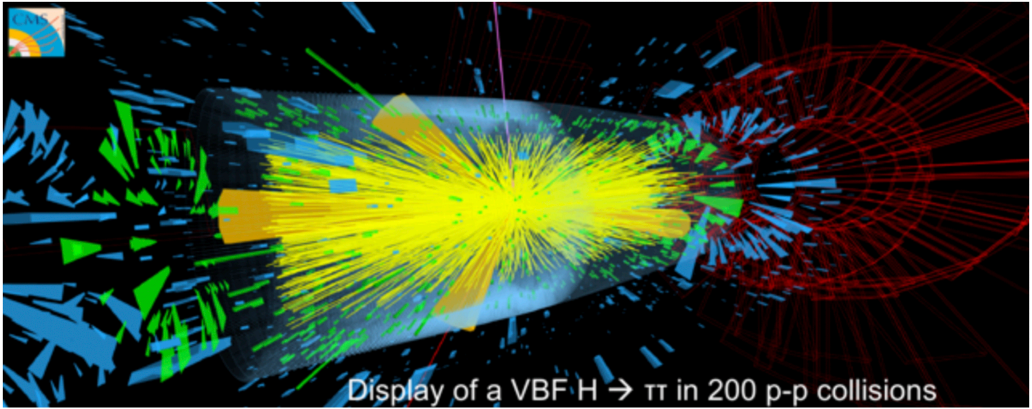
Regression in High Granularity Calorimeters at CMS

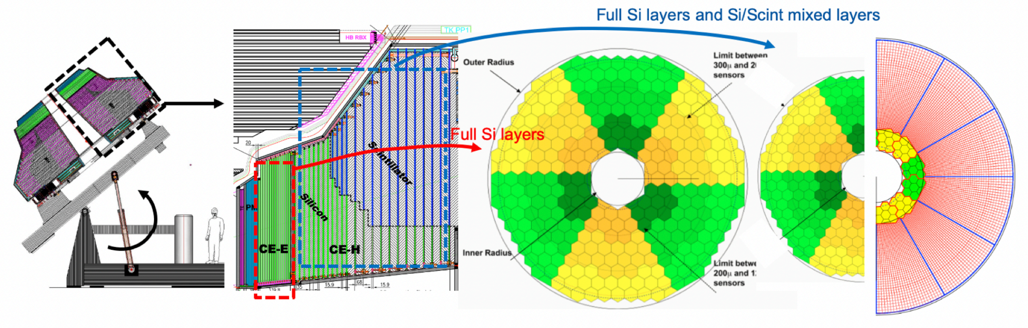
Benedikt Maier (CERN) – Jan 26, 2022

High Luminosity LHC – a pileup



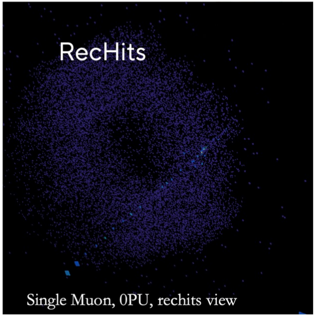
200 *simultaneous* pp collisions

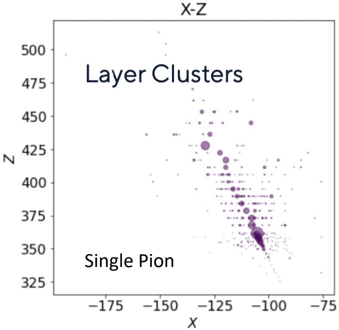
HGCAL at High Luminosity-LHC

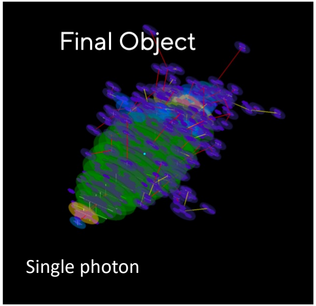


- At the end of this decade: High granularity calorimeter as forward ($1.5 < |\eta| < 3.0$) instrumentation at CMS
- Hexagonal silicon wafers in high-radiation region, scintillating tiles in low-radiation region
- Totaling about 6M channels: Needed to reject pileup contributions at HL-LHC
- Development in terms of hardware, electronics simulation, reconstruction algorithms happening now

Rechits $\rightarrow$ layer clusters $\rightarrow$ physics objects



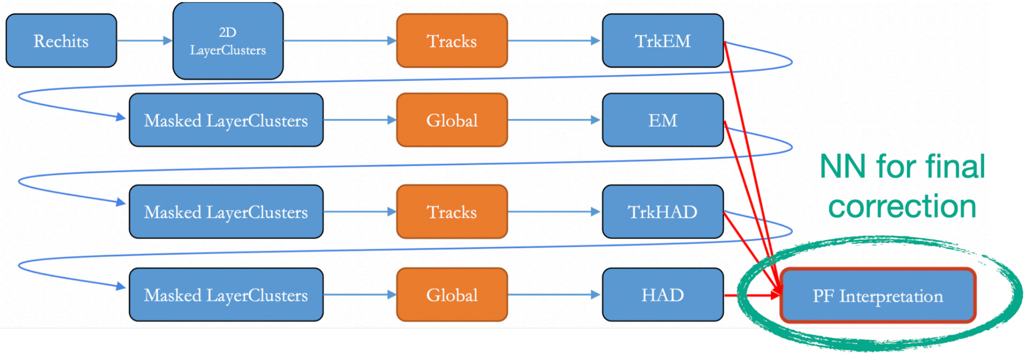




2D clustering algorithm

pattern recognition

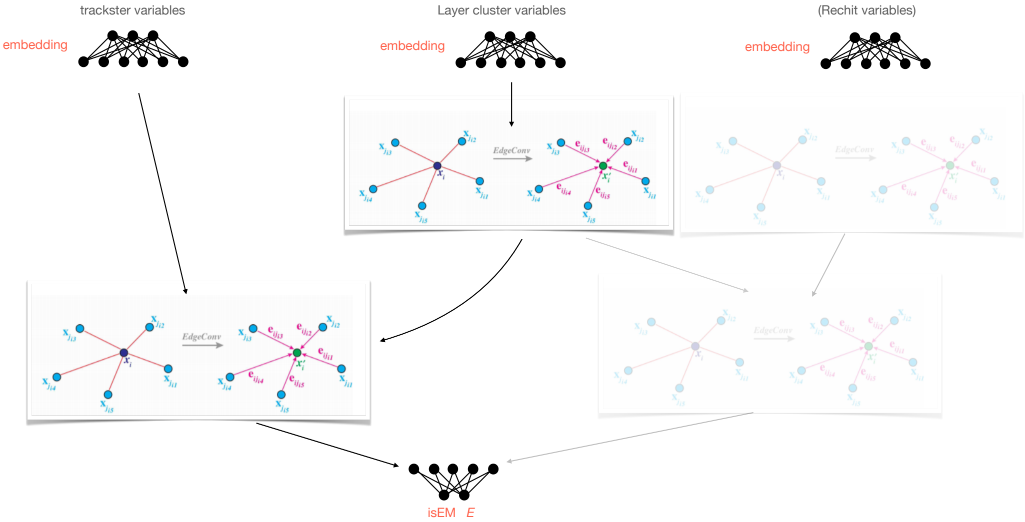
NNs for object correction



Particle energy and ID regression

- Estimation of energy and PID with a graph-based neural network
  - Great success for graph-based architectures in particle physics (ParticleNet, etc)
  - Might work here as well, as layerclusters and rechits can be seen as point clouds, on which graphNNs excel for segmentation and classification tasks
- For now: no assessment of time and memory performance related to different architectures
- But there are lessons learned already for speed-ups and reduced memory footprint
  - Static graphs vs. dynamic, etc.
  - Trying similar, yet slightly different architectures

Architecture



Training on subMIT

```
Universe      = vanilla
               executable = singularity_hgcal.sh
               should_transfer_files = YES
               transfer_input_files = train_hgcal.sh, train.yaml
               transfer_output_files = ""
               GetEnv       = True
               input = /dev/null
               output = /work/submit/bmaier/hgcal/reg/$(Cluster)_$(Process).out
               error = /work/submit/bmaier/hgcal/reg/$(Cluster)_$(Process).err
               log = /work/submit/bmaier/hgcal/reg/$(Cluster)_$(Process).log
               Requirements = BOSCOGroup == "bosco_cms" && BOSCOCluster == "ce03.cmsaf.mit.edu"
               request_gpus = 2
               arguments = $(Process)
               OnExitHold = ( ExitBySignal == true ) || ( ExitCode != 0 )
               queue 1
```

- The input samples have been generated on the CPUs in the subMIT cluster and are stored on the large storage at T2
- Paths to input and output folders are defined in train.yaml
- If you don’t have storage on the Tier-2 because you’re not a CMS user, ship your input data with the job – there is a 100 Gbs link for a reason

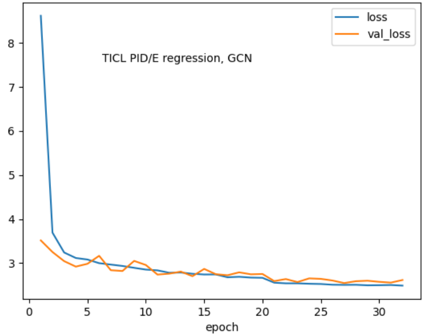
Training on subMIT

```
echo "Start running."

# Needed to run singularity+docker from within condor/bosco
unset XDG_RUNTIME_DIR

# Prepare training config and clone PUMA code
curpath=$PWD
sed -i "s|XXX|$curpath|g" train.yaml

# Getting ready for singularity
echo "About to enter the singularity container. This is the content of the current folder: "
pwd
ls -l
SINGULARITYENV_CUDA_VISIBLE_DEVICES=0,1,2,3 singularity exec --nv --bind /mnt/hadoop/cms/store/user/bmaier/:/mnt/hadoop/cms/st\
ore/user/bmaier/ --bind $PWD:$PWD docker://benediktmaier/torch-geometric /bin/sh $curpath/train_regression.sh

echo "Done."
```

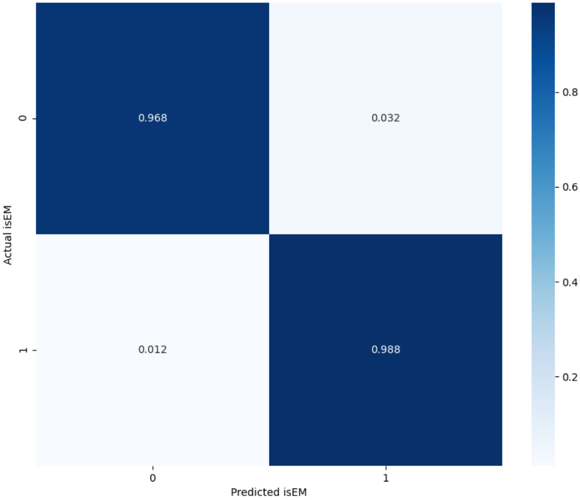
- The input samples have been generated on CPUs in the subMIT cluster and are stored on the large storage at T2
- Input data O(10 GB)
- Paths to input and output folders are defined in train.yaml
- Specifically for torch_geometric, it is now also available on LCG: source /cvmfs/sft-nightlies.cern.ch/lcg/views/dev4cuda/latest/x86_64-centos7-gcc8-opt/setup.sh

Training performance

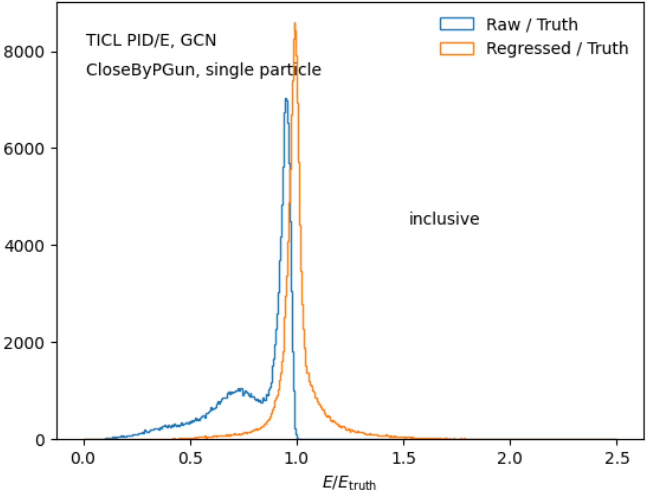
- Using single-particle samples, shot with CloseByParticleGun
- Training sample has approx. 160,000 particles (80k e/γ, 80k pion/kaon)
- Energy range between 5 and 500 GeV (flat)
- Batch size 150 particles, training for 30+ epochs
- Simple MSE loss
  - E and PID on equal footing



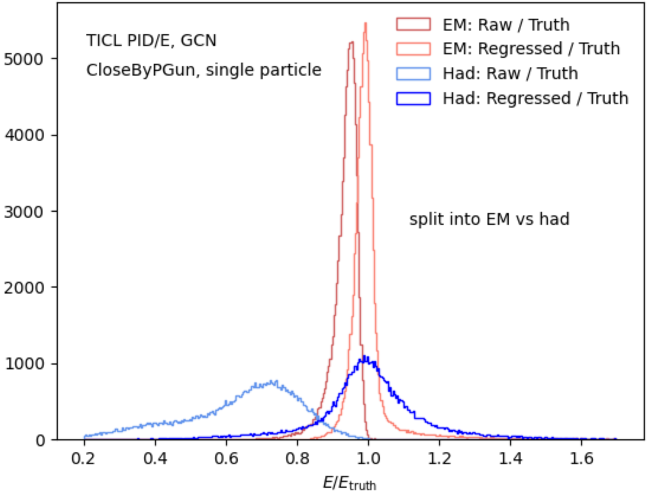
PID regression



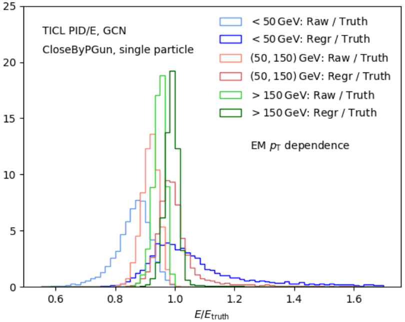
Energy regression

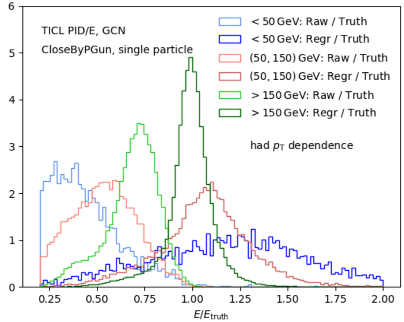


Energy regression



Energy regression





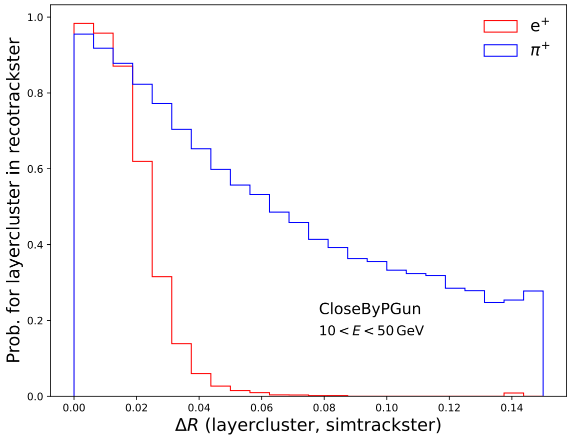
- As expected, network having a harder time the smaller the shower energy
- Currently under investigation: exploring different training architectures ($\rightarrow$ memory), different pattern recognition algorithms, ...

Summary & Outlook

- Using subMIT GPU resources to train graph neural networks in object reconstruction tasks for HL-LHC applications
- Given that the GPUs are on a cluster that is historically a CMS cluster, this is perfect for my usecase (using CMS data as input, etc)
- But the ingredients – like working with containers or mounted libraries – are enabling anyone to use those the GPU resources behind subMIT efficiently

Backup

LC efficiency



- If energy/LCs are missing from the trackster, it can mostly have three reasons
  - The LC collection was not inclusive enough to begin with
  - The algorithm is not tuned to pick up everything
    - Maybe for good reasons
  - LCs are being masked from early iterations and are not available

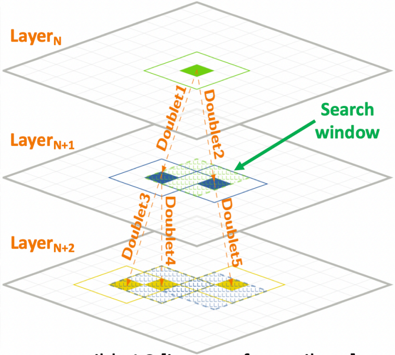
CA 1

Pattern recognition using CA

- 1st effort [used for HLT TDR]: based on “Cellular Automaton” method
  [ref: Berto, Francesco and Tagliabue, Jacopo, "Cellular Automata", *The Stanford Encyclopedia of Philosophy* (Fall 2017 Edition)]
  - A simple and fast approach; A set of rules repeated across LCs

Step 1: “Doublet” creation

- For each 2D layer cluster in layer *N*, open a search window in layer *N+1*
  - Search uses a 2D histogram in η, ϕ
    - Bin size: 0.05
      [~70 (20) mm at |η|~1.6 (2.8)]
    - Search window is 3x3 (5x5) bins for |η|<2.1 (|η|>=2.1) centered on the bin in which the LC in *N* sits
  - A layer cluster in this search window will make a “**doublet**” with the original layer cluster



- NB: timing information is used to select compatible LC [i.e., not from pileup]

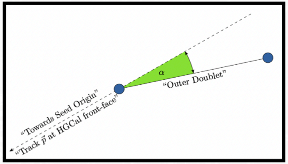
CA 2

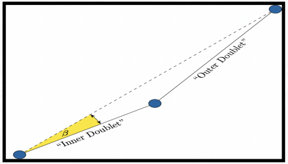
TICL: Pattern recognition (II)

- Pattern recognition (PR) based on “Cellular Automaton” method

Step 2: Doublet linking

- Doublets are linked if two angular requirements are satisfied:
  - Requirement on the direction of each doublet wrt the vertex
    - or wrt a track direction if this is a track seeded iteration
  - Requirement on the angle between the doublets





CA 3

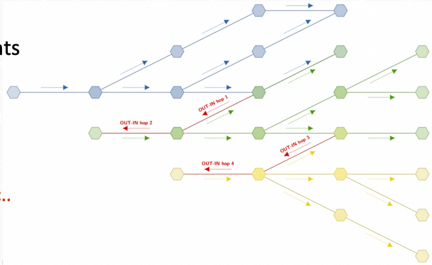
TICL: Pattern recognition (III)

- Pattern recognition (PR) based on "Cellular Automaton" method

Step 3: Trackster creation

HGCAL Depth

- Trackster formed from doublets satisfying the angular requirements
  - which may differ for different TICL iterations
- Additional requirements on:
  - #(missing layers), timing info, etc..



Collect the whole structure as a single trackster

TICLv3 iterations [i.e., HLT TDR]:
→ track-seeded EM
→ unseeded EM
→ track-seeded HAD
→ unseeded HAD
[MIP iteration currently turned-off

Architecture

- Input variables of trackster:
  - ts_raw_energy, ts_eta, ts_phi, ts_sigma1, ts_sigma2, ts_sigma3
- Input variables of layer clusters associated with trackster:
  - lc_energy, lc_eta, lc_phi, lc_nhits, lc_layer

- ... Later maybe also input variables of rechits associated with layercluster
  - rh_energy, rh_eta, rh_phi, rh_lcid
- This would allow to add new information to the training: the spatial dimensions of a layer cluster
  - But also costs in terms of memory consumption
  - Spatial dimension (“size of lc”) might be stored as an additional layer cluster attribute instead (coming soon)