes during backward pass, add PyTorch overhead... you're pushing 20 gigs. That's why we OOM



You might think: just use gradient checkpointing, problem solved. And it does help—it can cut activation memory by 3 to 5 times. But there are things it doesn't touch. The attention mechanism has these memory spikes during the backward pass that checkpointing doesn't prevent. And it does nothing for optimizer states or CUDA overhead. So checkpointing is part of the answer, but not the whole answer

Recommendation 1

Feature Caching

Instead of : Images -> ViT -> Temporal Model -> Output (end-to-end)

Do this:

Step 1: Images -> ViT -> Features (run once, save to disk)

Step 2: Features -> Temporal Model -> Output (train this part only)

Memory impact:

- Activation memory: 12.8GB -> ~50 MB

Trade-off:

- ViT stays frozen (no fine-tuning) + Two-stage process

My main recommendation is feature caching. Instead of running everything end-to-end, we split it into two stages. First, run each frame through the frozen ViT once and save those 384-dimensional feature vectors to disk. Then train only the GRU on those cached features. This drops activation memory from 12 gigabytes to about 50 megabytes. You can suddenly use batch size 64 with 20+ frames, no problem. The trade-off is that the ViT backbone stays frozen—but we were planning to use the pretrained RetFound weights anyway

Recommendation 1

Feature Caching - Gates Check

Does feature caching pass the gates?

Compute Cost [PASS]

ViT runs once per frame, not per epoch

Infrastructure [PASS]

Uses existing ViT, adds temporal model on top

Interpretability [PASS]

Clear separation: “ViT sees, temporal model thinks”

Let's check this against our three gates. Compute cost? Actually improves—ViT only runs once per frame total, not once per epoch. Infrastructure? Fully compatible—we're just adding a GRU on top of the existing ViT output. Interpretability? Actually quite good—there's a clear mental model: the ViT 'sees' each frame, the GRU 'thinks' about how they relate over time. All three gates pass.

Recommendation 2

MedNet Needs Changes

Current **MedNet** assumes: 1 sample = 1 image

For temporal data, we need:

1. Data Loader changes
   - Sample = sequence of frames
   - batch["image"] shape: (B, T, C, H, W) not (B, C, H, W)
2. Model wrapper for temporal
   - ViT per frame -> temporal model -> head
3. Exposed optimisation flags
   - Feature caching utilities

The second recommendation is about MedNet itself. Right now, MedNet assumes one sample equals one image. The data loaders, the model interfaces, everything is built around that. For temporal data, we need the data loader to return sequences of frames. We need model wrappers that handle 'ViT per frame, then temporal aggregator.' And we need memory optimization tools exposed in the config—mixed precision, checkpointing, caching utilities. These aren't huge changes, but they're necessary.

Recommendation 2

MedNet Needs Changes - Gates Check

Does changing MedNet pass the gates?

Compute Cost [PASS]

No extra compute, just different shapes

Infrastructure [NEEDS WORK]

Requires code changes in datamodule.py etc.

Interpretability [PASS]

Standard PyTorch patterns, well-known

So, investment now, payoff for all future temporal projects.

Gate check for the MedNet changes. Compute: no extra cost, just reshaping how data flows. Infrastructure: this is the one that needs work—someone has to actually modify the datamodule and add the wrappers. But it's a one-time investment. Interpretability: these are standard PyTorch patterns, nothing exotic. The key point: this investment pays off for every future temporal project at Idiap, not just this one.

Summary & Next Steps

What **now?**

1. OOM happens because activation memory scales as B X T
2. Feature caching is the most practical solution
3. MedNet needs temporal-native support

Next steps:

- Implement feature caching for the Aptos dataset
- Valudate temporal models on cached features
- Propose MedNet datamodule changes (to you guys)

To wrap up: the out-of-memory problem comes from activation memory scaling with batch times frames. Feature caching is the practical solution that works within our constraints. And MedNet would benefit from some architectural changes to support temporal data natively. My suggested next steps: implement feature caching for the angioreport project, validate that the GRU approach works on cached features, and then propose the broader MedNet changes to the team.