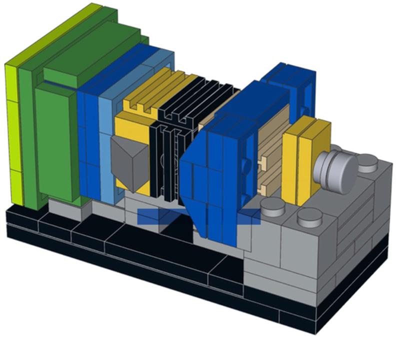
ML at LHCb

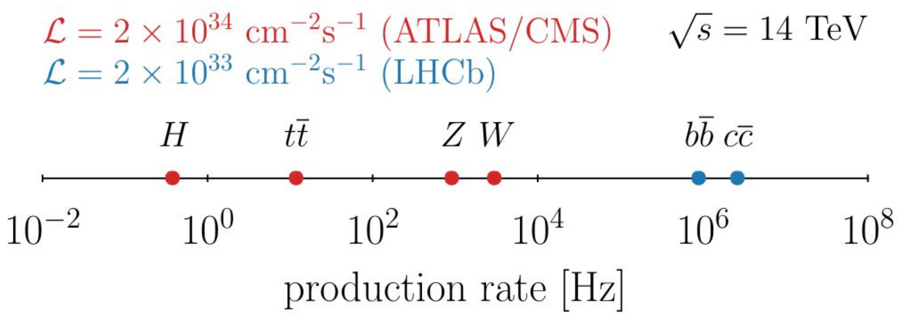
in general and at Nikhef



Maarten van Veghel,
with input from **Jacco de Vries**

Context (of online use) of ML at LHCb

- **LHCb** studies mainly decays of ***beauty*** and ***charm*** **hadrons** with **high signal rates**

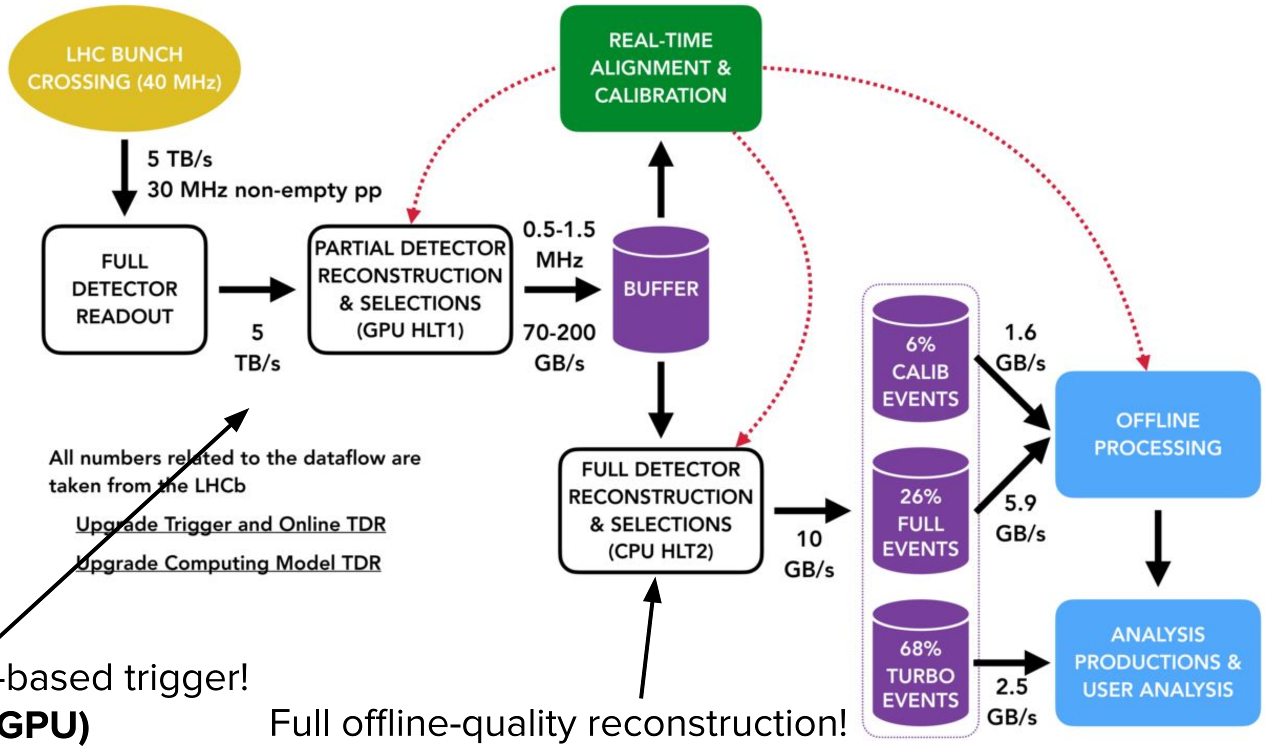


LHCb-PROC-2022-010

- **DAQ running at 40 MHz** to cope with **high signal rate**
  - *Reconstruction and selection* with as **many features** as possible, as **early** as possible
- Extract information from tracking sub-detectors and subsequently ***reconstruct*** and ***select***
  - Make use of **Machine Learning** (inference) **at earliest level as much as possible**
    - **Typically small (fast!) models with high-level quantities as input (around 10 - 20 typically)**
  - **Focus on online application** (first), as it is (probably?) **most unique about the LHCb ML situation**
    - Resources almost all at LHCb

Data flow of the current detector

LHCB-FIGURE-2020-016



Direct software-based trigger!
HLT1 (GPU)

Full offline-quality reconstruction!
HLT2 (CPU)

ML infrastructure *in online environment of LHCb*

- **Online environment needs**
  - **Most of all high speed!**
  - **Fast turn around time** of training and deployment, ...
  - **Common tools / standardization**
    - avoid customization / hard coded solutions as much as possible
    - improve maintainability and ease of use
  - **Production level code needs a lot of testing**
    - Run ML pipelines in CI/CD (Gitlab/Jenkins)
      - Also for fast turnaround time!

- **First developments now in production for HLT2 (CPU)**
  - Most applications, most interactions with ‘users’
  - **First focus on fastest algorithms, also have simplest models!**

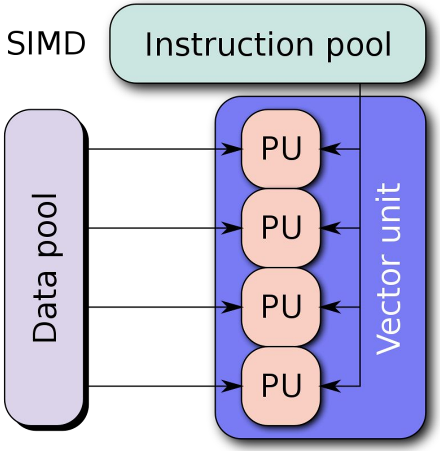
- But in the **future more emphasize on general libraries and GPUs**, developments ongoing
  - More challenging setup with demands on GPU/CPU compatible libraries and speed

Fast inference in HLT2: CPU

- Fast inference of **relatively simple models** (MLPs)
  - **Shapes of models fully set at compile time**
  - Custom implementation within Gaudi framework
    - Allows full control (of speed ups)
    - Typical MLP layers supported
    - Integration with (SIMD) event model

  - **Evaluation using SIMD**
    - Automatic batching when running over ranges like *std::vector* with non-SIMD event model
  - **Weights loaded during configuration from database**
    - Allows flexibility with retraining and deployment

- **Training infrastructure**
  - API with **PyTorch**
    - Regression test to ensure similarity
    - Easily extendable to other training software
  - Example of **training runs in CI / Jenkins**

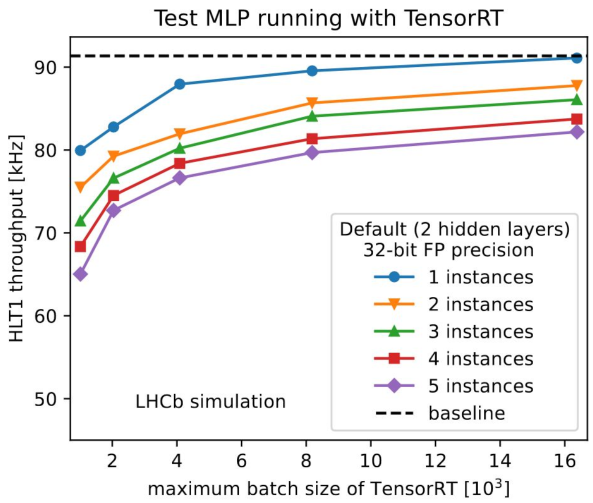


Fast inference in HLT1: GPU

- **Different beast than CPU**
  - Typically different (and more) memory and dimensionality considerations and constraints
  - Needs to run on **both GPU and CPU**
    - Both ***TensorRT*** and ***ONNXRuntime*** availability

- Number of neural-net implementations are only **increasing**
  - efforts going into right direction,
    but currently no general infrastructure

- Effort ongoing here at Nikhef, lead by Roel Aaij, on making available **general inference libraries and infrastructure**

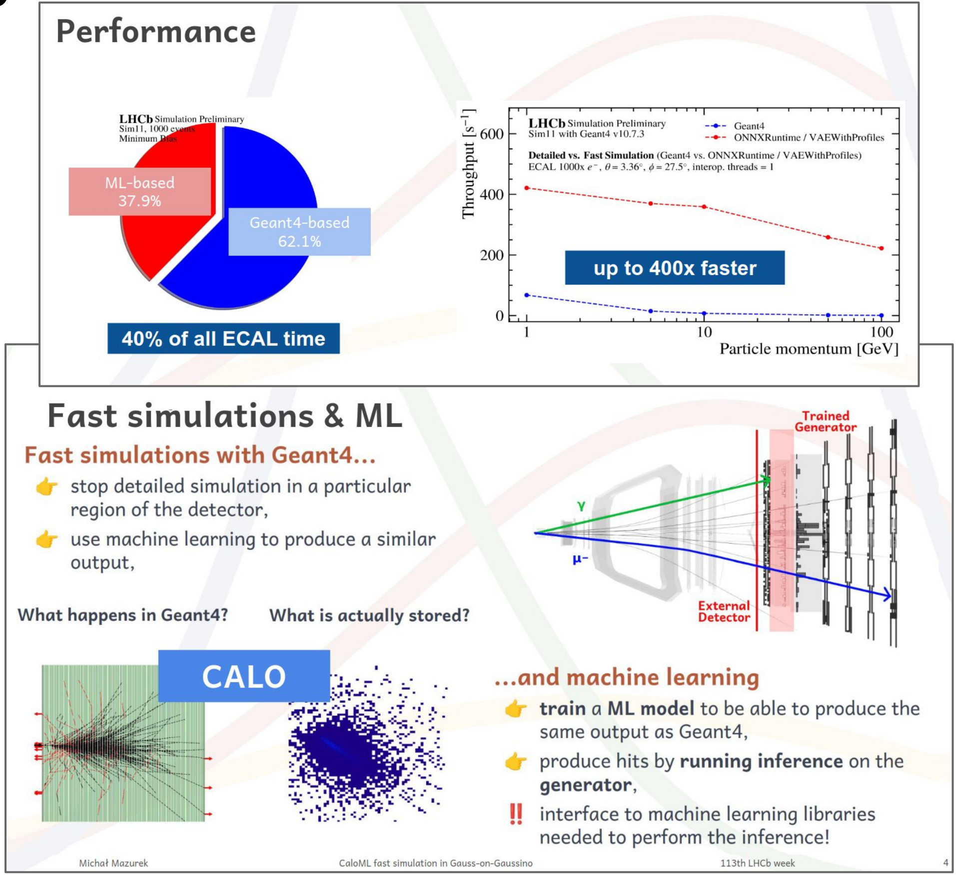


See e.g. LHCB-FIGURE-2023-006

Offline use of ML at LHCb

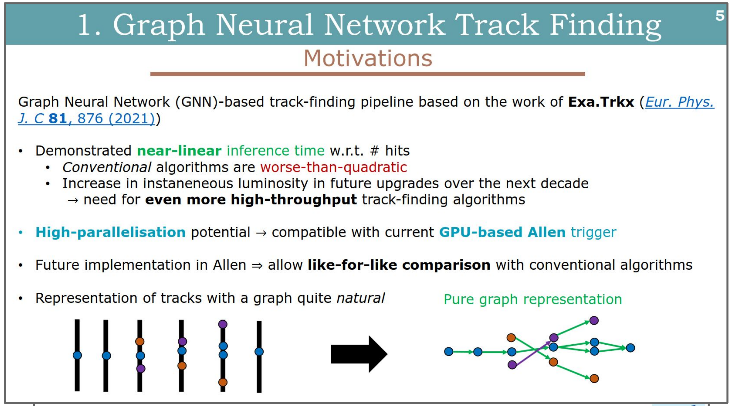
- Most applications in **analyses**, typically rather simple models are sufficient (BDTs, MLPs, …)
  - I'm not aware of more sophisticated models here, or even need of?
- Use in **simulation** is almost production ready
  - VAEs, but also even more classical methods like point libraries
  - Main care had to be taken in validation and calibration
- Specific applications like **flavour tagging**
  - Main use is BDTs
  - Use of Transformers, GNNs, etc… are still in development

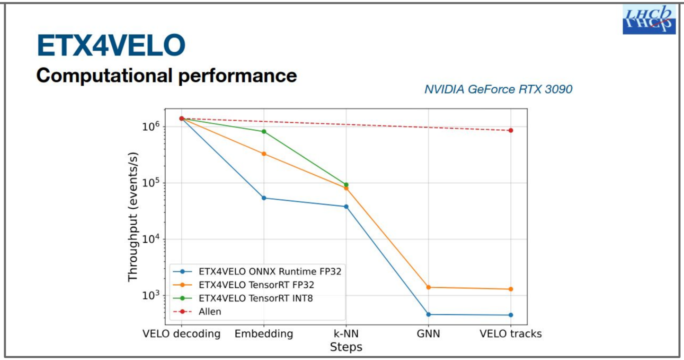
- **All need better infrastructure, just like online** (more libraries/standardization/testing/…)

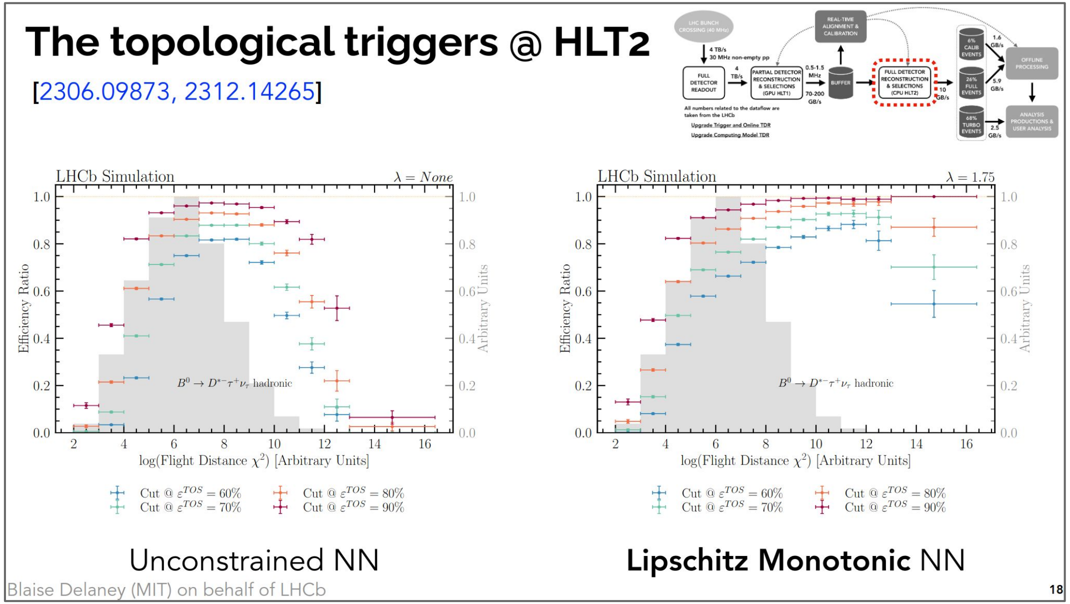


ML R&D at LHCb

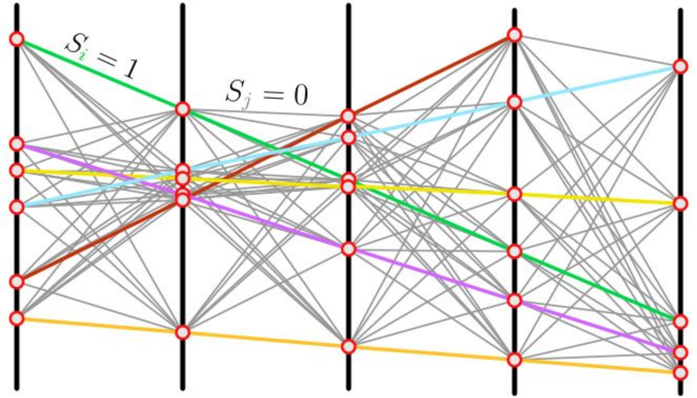
- Some experience so far at LHCb
  - Developing a tool versus looking for problems
    - *How to do it versus not how to do it?*
- By now, use of **Lipschitz monotonic NNs** used widely in production, so not R&D anymore!
  - Triggers our main physics!







Blue sky



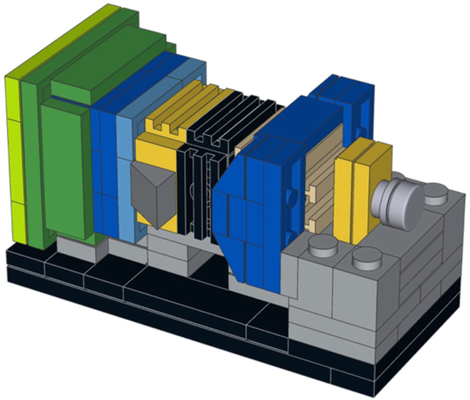
- Effort ongoing at (e.g. our own Maastricht!) using quantum algorithms (including quantum ML)
- In the context of **pattern recognition** for track reconstruction with SURF, IBM and FASTER (WP3.2)
- Goal: explore what is possible, what are limitations

- Formulate problem as an Ising-like Hamiltonian connecting 2-hit segments:
  - **Variational Quantum Eigensolver**
    finding the ground state of |Segments on or off>
  - **Variational Quantum Linear Solver**
    solving a linearized set of equations (based on [JINST 18 P11028])
  - Challenge: finding an 'ansatz' (a suitable circuit structure):
    ➔ Collaborate with UM comp.sci. dept (DACS), using q-Monte Carlo Tree Search
    ➔ Brute-force on stoomboot

- Important for qML: **embed QC with HPC**, e.g. Snellius (with EuroHPC funding?)

TLDR

- **At LHCb most unique application of ML is online**
  - **Fast inference** is crucial and main driver op development
    - Usually custom inference is fastest,
      but providing a good API to training libraries is essential!
    - But also trying external inference libraries!
- In general **developing infrastructure to improve maintainability**
  - Cannot overstate the importance of
    - Testing, pipelines, model storage, …
  - Both for CPU and GPU applications!
  - Making R&D easier
- Other applications in e.g. **simulation** and flavour tagging
  - Less unique? but very useful nonetheless
- **R&D and blue sky** approached tried and ongoing
  - **Lipschitz monotonic NNs**, can highly recommend!
  - In development
    - Quantum (ML) algorithms, e.g. for tracking (HL-LHC)

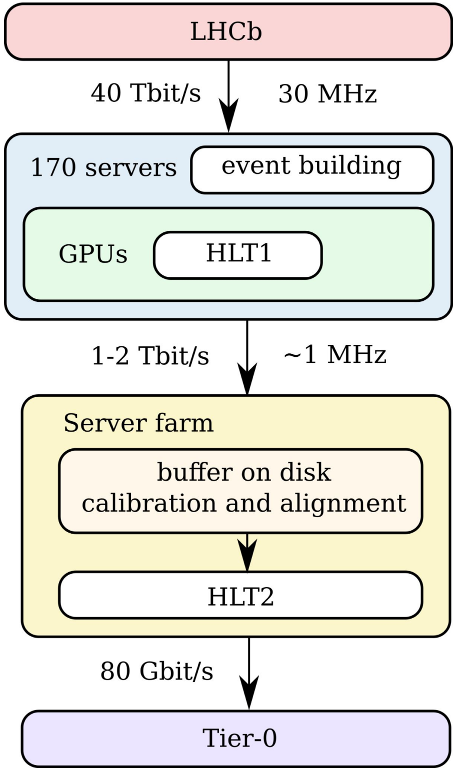


Backup

First level trigger at LHCb HLT1

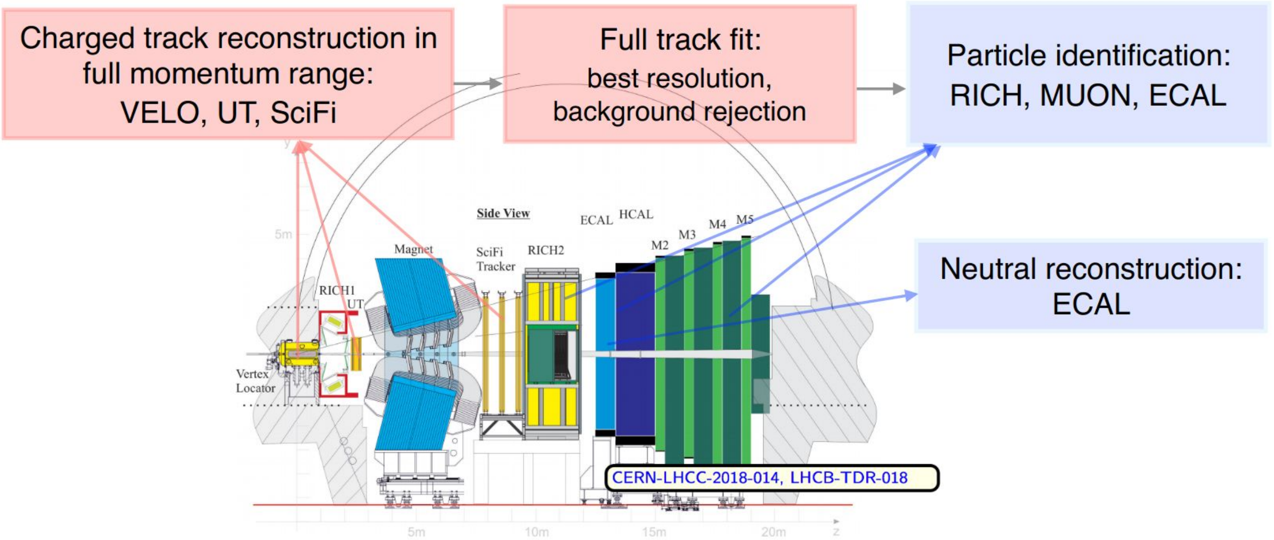
[Comput Softw Big Sci 4, 7 (2020)](#)

- **About 400 GPUs** reduce the rate of incoming data from 5 TB/s to approximately 100 GB/s
  - About **order 100 kernels** running, with the [Allen](#) software project
  - Ballpark: with 500 GPUs, **minimum** requirement is **60 kHz per GPU** for 30 MHz non-empty bunch crossings
- **Reconstruction**
  - Charged particles in tracking detectors
    - clustering, tracking, vertexing
      - Track fit and secondary vertex reconstruction
  - Muon stations / calorimeter reconstruction
    - Muon and Electron PID
      - *Including neural nets*
    - Neutrals reconstruction
- **Selection**
  - focused on ***displaced charged tracks***
    - *Including neural nets for two-track combinations*



Second and final level trigger HLT2

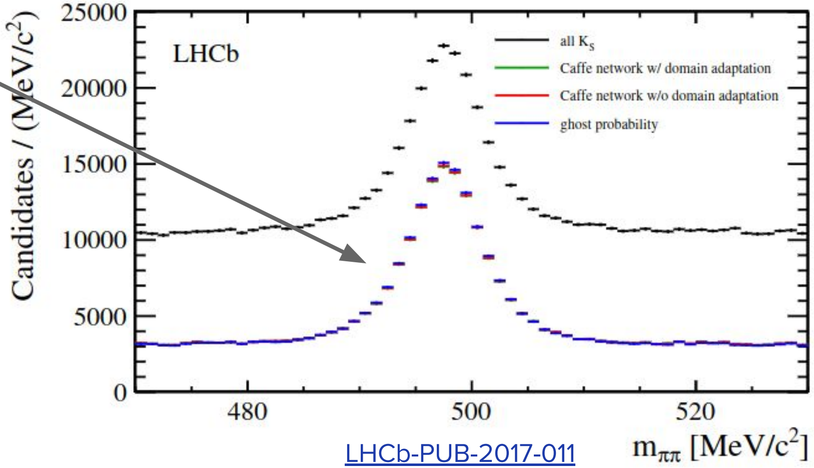
- **Full, offline-quality *(after alignment and calibration)* reconstruction** with full-quality track fit to achieve high momentum resolution, calibrated PID and vertexing **on CPUs**
  - with improvements in muon ID, electron ID and bremsstrahlung reconstruction
- **Order of 1000 selections**
  - including dedicated reconstructions, selective information persistency, …
- Ballpark: about 200 Hz throughput needed assuming about 5000 servers with 1 MHz input



Applications of ML ***in online environment of LHCb***

- **Classification of reconstructed objects** (at all levels)
  - **Reconstruction**
    - Charged tracks
      - Real vs fake (ghost rejection)
    - Type of charged tracks
      - pion / muon / electron / ...
  - **Selection** level
    - Higher level objects
      - combination of tracks coming from heavy flavour decays
    - Typically trained / used for **selecting specific signals** with trigger lines
  - **Typical feature counts of 10-20**
- Other tasks like ***pattern recognition*** and ***anomaly detection*** are possible and studied

Ghost rejection MLP from previous LHCb Run 2



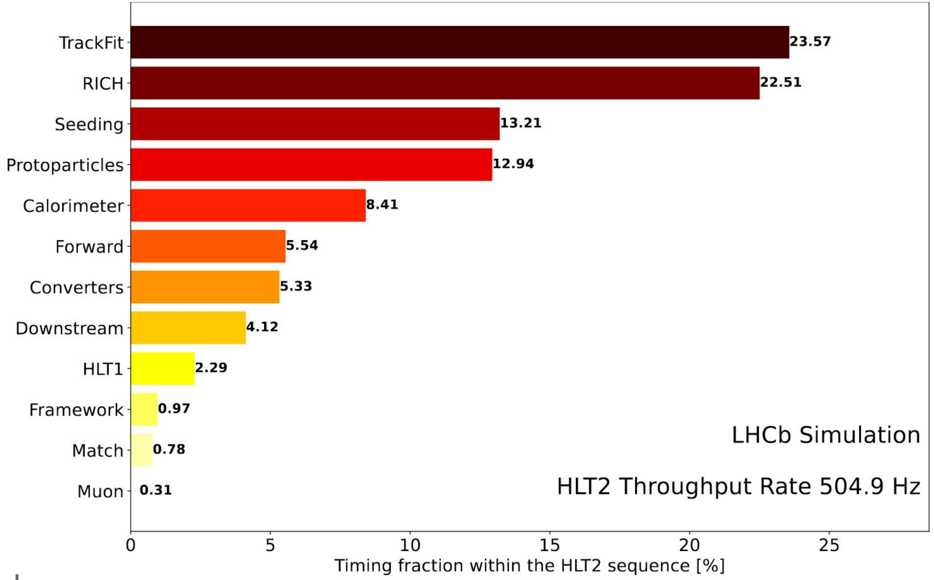
LHCb-PUB-2017-011

ML infrastructure *in online environment of LHCb*

- **Online environment needs**
  - **Most of all high speed!**
  - **Fast turn around time** of training and deployment, ...
  - **Common tools / standardization**
    - avoid customization / hard coded solutions as much as possible
    - improve maintainability and ease of use
  - **Production level code needs a lot of testing**
    - Run ML pipelines in CI/CD (Gitlab/Jenkins)
      - Also for fast turnaround time!
- **Most needed in HLT2 (CPU)**
  - Most applications, most interactions with 'users'
  - **First focus on fastest algorithms, also have simplest models!**
- But in the **future more emphasize on general libraries and GPUs**, developments ongoing
  - More challenging setup with demands on GPU/CPU compatible libraries and speed

HLT2 (CPU) throughput

- **Significant speed improvements** have been achieved using
  - **Multithreading / vectorization** also in this CPU-based software
  - *Structure-of-Arrays*
    - Reduce memory usage
  - Parallelization with SIMD
    - Single Instruction/Multiple Data [JINST 15 (2020) 06, P06018]
  - Smarter, more selective algorithms
    - Pre-select on input of time-intensive algorithms
  - Mainly used in reconstruction sequences
    - See reconstruction throughput breakdown on the right before speed ups



- **Developed new ML inference infrastructure to fully make use of that**

Testing, pipelines and experience in production

- **General aims achieved** for HLT2 (CPU) infrastructure
  - **Speeding up main classifiers** (roughly 10% of reco timing) ✅
    - **factor 2 - 3**
  - **Separate / fully optimized inference from training** ✅
  - **Maintained training pipeline** ✅

- **Running in production since start 2024** for HLT2 (CPU) infrastructure
  - **Already used for fast retraining due to online needs**
    - *Retraining within a day,*
      cross-checked / released / deployed within a few days
  - Multiple developers picked it up and are expanding it
    - Feedback so far is that it's easy to use and expand

- **General aims achieved**
  - **Fast turnaround time** ✅
  - **Ease of use** ✅

GitLab

Jenkins

Training of model

- Provided in LHCb!4305 and Moore!2768
  - **PyTorch interface**
- This needs careful testing that these do the same
  - more on that later
- Convert model defined in header to python object
  - Using python binding (cppyy)
  - Write weights to file (json)

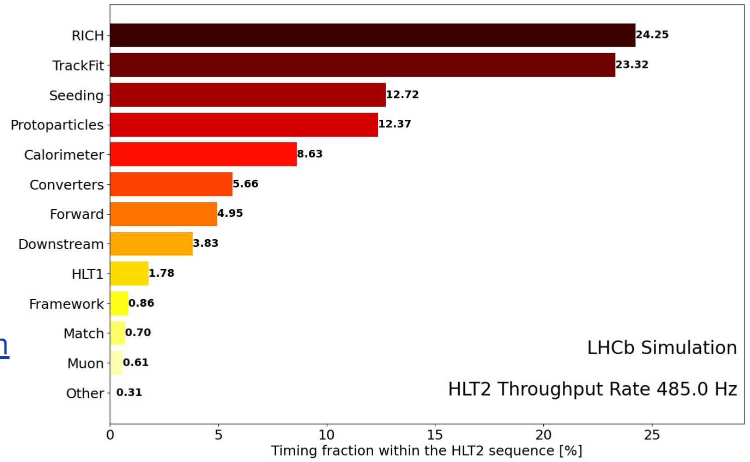
```python
from cppyy import gbl
from LHCbMath.VectorizedML import Sequence

ProbNN__Testing__Model = Sequence(gbl.ProbNN.Testing.Model())
```

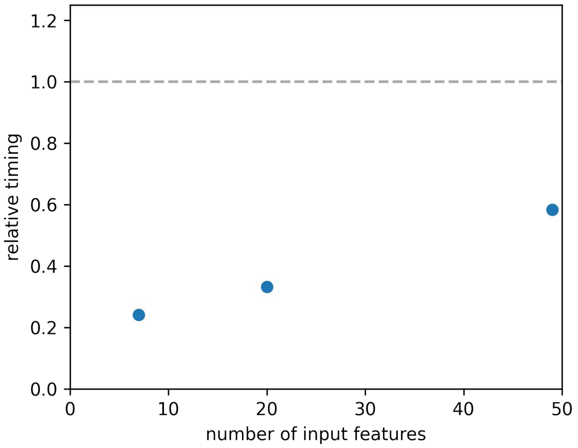
- Training script with PyTorch provided
  - See Hlt/RecoConf/options/hlt2_globalpid_training.py

```python
import torch
from torch import nn


class Sequence(nn.Module):
    def __init__(self, model):
        super().__init__()
        mod = model.model()
        # build layers
        mystack = []

        def get_torch_layer(layer):
            tlayer = None
            layer_type = layer.name()[:-2]
            if layer_type == "Linear":
                tlayer = nn.Linear(layer.nInputs(), layer.nOutputs())
            else:
                layer_dict = {"Sigmoid": nn.Sigmoid, "ReLU": nn.ReLU}
                tlayer = layer_dict[layer_type]()
            return tlayer

        from collections import OrderedDict
        for i in range(mod.nLayers()):
            layer = mod.get_layer(i)
            mystack.append((layer.name(), get_torch_layer(layer)))
        self._stack = nn.Sequential(OrderedDict(mystack))
        # save feature names
        feats = model.features()
        self._features = [feats.name(i) for i in range(feats.size())]

    def forward(self, x):
        return self._stack(x)
```

Speed tests

- Speed of *ChargedProtoParticleMaker* relative to *master*
  - **7% of reconstruction sequence** (see [here](#))!
  - dominated by ProbNN calculations
- Current models in *master*
  - TVMVA trained ➔ hard coded evaluation using [TMV_utils.h](#)
  - between 47-49 input features
    - Basically ‘all’ ProtoParticle info (incl. duplication)
  - up to two hidden layers (between 50 and 70 neurons)
- **New SIMD models**
  - Large: 49 inputs
    - Two hidden layers (58 / 68 neurons resp.)
    - Fairest speed comparison
  - Pruned: 20 inputs
    - Two hidden layers (30 / 35 neurons resp.)
    - **Realistic / reasonable scenario**
      - **About 3 times faster!**
  - Small: 7 inputs
    - Two hidden layers (12 / 12 neurons resp.)
    - Already very good performance!





Testing pipelines

- QMT testing of whole infrastructure, see [Moore!2768]
  - **Inference** (compilation + runtime) and inference / training closure (model evaluation through trainer)
    - Does the model produce the same result (reference)
    - Make sure (PyTorch) Python model is the same as C++ version
  - **Training**: data + model saving / loading
    - Two tests for
      - data
      - training
    - Make sure the full pipeline with training works

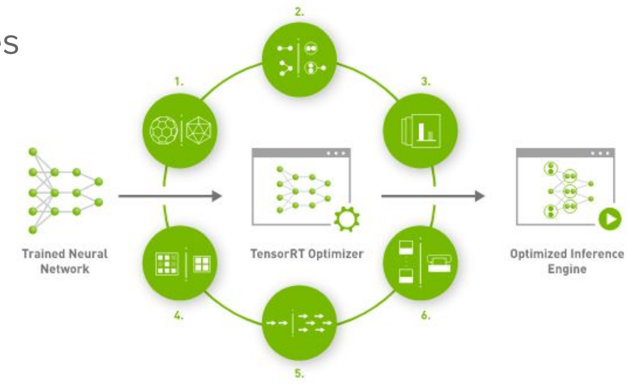
```
Defined model with features: ['EcalPIDe', 'ElectronShowerDLL', 'BremPIDe', 'HcalPIDe', 'RichDLLe', 'TrackChi2PerDoF', 'TrackGhostProb']
model coverted to PyTorch: Sequence(
  (_stack): Sequential(
    (Linear_0): Linear(in_features=7, out_features=12, bias=True)
    (ReLU_1): ReLU()
    (Linear_2): Linear(in_features=12, out_features=12, bias=True)
    (ReLU_3): ReLU()
    (Linear_4): Linear(in_features=12, out_features=1, bias=True)
    (Sigmoid_5): Sigmoid()
  )
)
difference between model and ref (prob; default) is -0.0000 +/- 0.0001
difference between model and ref (DLL)           is -0.00 +/- 0.00
AUC of prediction is 0.99650 and of reference is 0.99650
```

General libraries for ML inference in HLT1 (GPU)

- ***Flexibility, maintainability***
  - Hard/hand-coded ML inference is not flexible / not great to maintain
  - Platform to load **standardized ML-model data format**: **ONNX**
    - Supported by many (if not most) training software
    - **At CPU (HLT2) level being integrated with *ONNXRuntime***

ONNX

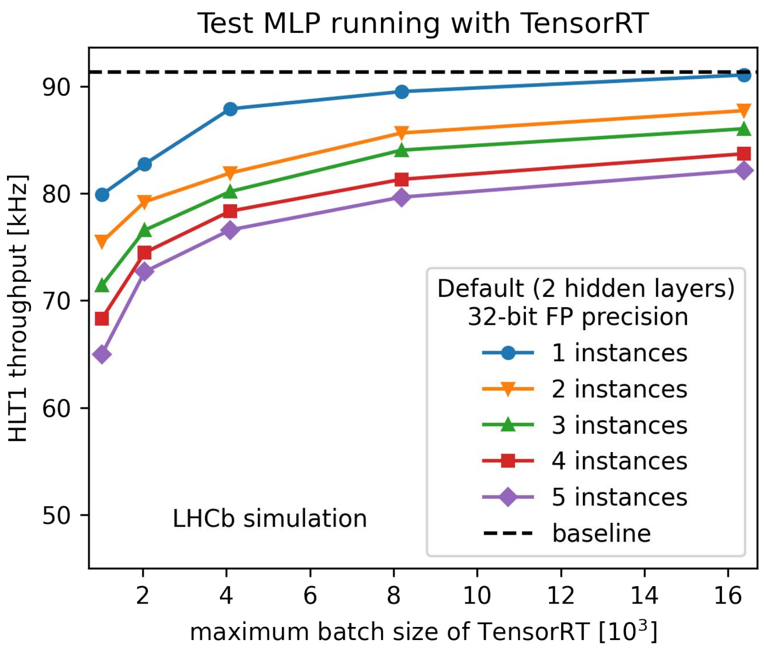
- Providing these features with inference on **GPU**
  - LHCb uses NVIDIA RTX A5000
  - **TensorRT** [link] from NVIDIA provides
    - Fast-inference platform / SDK
    - ONNX files can be read by it
    - Optimization possible within package, like quantization



1. **Weight & Activation Precision Calibration**
   Maximizes throughput by quantizing models to INT8 while preserving accuracy
2. **Layer & Tensor Fusion**
   Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel
3. **Kernel Auto-Tuning**
   Selects best data layers and algorithms based on target GPU platform
4. **Dynamic Tensor Memory**
   Minimizes memory footprint and re-uses memory for tensors efficiently
5. **Multi-Stream Execution**
   Scalable design to process multiple input streams in parallel
6. **Time Fusion**
   Optimizes recurrent neural networks over time steps with dynamically generated kernels

Throughput impact of TensorRT inference

- The **baseline model** tested with respect to TensorRT **batch size**
    - **Kernel overhead is main bottleneck**
        - These MLPs are small
- At high batch size it seems getting **feasible to run a few copies** of such neural nets!



Flavour tagging