Experience With

Model Serialization (mostly in ATLAS)

https://indico.cern.ch/event/1197680/contributions/5144405/

Motivation: Production ML in HEP

HEP Land

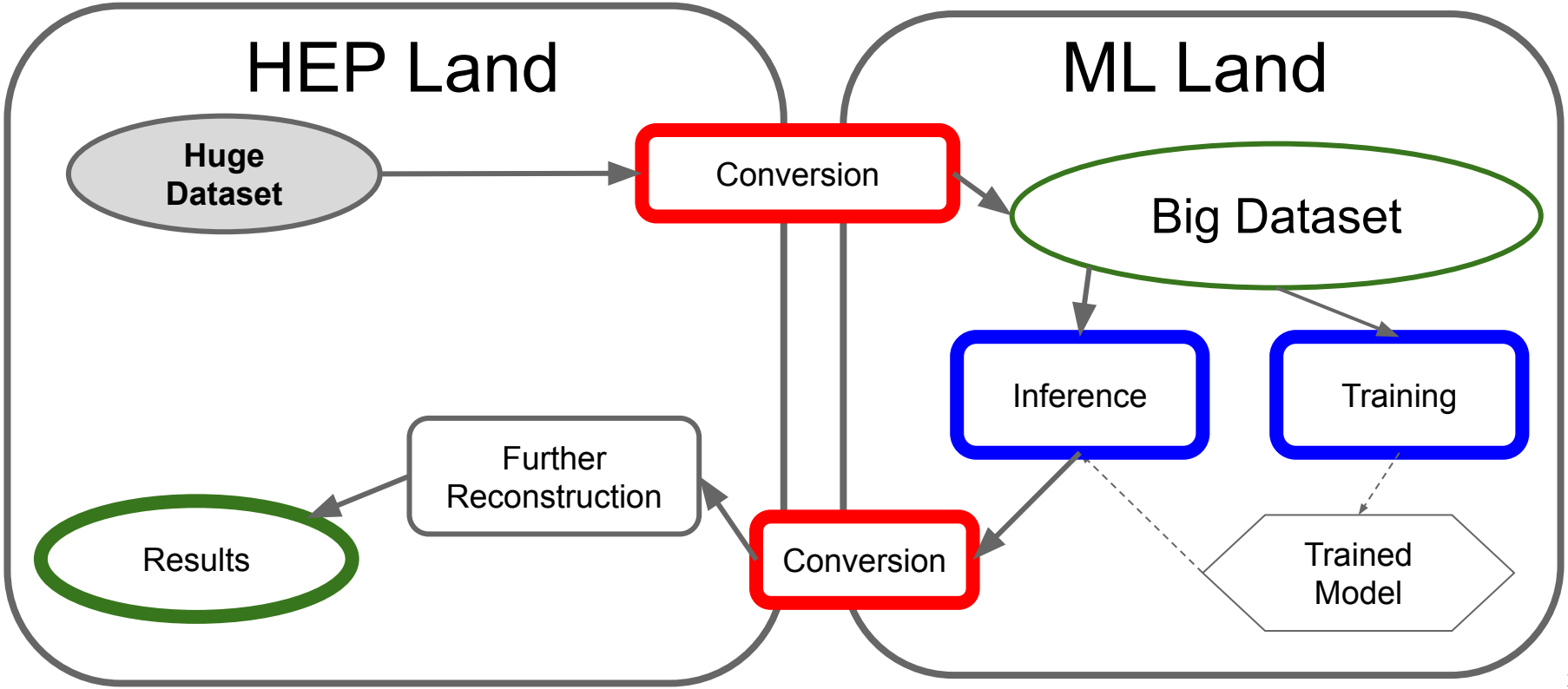
- Lots of C++
- Event Pipelines
- Lots of legacy
- Moves slow

ML Land

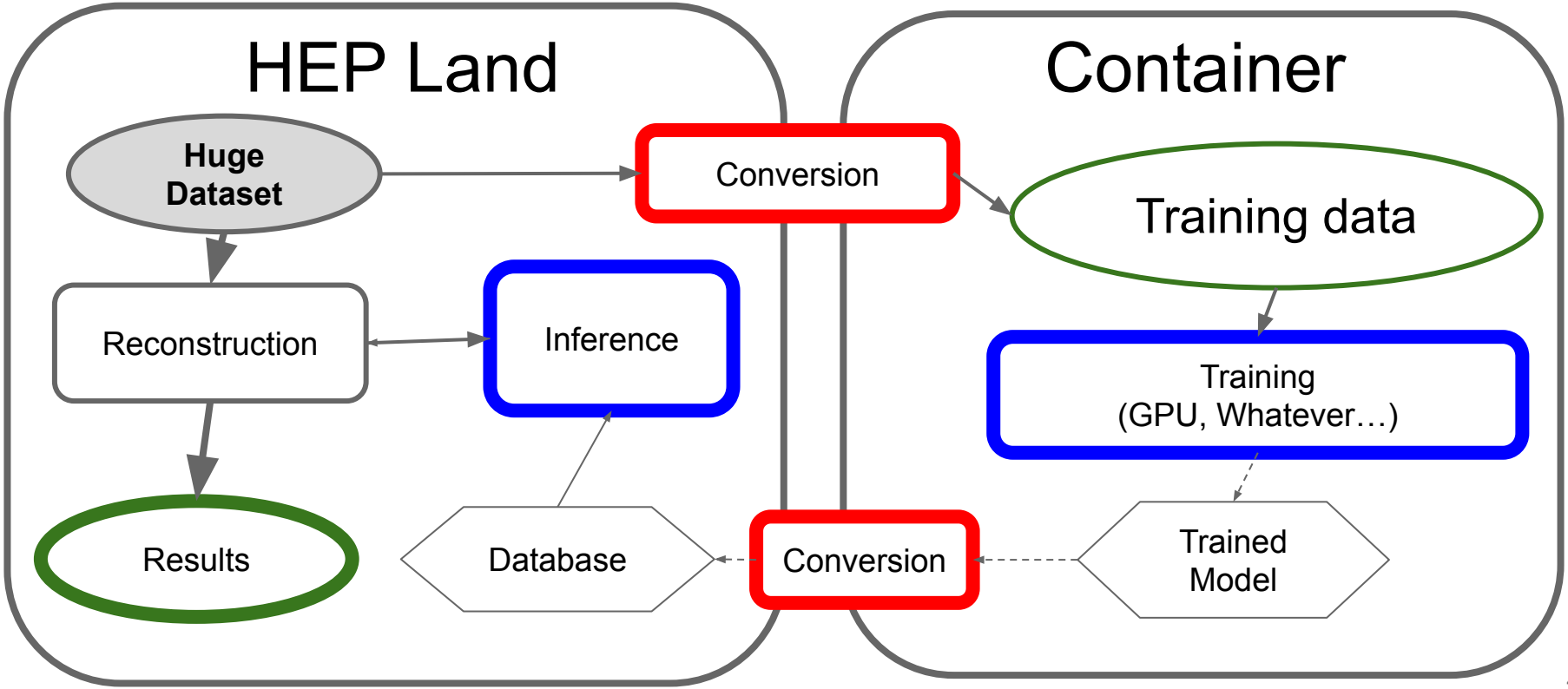
- Python
- Batched
- Minimal legacy
- Moves fast

- Many Dependencies

Default solution: Shipping the data

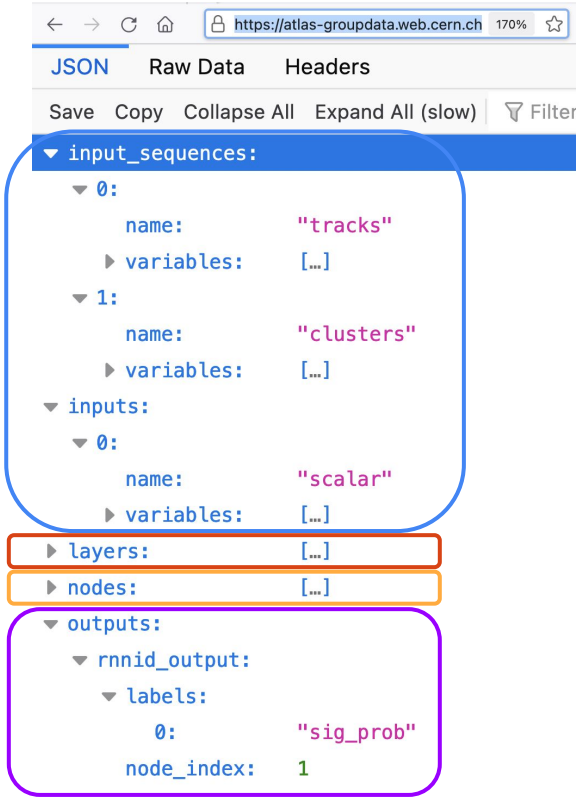


Shipping the model



Back in the dark ages... (Sep 2015)

- There were no inference engines
- Every ML framework
  - Had its own serialization
  - Did its own inference
  - Had a million dependencies
  - Had zero stability
- So we wrote our own (lwtnn)
  - Serialized to JSON
  - Separate **inputs**, **outputs**, **data**, **graph**
  - Started small: sequential models
    - Latter added graphs
- We store networks on a file system
  - Write only, locally cached on demand

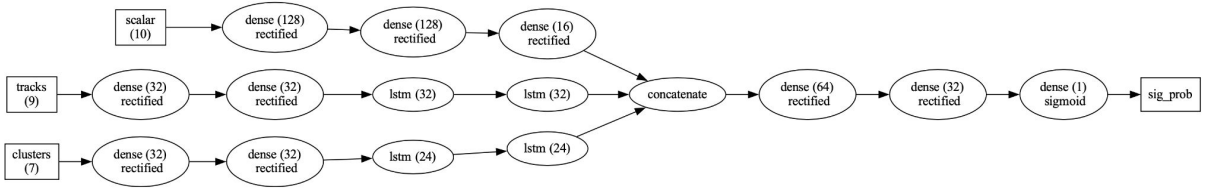


random Tau ID model I found

Model exploration (tau again)

```
> curl -s https://atlas-groupdata.web.cern.ch/atlas-groupdata/tauRecTools/R22_preprod/rnnid_mc16d_config_3p.json | egrep 'name'
    "name": "tracks",
        "name": "pt_log",
        "name": "pt_jetseed_log",
        "name": "d0_abs_log",
        "name": "z0sinThetaTJVA_abs_log",
        "name": "dEta",
        "name": "dPhi",
        "name": "nInnermostPixelHits",
        "name": "nPixelHits",
        "name": "nSCTHits",
    "name": "clusters",
        "name": "et_log",
        "name": "pt_jetseed_log",
        "name": "dEta",
        "name": "dPhi",
        "name": "SECOND_R",
        "name": "SECOND_LAMBDA",
        "name": "CENTER_LAMBDA",
    "name": "scalar",
        "name": "centFrac",
        "name": "etOverPtLeadTrk",
        "name": "dRmax",
        "name": "trFlightPathSig",
        "name": "SumPtTrkFrac",
        "name": "EMPOverTrkSysP",
        "name": "ptRatioEflowApprox",
        "name": "mEflowApprox",
        "name": "ptIntermediateAxis",
        "name": "massTrkSys",
```

- Easy to answer “what inputs were in this NN”
  - The primary source is just one `curl` command away
- We also added some graph parsing tools
- We could compress to 30% of the current size
  - Before compression, most NNs smaller than 2 MB
  - Never bothered in practice



And that worked good for a while...

- A lot of NNs weren't worth saving
  - We never had to deal with convolutions
- Some things actually got simpler
  - **RNNs → Deep Sets**
- Serialization was just small tweaks

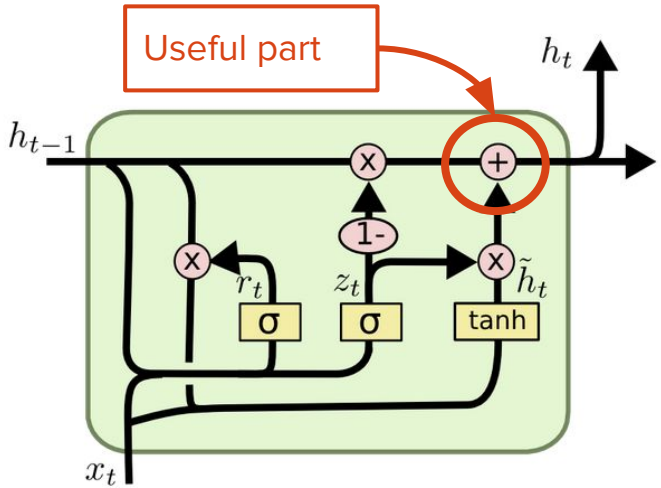


Image credit: Christopher Olah

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

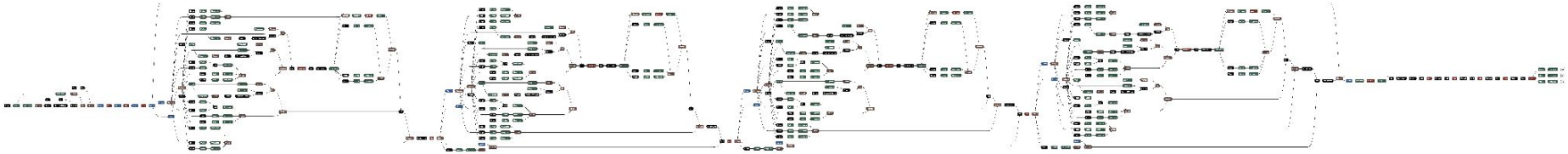
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

$$h_{\text{final}} = \sum_t x_t$$

Two developments changed things

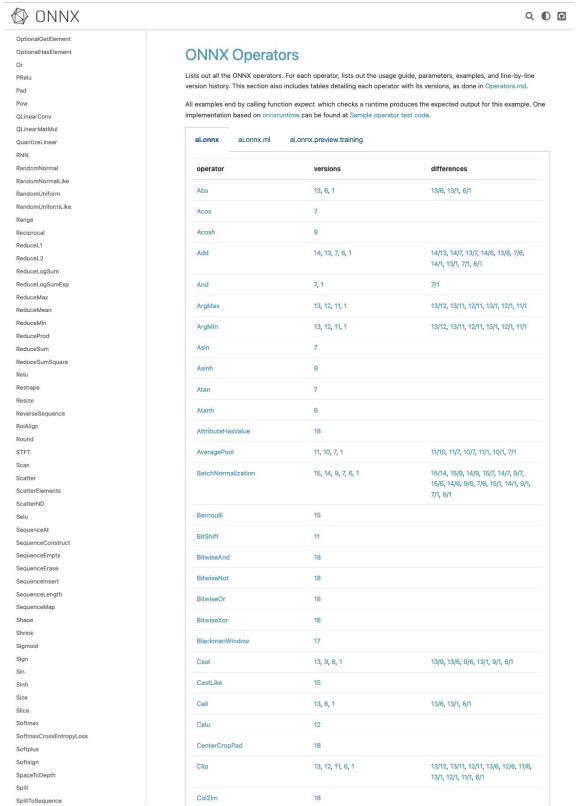


Above: ATLAS trigger b-jet discriminant, via netron

1. Graph networks / **attention** → Actually a pretty good idea
2. More industry support for inference
   a. We installed ONNX Runtime (not to be confused with *an* ONNX runtime)
      i. reads Open Neural Network eXchange files
      ii. **Took O(1 year)**, still build **with patches**
   b. And started supporting ONNX files

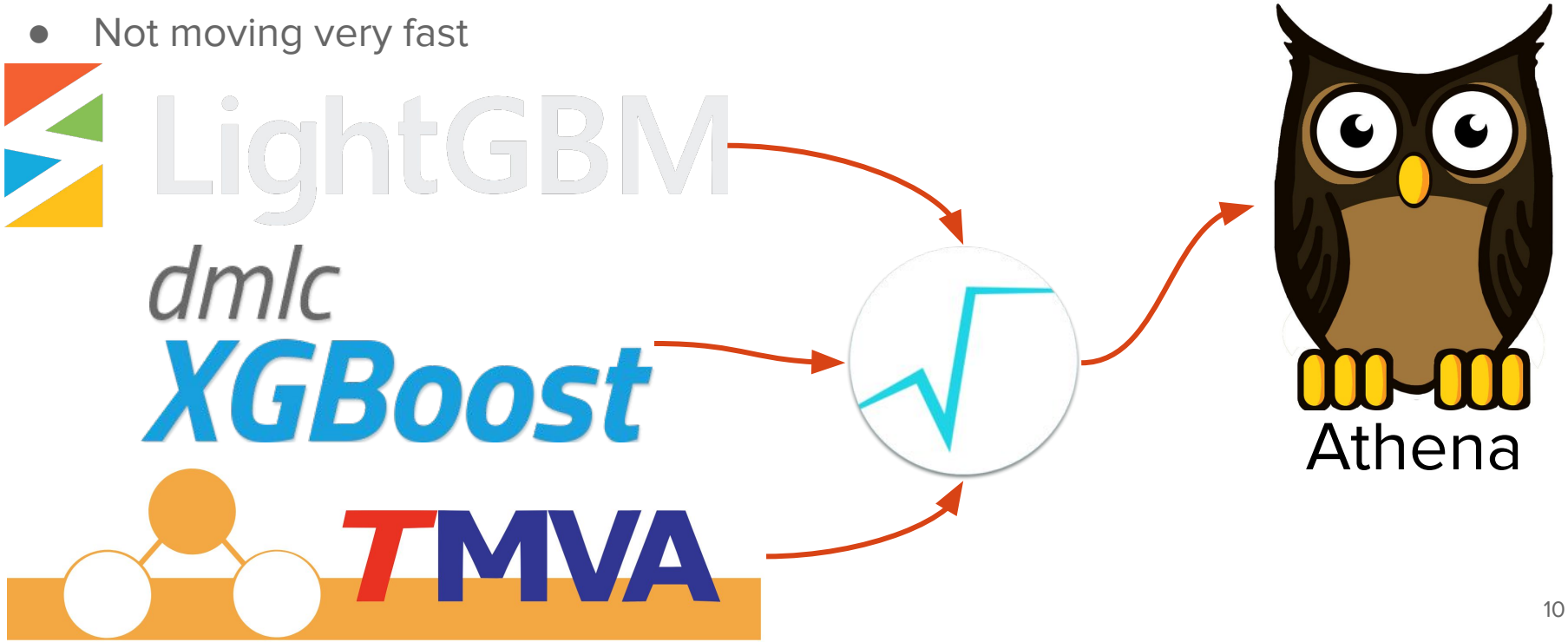
What is great / terrible about ONNX

- Supports **many operators**
  - around **187**, lwtnn is around 20
  - They are carefully **versioned**
  - … but still not universal support
- Lots of **tooling for interpretation**
  - Visualization, autodiff, inspection
  - Many “runtimes”
    - in practice we only use one
- Allows **metadata**
  - Protobuf is less readable than json
  - We added some scripts to extract it



Meanwhile, BDTs

- ATLAS has MVAUtils in Athena, stores BDTs as ROOT trees
- Not moving very fast



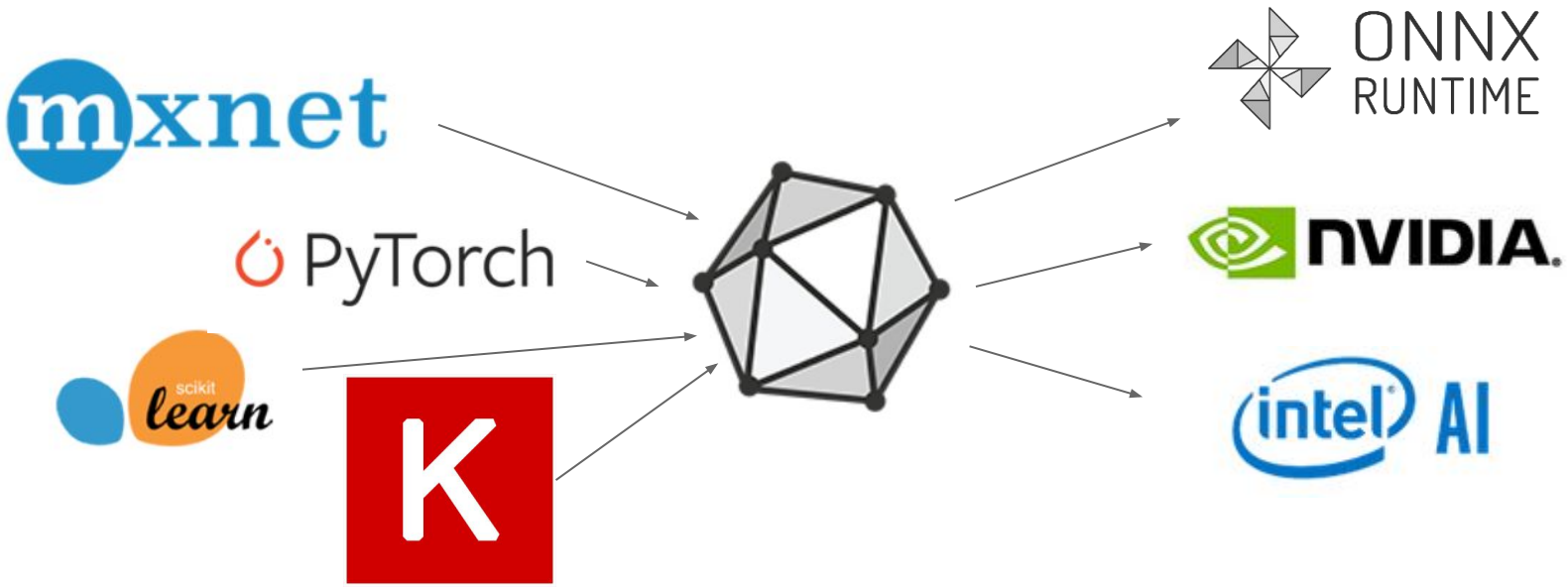
What has worked well generally

- Store endpoint metadata
  - ML frameworks map vector to vector ➔ most bugs are swapped inputs
  - We had to write some tools for ONNX
- **Write with reuse in mind**
  - Avoid unversioned code, notebooks, etc
  - Avoid data dependencies
- **Goals beyond papers and talks are important**
  - Code / data as a citable product would be nice
  - (more specific to production) The training pipeline should be viewed as a dependency

What's next?

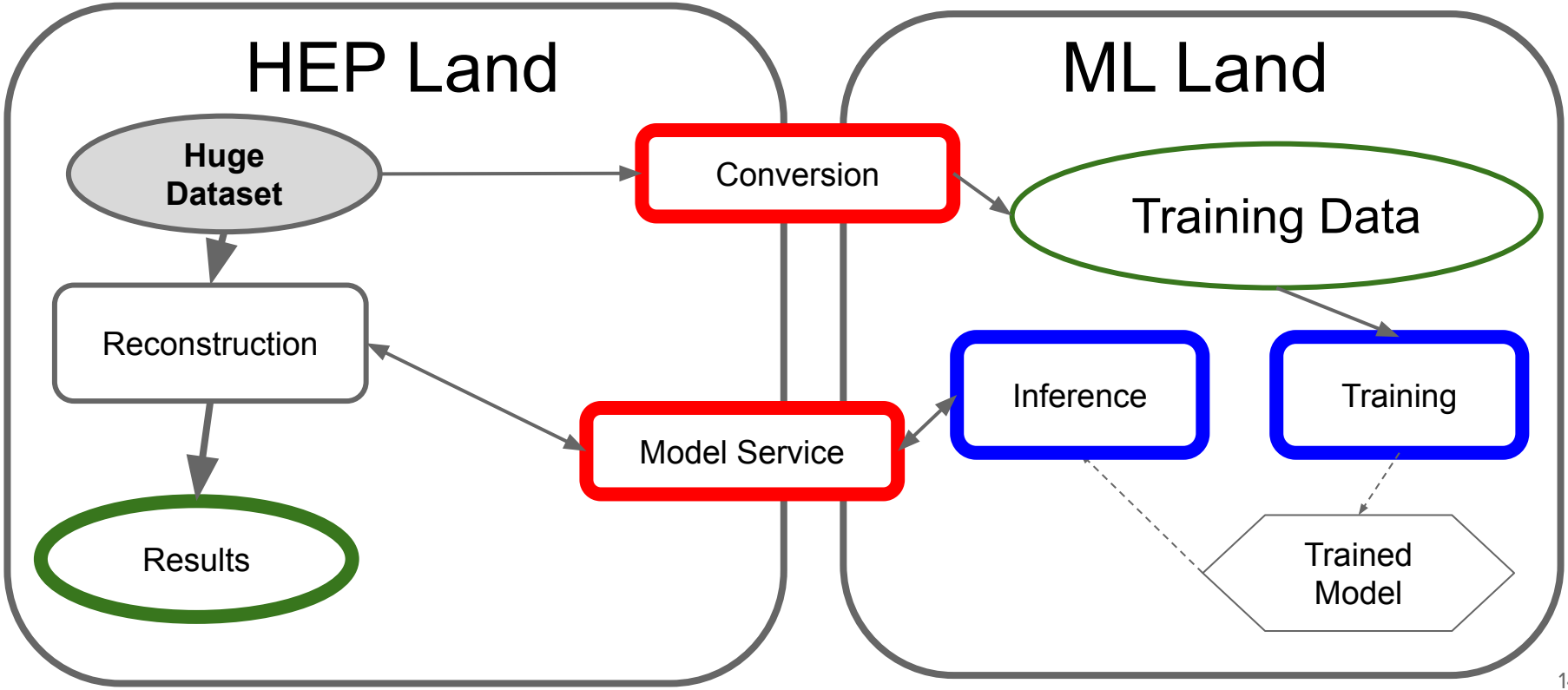
(wild speculation starts here)

Will ONNX save the day?



- Pro: there's a [well defined specification], with many community contributors
  - Brought to you (in part) by Microsoft. What a time to be alive!
- Con: common standards always lag, **big players have their own inference frameworks**

Or should we just use containers?

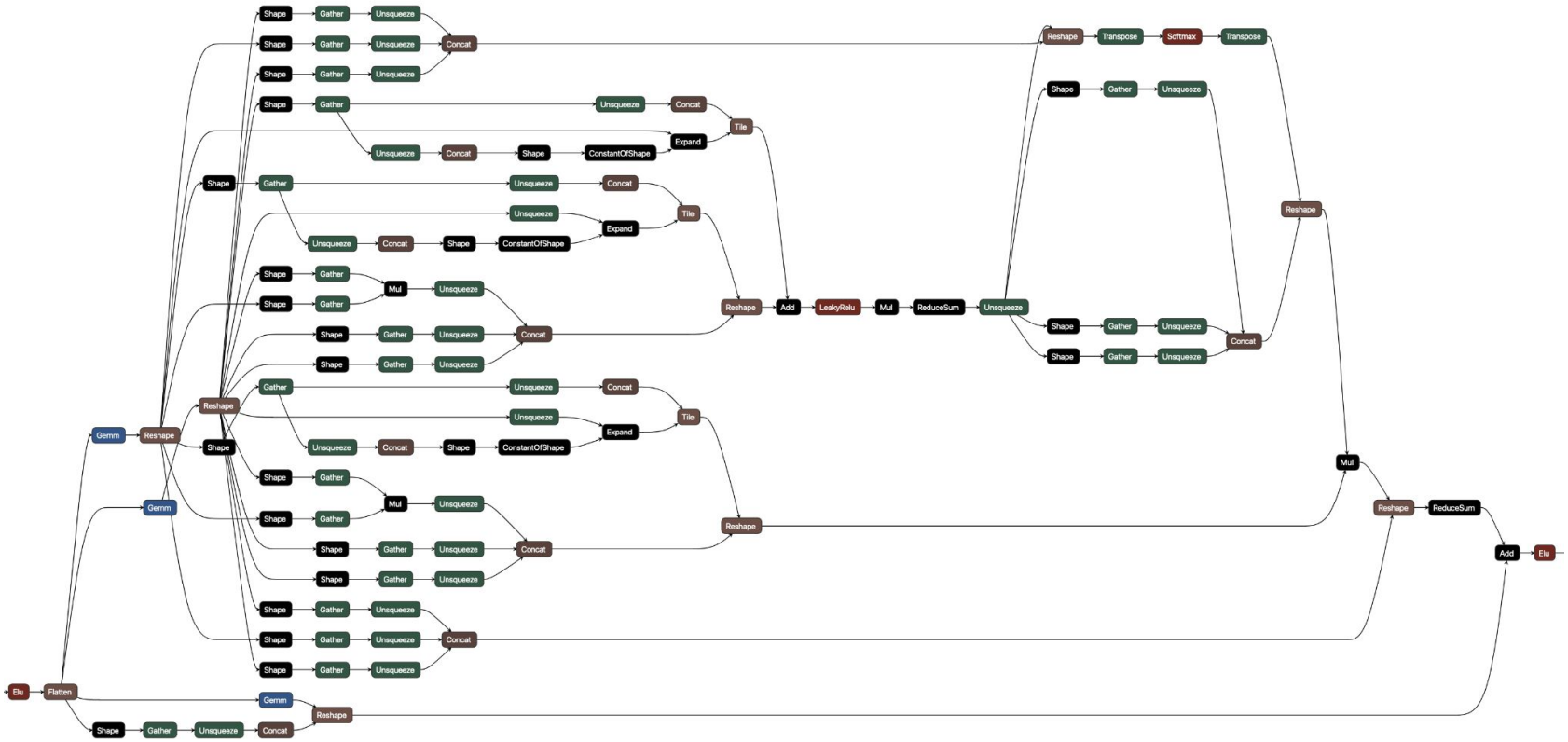


Review: Open Questions

- Is there a need for a lighter format than ONNX?
  - How about a lighter library than ONNX Runtime?
- What about BDTs?
  - There doesn't seem to be a standard here
- How much should we "containerize"?
  - But model conversion isn't "free"
    - It's sort of annoying
  - Passing data to a container is the ultimate "black box"
    - Not great if you want to compare models
- Do we want to preserve the final NN, or the whole optimization?
  - In production we consider the training pipeline important

Backup Slides

About that attention unit



(Failed) Model Exploration

https://netron.app/?url=https://atlas-groupdata.web.cern.ch/atlas-groupdata/BTagging/20220813trig/gn1/antikt4empflow/network.onnx

The pain points

- Experiment operates like a franchise
  - Many autonomous groups
  - (relatively) small central budget
- We're light on "software engineers"
  - A good data-heavy shop is 1:1 engineer:researcher
    - We're pretty much all researchers
  - Our engineers are often short tenure

- Machine learning software is ugly
  - Fast moving, with lots of dependencies
- ATLAS has one repository for trigger, reconstruction, simulation, and analysis

More network exchange formats

https://www.khronos.org/nnef

Room for collaboration

- Build a common minimal-dependency library
- Tools to interface with / examine ONNX models
  - We might even become contributors
- Better container infrastructure / training