ML Activities and Plans

https://indico.cern.ch/event/770307/

Quoting Doug Benjamin:

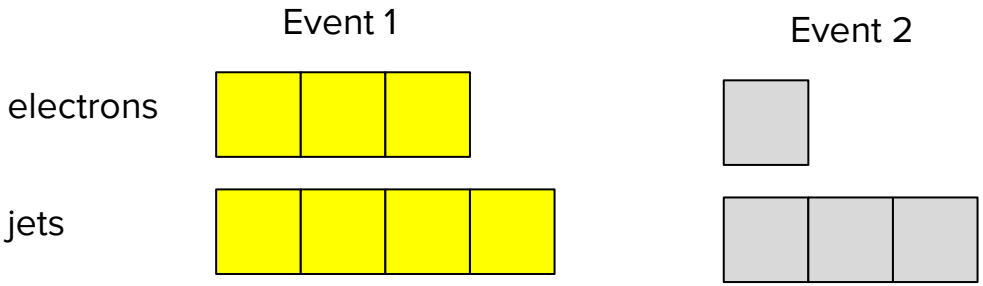
The talk should cover:

- ML status, activities, plans for next 1 year , next 3 years.
- Questions that should be answered include:
  - What is currently being done? With the goal to understand if we are in an speculative R&D phase or directed R&D.
  - What resources the ML forum needs from ADC in the next year and the next 3 years?
  - Is there a path to large scale use of ML?
  - How will it (ML) be used Physics analysis, Event simulation etc?
- **TL;DR: Things are ramping up, we need help in specific areas**

Data Access: Why ML is different

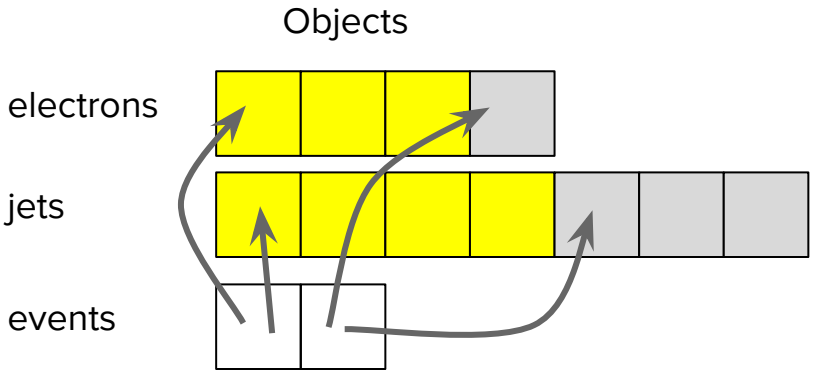
The HEP Pattern:

- event points to objects
- Events processed one at a time
- **Max “batch size” for jets is limited by number in event**

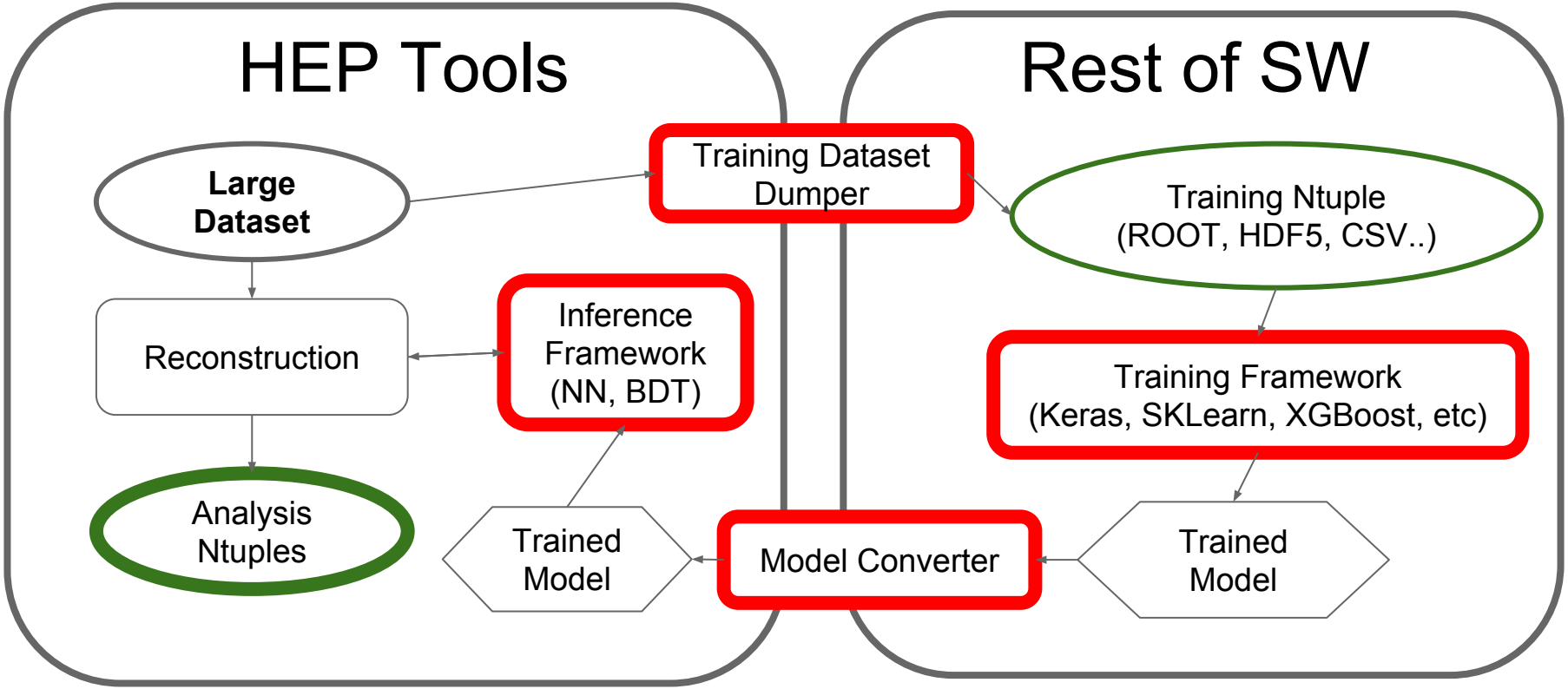


Ideal ML layout:

- **Single Instruction, Multiple Data**
  - **SIMD for jargon-lovers**
- One dataset for each object
- **Objects processed in batches**
  - Around 1k per batch
- Can recover events from offset
- In practice each object is labeled



The Machine Learning Workflow



Where are the bottlenecks?

- The **training** is **slow**
  - **This is also the obvious place to use GPUs**
  - **But it runs once, doesn't use that much processing overall**
- The **inference** is **what we run most frequently**
  - **I get the impression that this is where people want to use GPUs**
  - **At the moment it's not a bottleneck**
- **Dumping datasets was a problem**
  - **Not so much these days**
- **Model conversion** is a **psychological** barrier
  - Not so hard if you understand your model
  - But annoying if you have to debug it

Bottlenecks

Inference support: We're fine

- lwtnn supports leading edge, widely used (in release 21.2)
  - No convolutions, more on this later
- Interface makes it stupid-easy to apply:

```
// get your saved JSON file as an std::istream object
std::ifstream input("path-to-file.json");
// build the graph
LightweightGraph graph(parse_json_graph(input));
```

```
// compute the output values
std::map<std::string, double> outputs = graph.compute(inputs);
```

- **Note the lack of global state: this is easier than *anything* in ROOT**

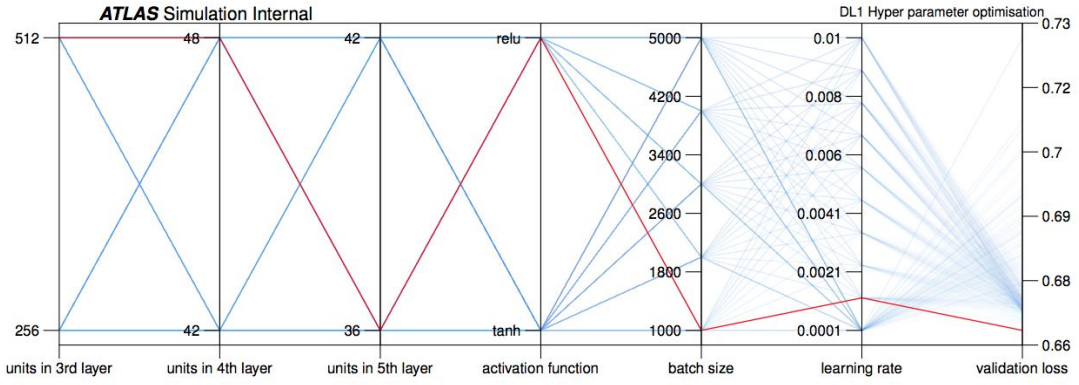
Inference: What about CPU, memory?

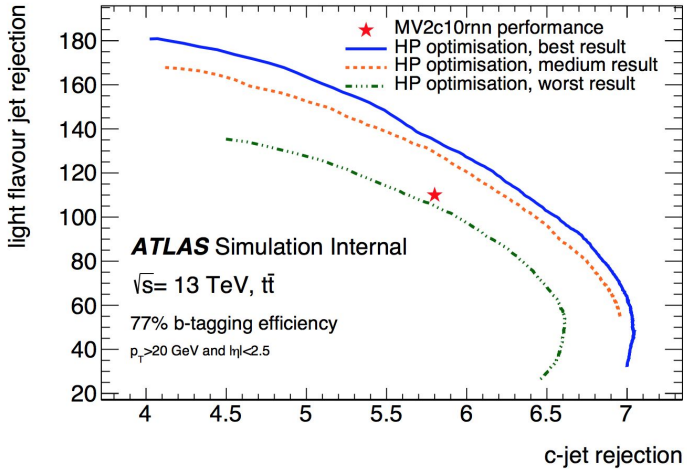
- Current NNs are simpler than BDTs we've used since run 1
  - Latest round of flavor tagging may add a few ~5 MB NNs
  - We've made **absolutely no effort** to prune (or combine) these NNs
    - **Tones of low hanging fruit here if memory becomes a problem**
    - **No need to touch the implementation, just smarter training / organization**
- I've **tried** to find a CPU bottleneck here
  - With **6 versions of the flavor tagging**, NNs / BDTs *might* use as much as secondary vertices
    - **We're working to factorize FTag code**, might have more numbers soon
  - Bottlenecks are elsewhere

- **Bottom line: We still have *a lot* of room here**
  - **Nothing is "on fire"**
  - **We'll almost certainly hit a bottleneck in the longer term**

Training: Hyperparameter Scans

- **Pilot program:** b-tagging hyperparameter scans (link to Manuel Guth's plots)
  - **Big improvement in performance**, will go into run 2 analyses
    - It's nice to show that **QT students can do useful work**
  - Hopefully expand methods to other CP / physics groups
  - Typical jobs take 5 minutes

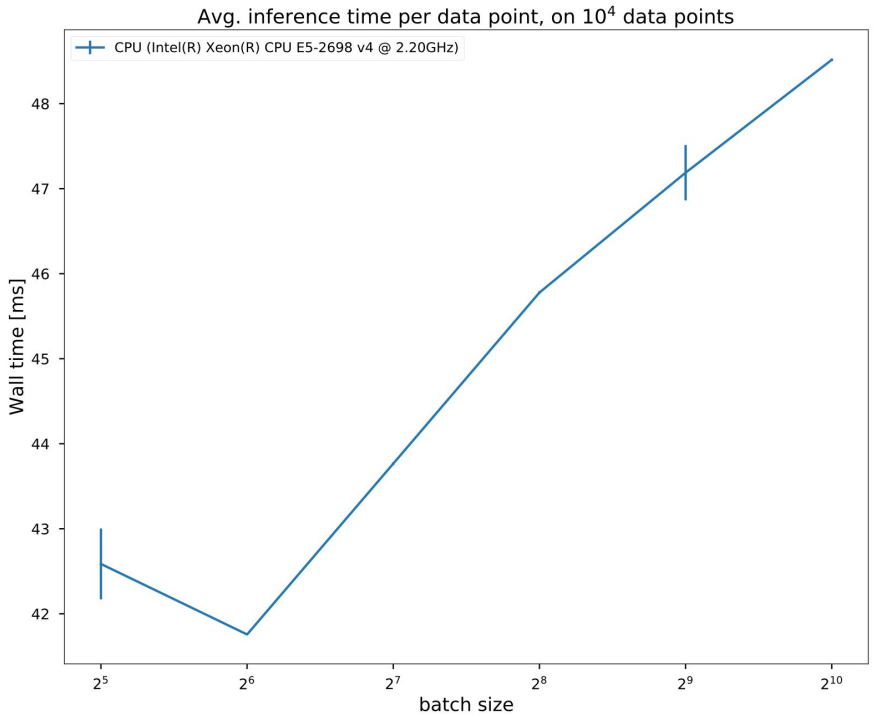


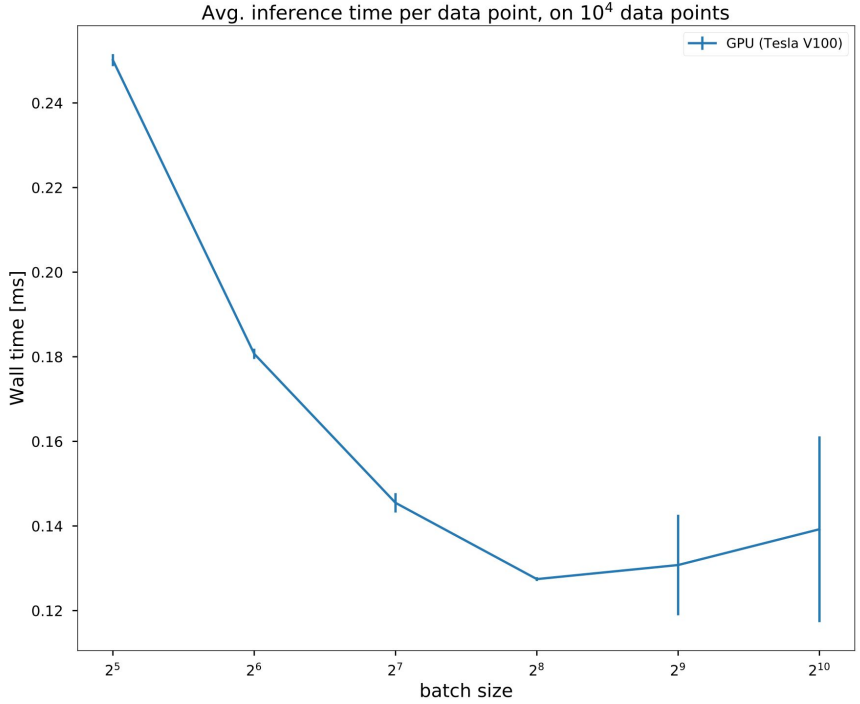


Training: The slower NNs

- This is **more fun for computing people**
- E/Gamma Energy regression:
  - Training time **1 day**
  - See more: https://indico.cern.ch/event/800614/contributions/3327152/
- Calo GAN:
  - Training time **3 days**
  - See more: https://indico.cern.ch/event/800614/contributions/3327155/
- These are just two examples, we'll have more
- **Good candidates for distributed training**
- **Batch processing is essential here**

CNN: Batch Size vs Time





“You have 3,500 Physicists”

(So why aren’t they all doing machine learning?)

Bottleneck Starts Upstream

- Plenty of uses for ML, too much faffing with data
  - We often want **lower level data**, interaction with primary AOD / DAODs
- People are *really* confused about data access
  - Anonymous postdoc: "**reading an xAOD is non-trivial**"...
  - ... **but it *is* trivial, people just don't know this**:

```
xAOD::Init().ignore(); // <-- some global magic
xAOD::TEvent event(xAOD::TEvent::kClassAccess);
std::unique_ptr<TFile> ifile(TFile::Open(file_name.c_str(), "READ"));
event.readFrom(ifile.get()).ignore();
event.getEntry(0);
const xAOD::JetContainer *jets = 0;
event.retrieve(jets, "AntiKt4EMTopoJets").ignore();
std::cout << "jets in entry: " << jets->size() << std::endl;
```

Dataset_Final_updated_v2.h5

- Writing a training dataset is also pretty easy:
  - Make a file: `H5::H5File file("name.h5", H5F_ACC_TRUNC);`
  - Set up a writer (for the first 5 jets)

```
H5Utils::Consumers<const xAOD::Jet*> cons;
cons.add("pt", [](const xAOD::Jet* j){ return j->pt();}, NAN);
cons.add("index", [](const xAOD::Jet* j){ return j->index();}, -1);
H5Utils::Writer<1,const xAOD::Jet*> writer(file, "jets", cons, {{5}});
```

  - Write it out in an event loop:

```
writer.fill(*jets);
```

- **Takes minimal effort, runs in AnalysisBase**
  - **Easy to dump multiple rank N datasets in parallel → SIMD compatible out of the box**

Dumping Info: Computing Overhead

- Size of Training Datasets: Negligible, **several GB**
  - On the scale of analysis workflows this very small
    - My analysis saves **9 TB** of ntuples, runs at **20 Hz** ...
    - ... and takes 2 months to go from xAOD -> histograms...
- Some people have asked about “direct” xAOD access from Keras, PyTorch, etc
  - This *massively* complicates the dependencies on both sides
    - ROOT is **not** an ML dependency, this is a **Very Good Thing**
  - If you *already* have ntuples you can convert with **uproot**

- **Bottom line: so far haven’t seen a big bottleneck**
  - **Plenty of industry solutions being investigated**

On Training: Public Resources Wanted

- Training requires:
  a. **Training data** ← requires someone who knows the EDM
  b. A **grad student** ← plenty who want to take part
  c. **Software** ← minimal ML stack
  d. A **GPU** ← preferably one with a responsive admin
- Historically **we haven't always been able to match these** up
  - Lots of training on CPUs while **GPU clusters go unused**
- The training environment is diverse
  - **Docker containers will be great**
- **We also need local resources for prototyping / training**

Training Image Wishlist

- See a number of previous talks
- Huge wishlist for containerization effort:
  - Support for non-CentOS operating systems (i.e. ubuntu)
  - **Dockerfile / CI examples**
    - ML software moves very fast, should encourage experimentation
    - **More complaining**
  - Image distribution
- Also some things from panda:
  - **Arbitrary job metadata?**
    - Would be nice to query via panda monitor

- **There's lots of work to do here, top priority in the next year**

Model Conversion and Bookkeeping

- Most popular frameworks are TMVA and Keras
  - TMVA (BDTs) supported by custom ATLAS code
    - Seem quite stable, so far I've seen no interest in extending these
  - Keras models converted to JSON via lwtnn
    - Plain text format, we could zip I suppose...
  - The GroupData area is **awesome** to store models. No issues so far
- **Some people have expressed interest in PyTorch**
  - So far they don't seem very serious, could usually use Keras anyway
  - If anyone wants to use pytorch, we should talk about conversions
- **Bottom Line: not on fire**
  - **some help evaluating interoperability of the models welcome**
  - **I'm trying to get the TMVA team interested...**

Longer Term (Next 3 Years)

Inference Revisited

- lwtnn (and similar tools) might not scale forever
  - **We don't support convolutions**
    - **... because they haven't been useful**
  - We might get some help from Alice
  - But for now **I just add things as they are needed**
    - To be fair: this has been *much* less work than adding an external framework
- CMS took a **very different** approach
  - See Feburary 7th Core SW meeting
  - Integrated both TensorFlow *and* MXNet
    - This took them a huge amount of work ➜ we'd have to start soon to follow suit
    - So far they aren't doing batch processing either, nor using GPUs for inference

DL Frameworks in Reconstruction

- Two options:
  - **Aim for the ground, see if we crash**
    - Again, CPU isn't a problem yet...
  - **Start playing with something now**
- We're looking into using TensorFlow
- Lessons from CMS:
  - **Two frameworks is better than one** → They installed both MXNet and Tensorflow
  - **Threading is hard** → TF is as intrusive as Athena, lots of custom hacks needed
  - **Batch processing is really hard** → They still run one pattern at a time

So where does that leave us?

- We have a lot of use cases for **embarrassingly parallel** ML
  - Making an effort to open up all available resources
  - **Hope to expand pilot effort from b-tagging to other uses**
  - I doubt training will use a large fraction of GPU, though…
- *Some* cases for distributed learning
- Inference with GPUs in reconstruction is a work in progress

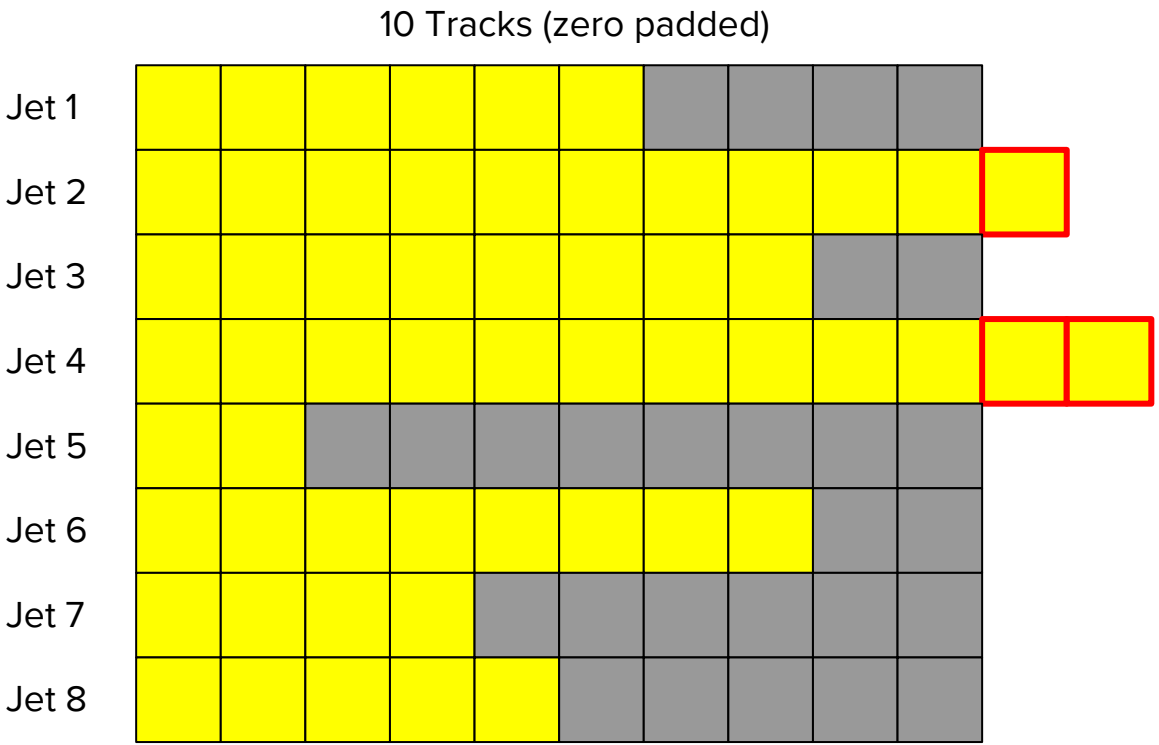
- A few things in the way of ramping up:
  - The “easy” ones: **we need to write / interface to software. And train algorithms.**
  - The less easy one: requires us to **think differently about our data**, in a more SIMD-friendly way

Backup

What About Variable Length Arrays?

- Assume one pattern per jet
- The limitation here is numpy
  - Likes square blocks
- RNNs *are* variable length
- Masking makes batch processing possible
- In practice: some **truncation**

- **My take: ugly but fine**



Images: in search of a “killer app”

- Images **inflate the dimensionality** of the input space massively
  - This is a perfect use for GPUs
- First “jet image” paper I could find was 2015
- Franchesco Rubbo, October 2017:
  - ‘Indeed we don’t have a conclusive "killer app", yet…’
- As far as I can tell, nothing has changed in 2019
  - Meanwhile feed-forward networks, RNNs, and adversarial networks have been integrated into reconstruction (and the trigger)
  - **My take: we need grad students to dig deeper into reconstruction**
- **To me “images” are still unproven**
  - **We can do the same thing with a simpler representation**

The Pipeline Problem (more for ASG?)

- Like it or not, we run on grad student sweat (and sometimes tears)
- How do we ramp up ML studies?
  - Need to teach grad students **the EDM**
  - I consider EventLoop, Athena, ToolHandles, etc to be secondary
    - To say nothing of CxAODs, framework wars…
- Biggest obstacle to doing ML is still data prep
  - Again, this is **NOT** a problem going from final DS ➔ GPU
  - The issue is complexified workflows
- I don’t expect this to change overnight
  - We’re working to provide better workflows (b-tagging is a pilot now)

Peter Principle

- The original:
  - “every employee tends to rise to his level of incompetence.”
    - ... then they are no longer promoted
- For data pipelines:
  - “Every framework grows until grad students no longer want to produce it”
    - ... then they write another one using the output