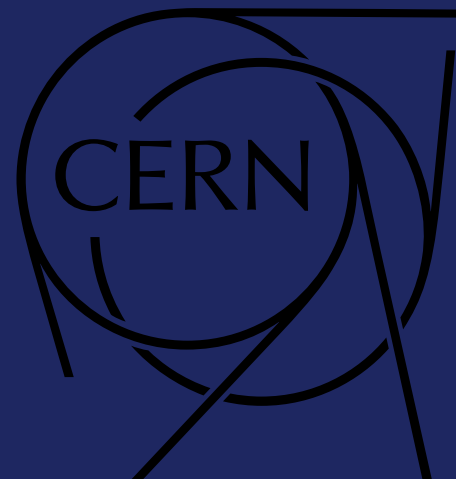
UNIVERSITÉ
DE GENÈVE

REAL-TIME INFERENCE AT THE LHC

Davide Fiacco

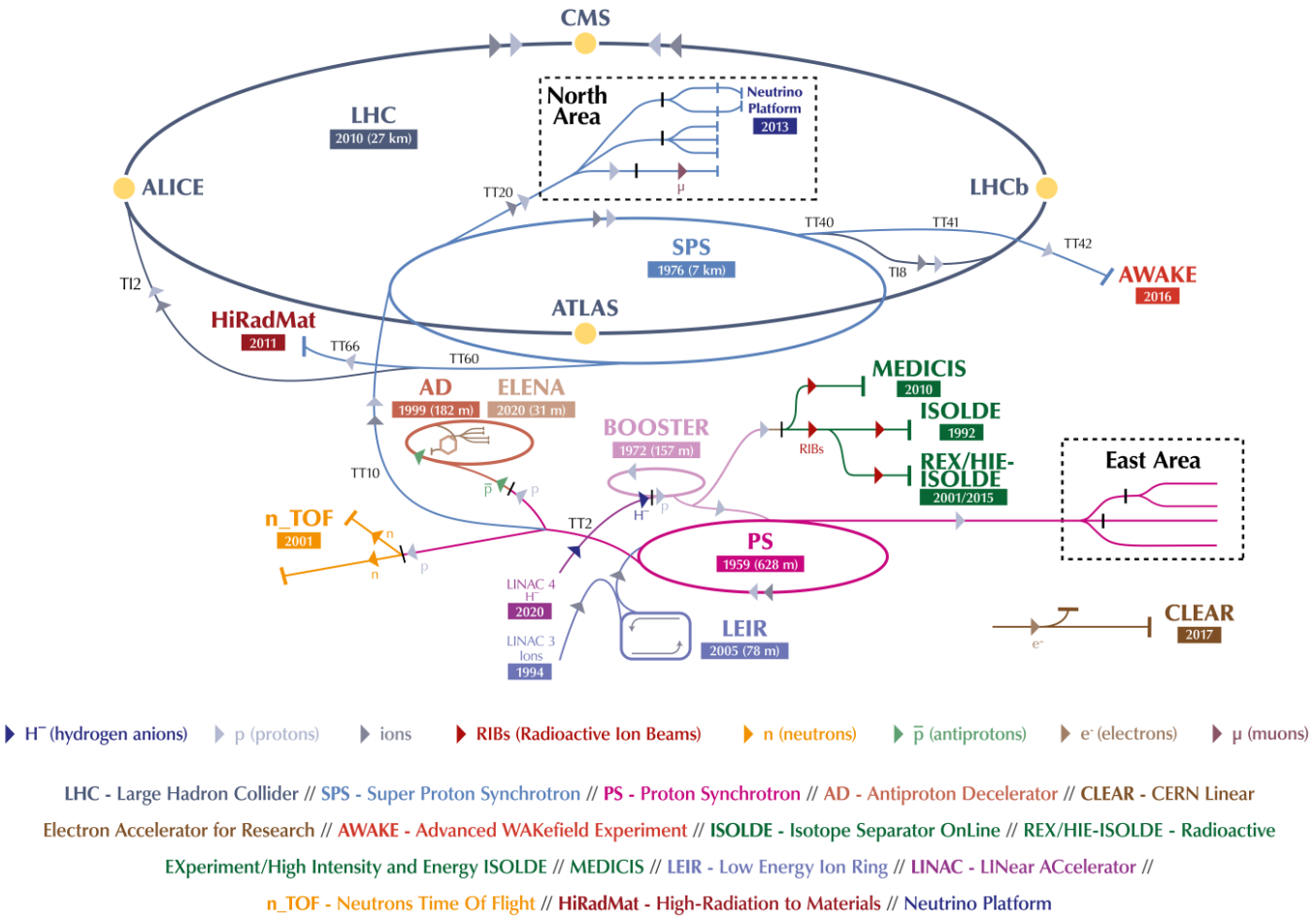
davide.fiacco@cern.ch



School on AI/ML Methods at the LHC · IBA Karachi · 16 July 2026

The Large Hadron Collider (LHC)

27 km ring, ~100 m underground
Geneva, Switzerland / France



LHC - Large Hadron Collider // SPS - Super Proton Synchrotron // PS - Proton Synchrotron // AD - Antiproton Decelerator // CLEAR - CERN Linear Electron Accelerator for Research // AWAKE - Advanced WAKefield Experiment // ISOLDE - Isotope Separator OnLine // REX/HIE-ISOLDE - Radioactive EXperiment/High Intensity and Energy ISOLDE // MEDICIS // LEIR - Low Energy Ion Ring // LINAC - LINear ACcelerator // n_TOF - Neutrons Time Of Flight // HiRadMat - High-Radiation to Materials // Neutrino Platform

Protons circulate in two beams and are steered into head-on collisions at four points around the ring — each point is one experiment.

40 MHz
proton-bunch crossing rate

13.6 TeV
collision energy (centre-of-mass)

4
large experiments: ATLAS, CMS, LHCb, ALICE

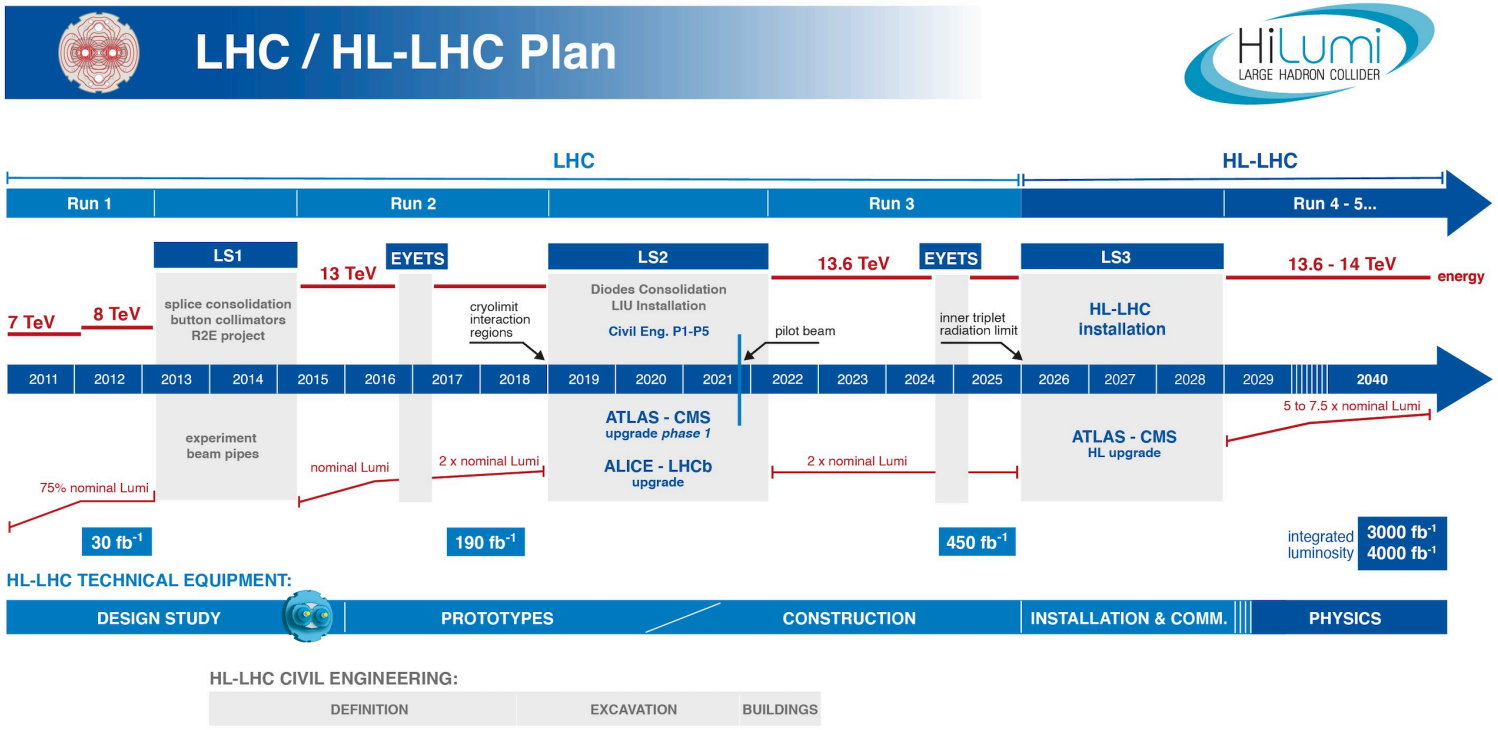
~1 PB/s
raw data generated across the ring

From LHC to HL-LHC: a more challenging environment for the DAQ

Pileup : the number of individual proton-proton collisions that land in the same bunch crossing

Instantaneous Luminosity : It's used to quantify the performance of the collider, and it's related to how densely packed and precisely aimed the two proton beams are as they cross

Starting in June 2030, HL-LHC will deliver up to 3 times today's peak luminosity and push average pileup up to ~3 times the Run value



The Trigger & Data Acquisition (TDAQ) work will be even more challenging

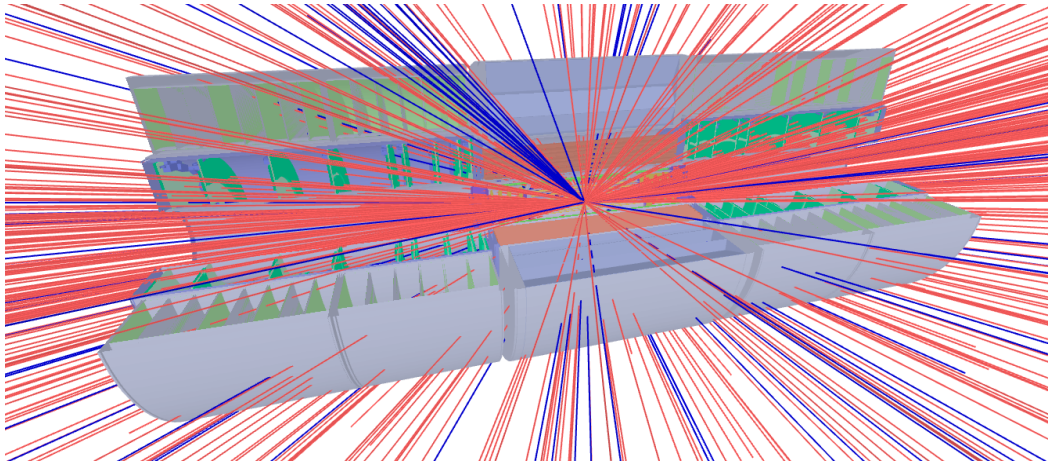
From LHC to HL-LHC: a more challenging environment for the DAQ

Pileup : the number of individual proton-proton collisions that land in the same bunch crossing

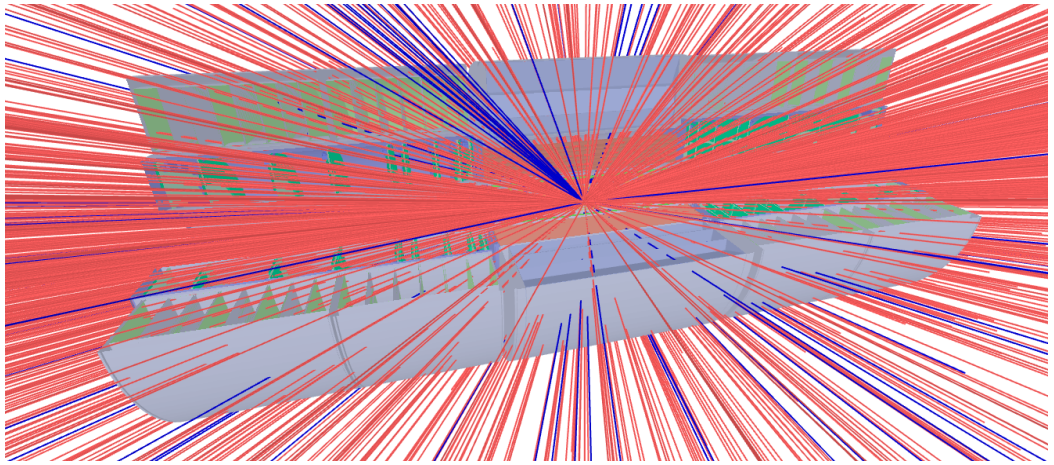
Instantaneous Luminosity : It's used to quantify the performance of the collider, and it's related to how densely packed and precisely aimed the two proton beams are as they cross

Starting in June 2030, HL-LHC will deliver up to 3 times today's peak luminosity and push average pileup up to ~3 times the Run value

Pile-up 60



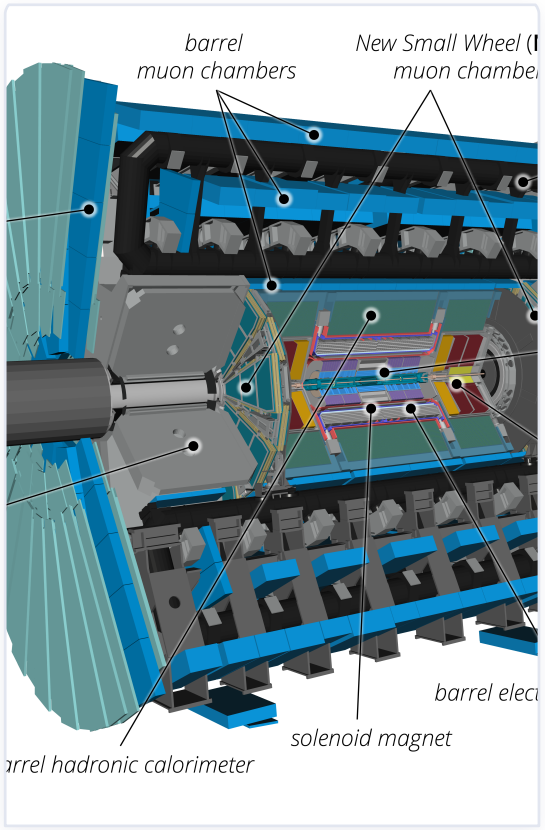
Pile-up 200

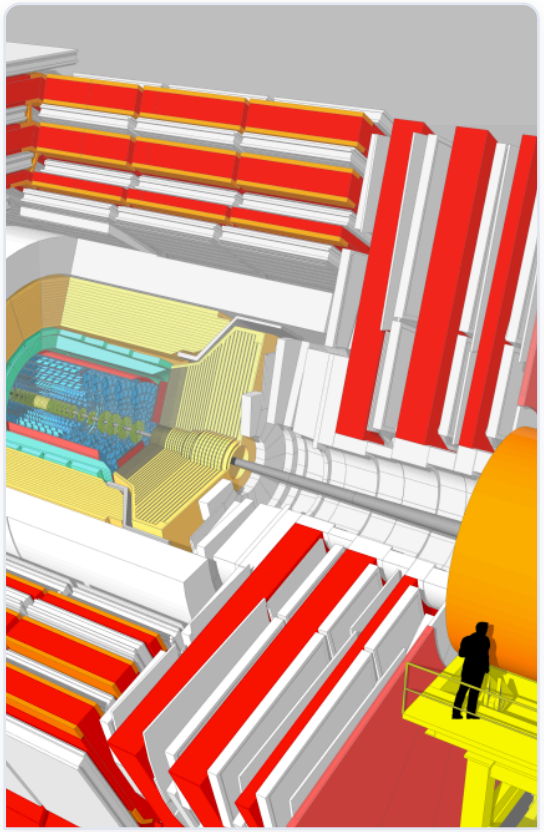


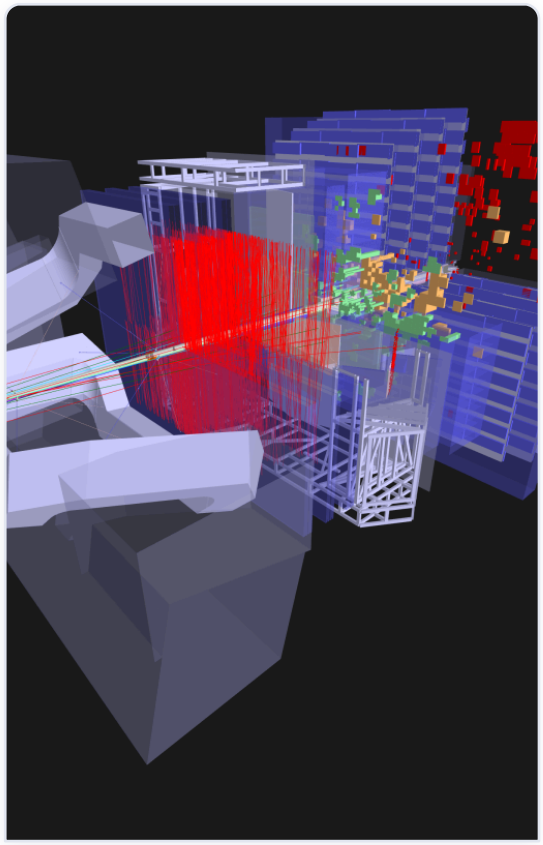
The Trigger & Data Acquisition (TDAQ) work will be even more challenging

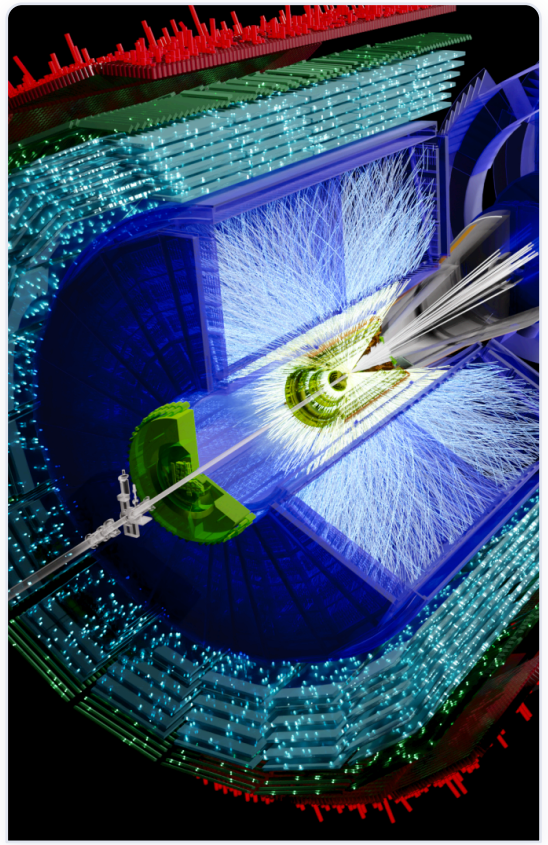
"PhyGHT: Physics-Guided HyperGraph Transformer," arXiv:2602.20475 (2026)

The 4 LHC main experiments









ATLAS

General-purpose detector.

Higgs, SUSY & BSM searches, precision SM measurements. Nearly 4π coverage.

CMS

General-purpose detector.

Same physics goals as ATLAS but independent design — a 3.8 T solenoid and compact layout.

LHCb

Forward spectrometer.

Dedicated to b- and c-hadron decays: flavour physics, CP violation, rare decays.

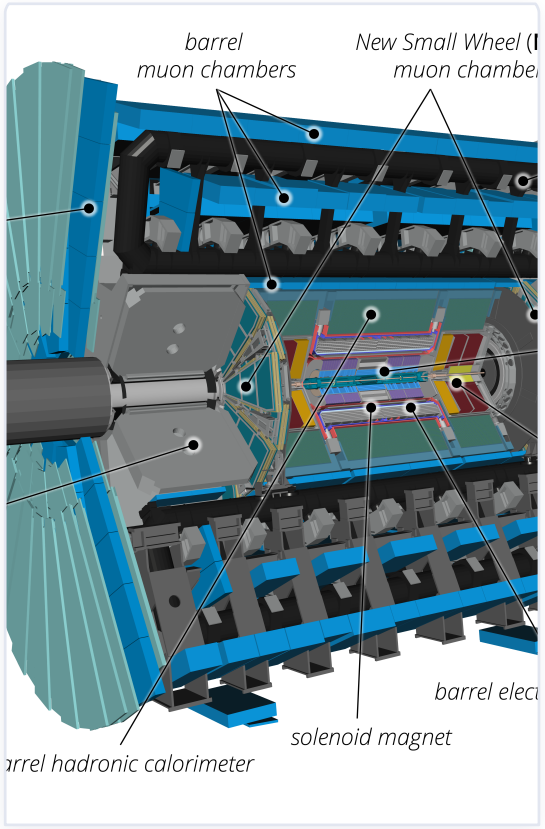
ALICE

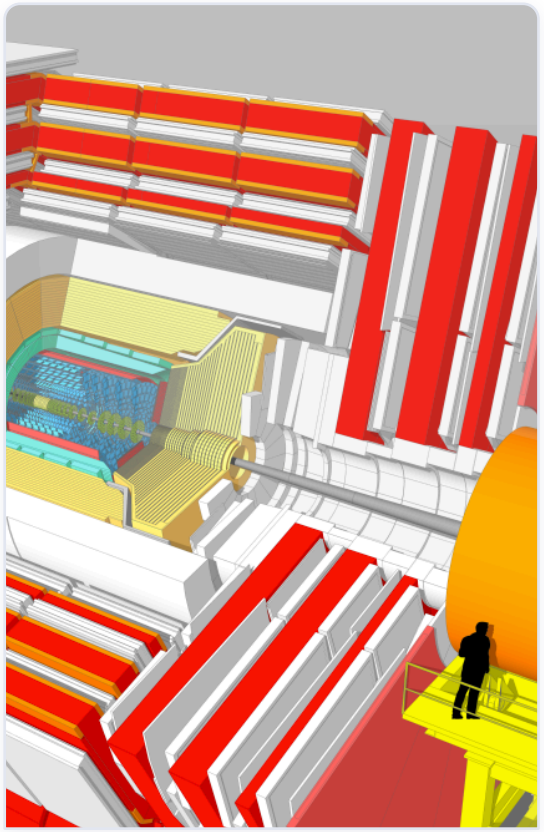
Heavy-ion detector.

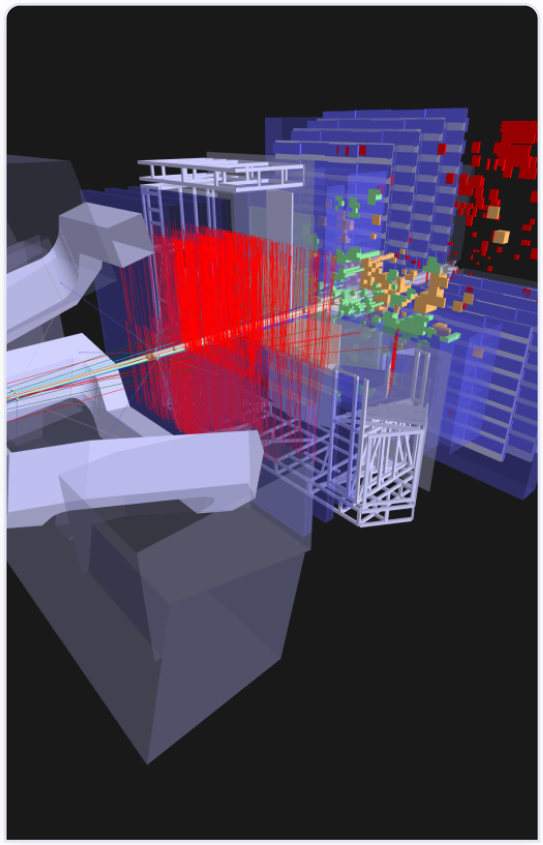
Pb-Pb collisions, studies the quark-gluon plasma — a state of matter from the early universe.

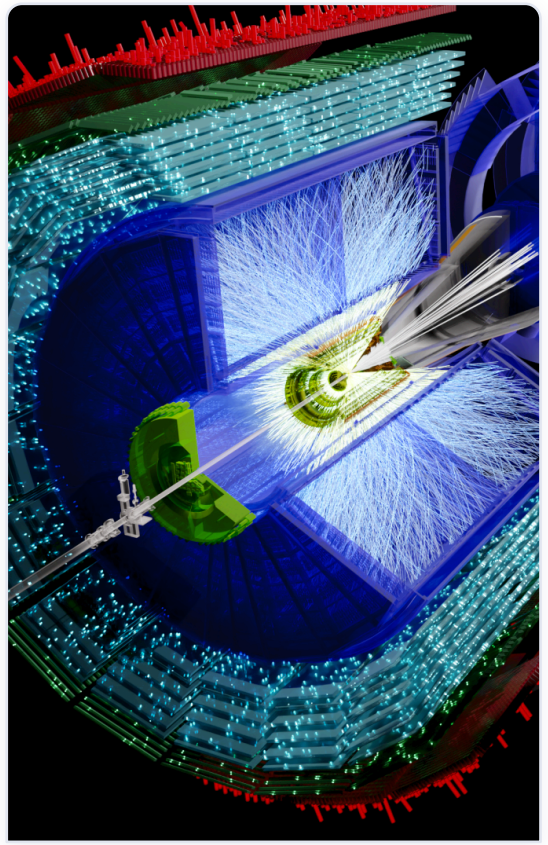
Different physics goals → different detectors → (as we'll see) different trigger strategies.

The 4 LHC main experiments









ATLAS

General-purpose detector.

Higgs, SUSY & BSM searches, precision SM measurements. Nearly 4π coverage.

CMS

General-purpose detector.

Same physics goals as ATLAS but independent design — a 3.8 T solenoid and compact layout.

LHCb

Forward spectrometer.

Dedicated to b- and c-hadron decays: flavour physics, CP violation, rare decays.

ALICE

Heavy-ion detector.

Pb-Pb collisions, studies the quark-gluon plasma — a state of matter from the early universe.

Different physics goals → different detectors → (as we'll see) different trigger strategies.

Data Streaming for ATLAS and CMS

The event size for Run 3 was (after zero-suppression, etc.) of the order of 2.1 MB, could we save all the events?

40 MHz
bunch crossings per second

×

× ~2.1 MB
raw data per collision

=

= ~84 TB/s
streaming data volume

That's on the same order of magnitude as all global internet traffic, averaged across the entire planet, every second the LHC runs.

(… and for HL-LHC will be also worst than that, because of the bigger detector occupancy each event will have a bigger size …)

You cannot store it. You cannot even read most of it off the detector. A keep-or-discard decision has to be made for every single collision: Trigger

The 2-level Trigger Strategy adopted for Run 3

STAGE 1 — LEVEL-1 (HARDWARE)

- Custom electronics: FPGAs & ASICs, not CPUs
- Sees only coarse info: calorimeter towers, muon hits
- Fixed, deterministic latency: 2.5–4 μs
- 40 MHz → ~100 kHz

STAGE 2 — HIGH-LEVEL TRIGGER (SOFTWARE)

- Commodity CPU/GPU farms
- Full detector granularity now available
- Relaxed latency: ms per event
- ~100 kHz → ~1–3 kHz to storage

1 in ~30,000

collisions is ever kept for physics analysis

(One-line teaser: LHCb and ALICE removed Stage 1 entirely for Run 3 — we'll come back on this later)

The 2-level Trigger Strategy adopted for ~~Run 3~~ HL-LHC

1 in ~3,000

collisions is ever kept for physics analysis

STAGE 1 — LEVEL-1 (HARDWARE)

- Custom electronics: FPGAs & ASICs
- Latency increased to 10 µs (12.5 µs) for ATLAS (CMS)
- 40 MHz → ~1 MHz

STAGE 2 — HIGH-LEVEL TRIGGER (SOFTWARE)

- Commodity CPU/GPU farms
- No much Latency increase thanks to the increased resources
- ~1 MHz → ~10 kHz to storage

More collisions to untangle, more data per decision, but no really much more time to do it in

Four conditions any trigger algorithm must satisfy

Whatever runs in the trigger it has to respect all the following four conditions

1 Accuracy: The decision is irreversible

There is no re-try, a rejected trigger event is gone forever, so the trigger has to be accurate

2 Latency: the trigger has to be fast

A model that is 50% more accurate but 2× slower is simply unusable if outside the latency cost

3 Occupancy: it has to fit in the available resources

At L1, the entire trained model often has to live on a single FPGA chip

4 Truncated, low-fidelity inputs

No full detector granularity, no fine tracking, the model must decide well on degraded information

Could machine learning be a good fit for triggers?

1 Exploits correlations no hand-cut ever could

A rectangular cut on a few variables can't capture the true, high-dimensional decision boundary between signal and background — a trained model can.

+

2 Adaptable to new or unanticipated signals

A fixed trigger menu only catches what physicists explicitly coded for. A model trained on generic features generalises further — and can be retrained, not redesigned, when conditions change.

+

3 Model-agnostic techniques: anomaly detection

Unsupervised models don't need a predefined signal at all — they can flag “unusual” events regardless of what makes them unusual, catching the unknown unknown.

= Yes!

Nearly a decade of results at HLT

2017 — DeepCSV live at CMS HLT

2018–2022 — DeepTau, Ringer expand across HLT

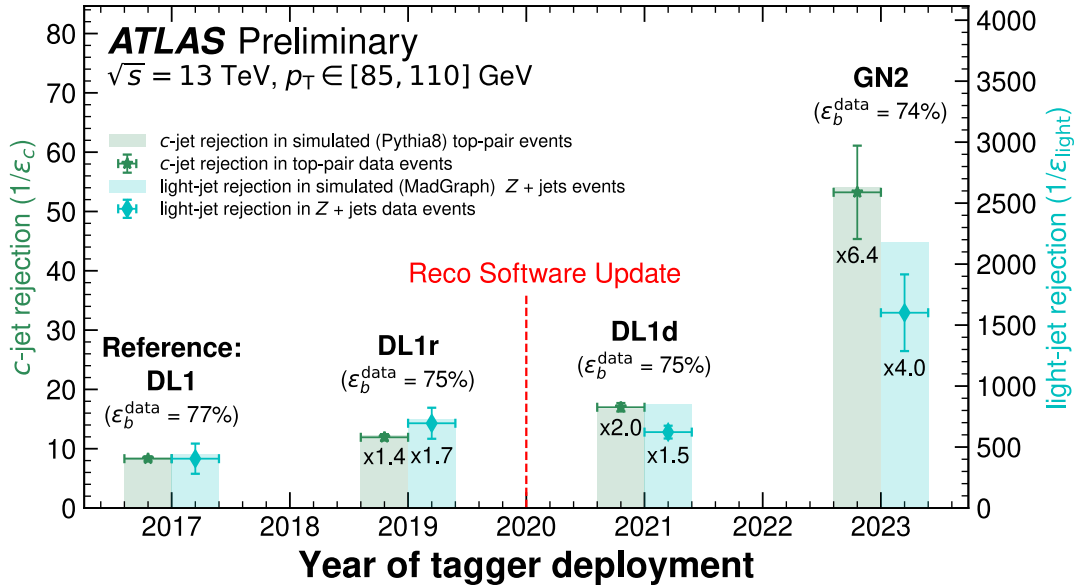
2022–now — DeepJet, ParticleNet, GN2 — Run 3

Next — push the same idea to L1/L0

"Deep NN at HLT" isn't a new idea — it's a mature, proven one. Deep neural networks have been running live in HLT selections since 2017 (CMS's DeepCSV, ATLAS's RNN b-tagging), continuously refined ever since.

\+ **4** **Cheap approximations of same offline algorithms**

"At HLT the ML models offer quick near-offline-quality decisions, inside ms time budget.



Example of ML-based b-jet tagging evolution: all these models have been also implemented at trigger level, enhancing the trigger discrimination power!

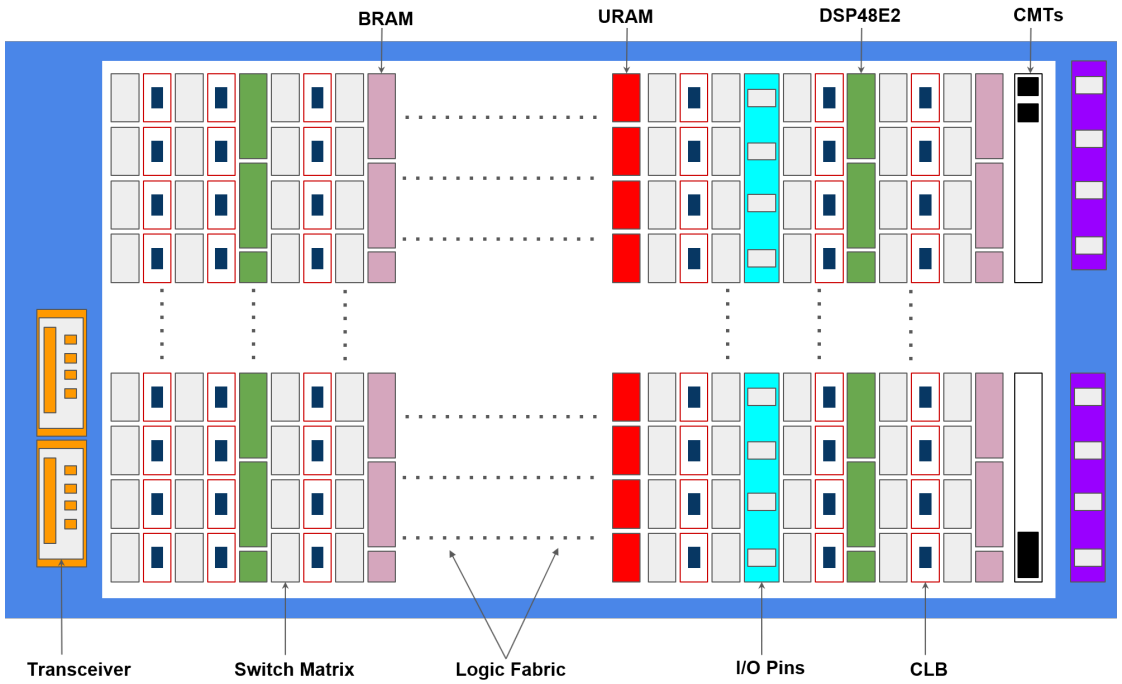
ATLAS public plot FTAG-2023-07, https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/FTAG-2023-07/

So what does “real-time” actually means at Level 1?

At HLT, “real-time” meant a few hundred milliseconds on ordinary CPU/GPU servers. At L1, it means micro-nanoseconds on a completely different kind of chip: FPGA

Field-Programmable Gate Array (FPGA)

- A chip of reconfigurable logic cells (LUTs), fast multiply blocks (DSPs), and on-chip memory (BRAM), linked by programmable wiring
- You don't write instructions for it but you configure the wiring to physically BUILD a custom circuit shaped like your algorithm
- Every logic cell fires every clock tick; it’s fully parallel, and it has a fully deterministic latency



Y. Akhauri, https://arxiv.org/pdf/2004.05898

Main FPGA components:

- Logic cell (LUT) — a tiny reconfigurable truth table
- DSP block — a hard-wired multiplier/accumulator
- BRAM — fast on-chip memory block
- Programmable interconnect — the “wiring” you configure

Fitting a model onto a chip:

Remember points 2 and 3 of a well-functioning trigger: a model has to physically fit inside a fixed number of LUTs, DSPs, and BRAM blocks of the FPGA with the required Latency

Fitting a model onto a chip: Knowledge distillation

Train a small, fast "student" model to mimic a larger "teacher", shrinking the architecture itself

After we trained the teacher with e.g. a classification task using the loss L, we retrain a small model called student combining the same loss with an extra arm that depends on the output of the teacher

$$L_{student} = \alpha \cdot L_{classif}(y,\ softmax(z_s)) + (1-\alpha) \cdot \tau^2 \cdot L_{soft}(softmax(z_t/\tau),\ softmax(z_s/\tau))$$

Where z_s and z_t are the predictions of the student and the teacher (respectively), α is used to balance the two loss terms and τ is called temperature and is used to decrease the dominance of the most probable logic, enhancing the correlation between classes

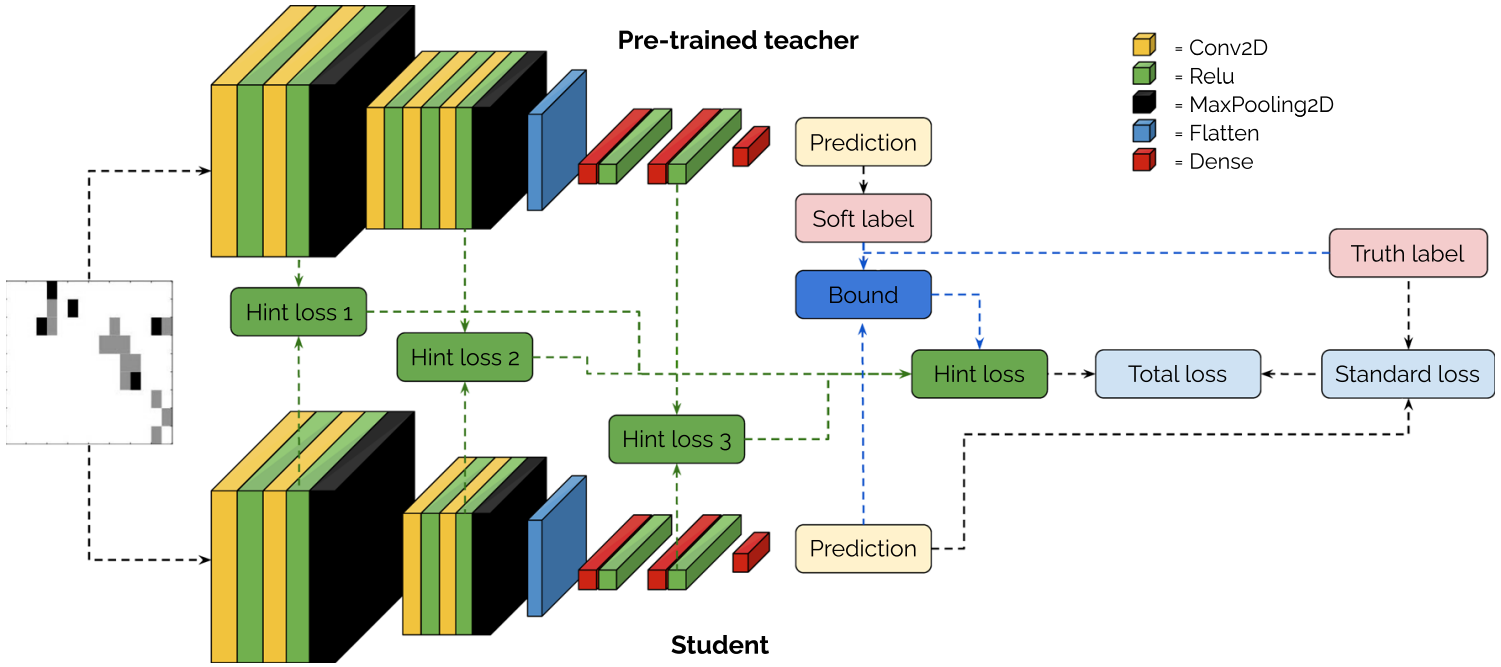
Fitting a model onto a chip: Knowledge distillation

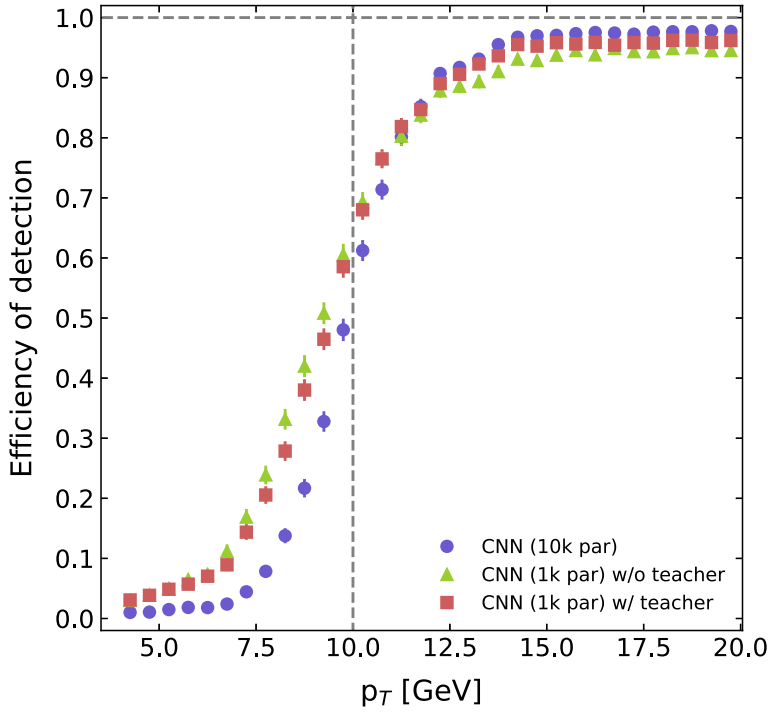
Train a small, fast "student" model to mimic a larger "teacher", shrinking the architecture itself

After we trained the teacher with e.g. a classification task using the loss L, we retrain a small model called student combining the same loss with an extra arm that depends on the output of the teacher

$$L_{student} = \alpha \cdot L_{classif}(y,\ softmax(z_s)) + (1-\alpha) \cdot \tau^2 \cdot L_{soft}(softmax(z_t/\tau),\ softmax(z_s/\tau))$$

Where z_s and z_t are the predictions of the student and the teacher (respectively), α is used to balance the two loss terms and τ is called temperature and is used to decrease the dominance of the most probable logic, enhancing the correlation between classes





S. Francescato, Eur. Phys. J. C 81, 969 (2021)

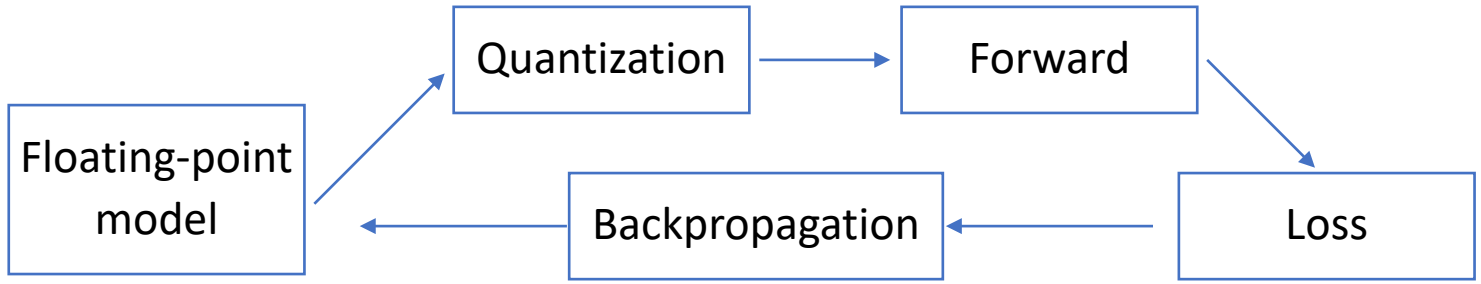
Fitting a model onto a chip: Quantisation-aware training

On FPGA we need to fix the number of bit for each weight of the model… and floating point arithmetic is typically too resource-hungry to run at trigger scale, so every weight and activation running on an FPGA has to be described with a fixed-point

The naive approach: trains the model normally at full precision and only rounds the weights afterward **(Post-Training Quantization)**

Quantization-aware training (QAT) instead simulates the rounding during training itself: the forward pass uses the low-precision, rounded weights, so the loss the network sees already reflects the quantization error during training

The network effectively learns to compensate for its own future quantization, recovering most of the accuracy PTQ would have thrown away

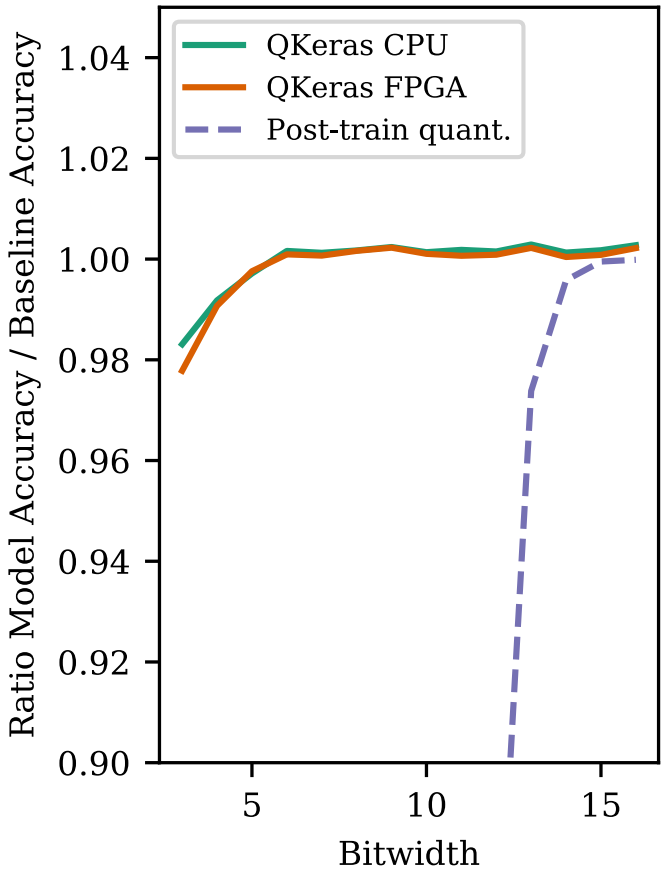


To maintain the differentiability, quantization operation in backward op. is substitute as just identity (Straight-Through Estimator, STE)

Fitting a model onto a chip: Quantisation-aware training

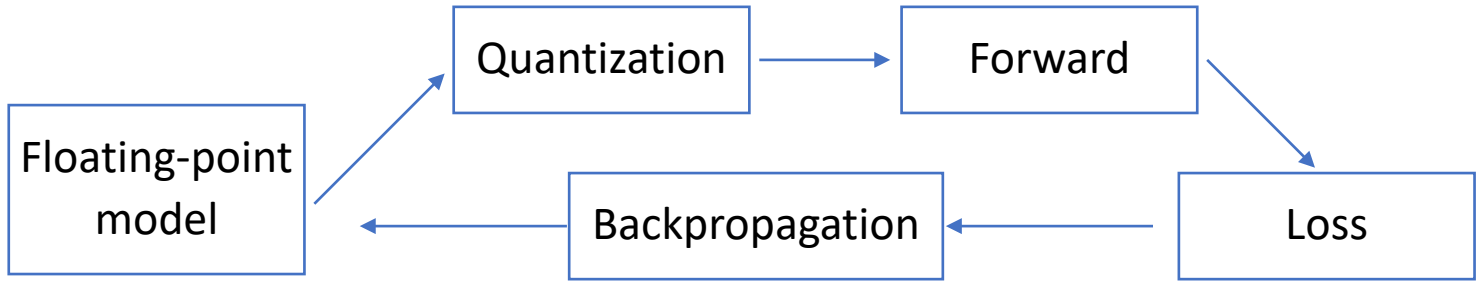
On FPGA we need to fix the number of bit for each weight of the model… and floating point arithmetic is typically too resource-hungry to run at trigger scale, so every weight and activation running on an FPGA has to be described with a fixed-point

The naive approach: trains the model normally at full precision and only rounds the weights afterward **(Post-Training Quantization)**



Quantization-aware training (QAT) instead simulates the rounding during training itself: the forward pass uses the low-precision, rounded weights, so the loss the network sees already reflects the quantization error during training

The network effectively learns to compensate for its own future quantization, recovering most of the accuracy PTQ would have thrown away



To maintain the differentiability, quantization operation in backward op. is substitute as just identity (Straight-Through Estimator, STE)

F. Fahim, B. Hawks, et al. in Proceedings of TinyML Research Symposium 2021, arXiv:2103.05579

Fitting a model onto a chip: Quantum Aware Pruning

We cancel the null weights and ignore them in the implementation

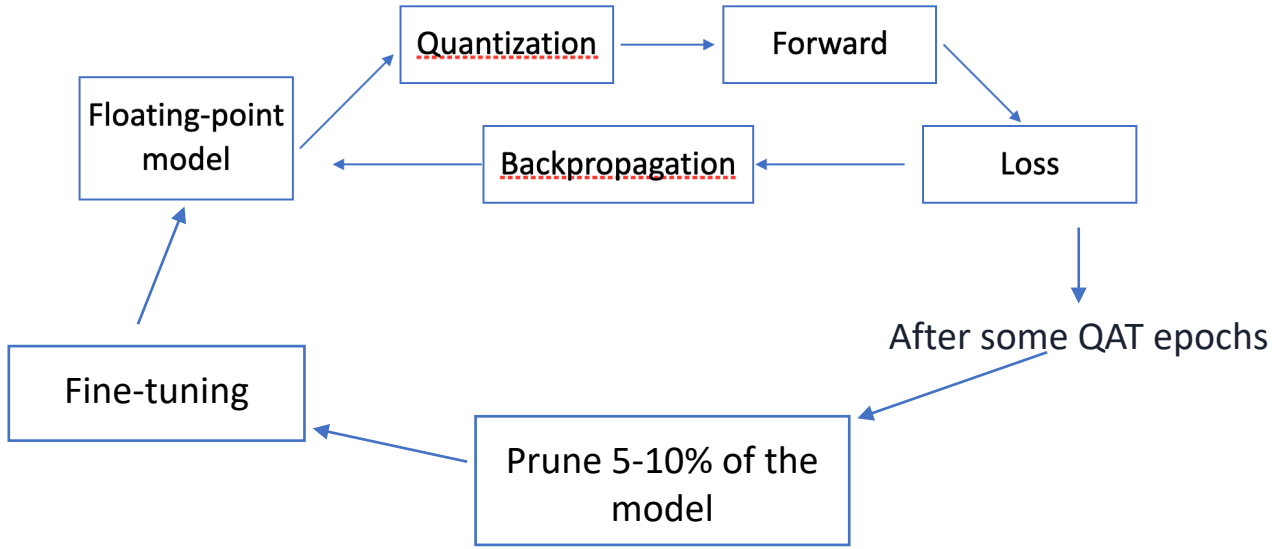
It's a very simple and effective idea, but how can we implement it in the most effective way?

Simpler: add a penalty term to the loss like $L_{penalty} = \lambda \cdot ||w||_1$ to force weights to have lower values

After the training we can order the weights depending on their abs-value and run away the less important….

But we know we need to do also quantisation, so it's better to merge the two idea to search a balance

Example:



Fitting a model onto a chip: Quantum Aware Pruning

We cancel the null weights and ignore them in the implementation

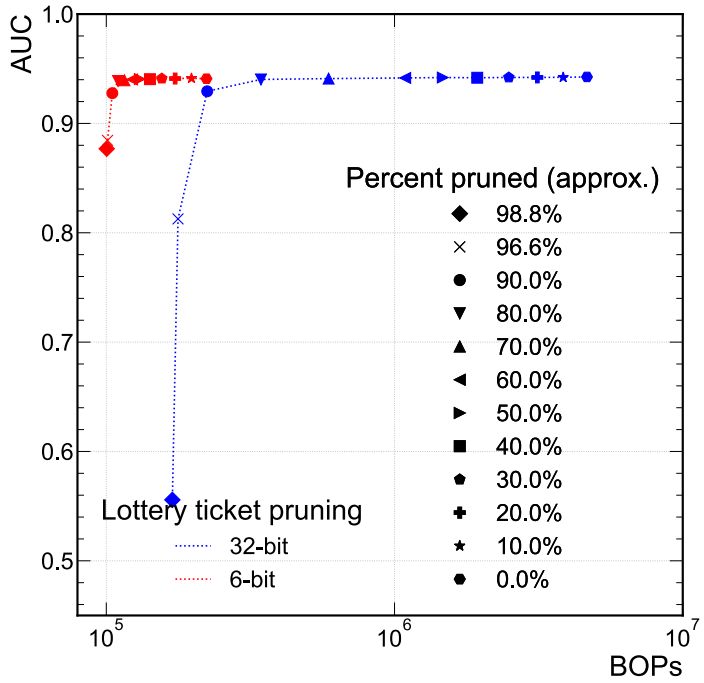
It's a very simple and effective idea, but how can we implement it in the most effective way?

Simpler: add a penalty term to the loss like $L_{penalty} = \lambda \cdot ||w||_1$ to force weights to have lower values

After the training we can order the weights depending on their abs-value and run away the less important….

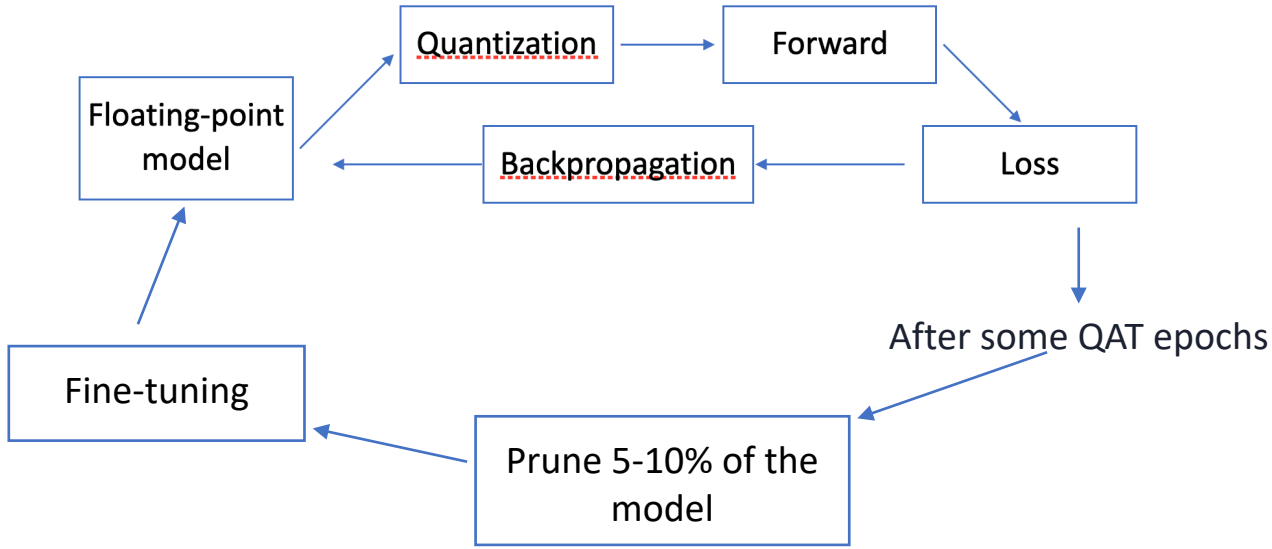
But we know we need to do also quantisation, so it's better to merge the two idea to search a balance

AUC = Area Under the Curve



BOPs = Bit Operations per layer

Example:



F. Fahim, B. Hawks, et al. in Proceedings of TinyML Research Symposium 2021, arXiv:2103.05579

Fitting a model onto a chip: High Granularity Quantization

Actually just applying QAT we decide the number of bits that each layer of the model need to use, but we could try to merge the QAT and pruning idea toghter trying to reduce both resource usage and loss

High Granularity Quantization (HGQ) the bit width itself becomes a trainable parameter, per weight (or per activation) and are not fixed by hand

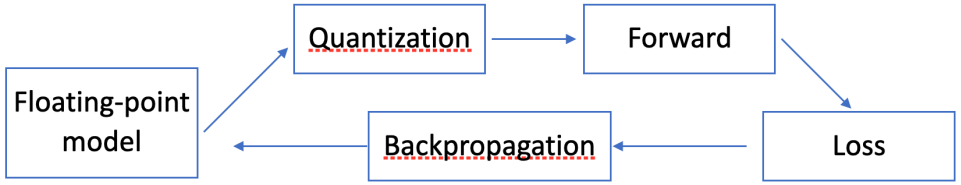
We need to add to the loss a term to contain the bit-widths of the model:

$$L_{student} = L + \lambda \cdot EBOPs(f_1, f_2, \dots) \quad \text{where} \quad EBOPs(f_1, f_2, \dots) = \Sigma_{i,j \in M} \; f_i \cdot f_j + \Sigma_{f_k, f_l \in A} \; max(f_k, f_l)$$

M is the set of multiplication operations, and A the set of sums/subtractions

And then the f_i values are used in the quantization operations of the weight inside the same loop of the QAT

Recap: quantize w_i means $w_i^Q = \dfrac{round(w_i \cdot 2^{f_i})}{2^{f_i}}$



After the training we can round f_i to be integer and then fine-tune

Fitting a model onto a chip: High Granularity Quantization

Actually just applying QAT we decide the number of bits that each layer of the model need to use, but we could try to merge the QAT and pruning idea toghter trying to reduce both resource usage and loss

High Granularity Quantization (HGQ) the bit width itself becomes a trainable parameter, per weight (or per activation) and are not fixed by hand

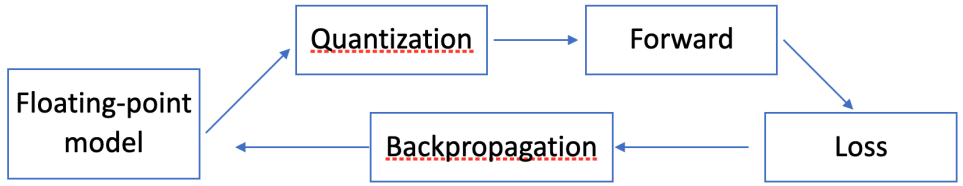
We need to add to the loss a term to contain the bit-widths of the model:

$$L_{student} = L + \lambda \cdot EBOPs(f_1, f_2, \dots) \quad \text{where} \quad EBOPs(f_1, f_2, \dots) = \Sigma_{i,j \in M} \; f_i \cdot f_j + \Sigma_{f_k, f_l \in A} \; max(f_k, f_l)$$

M is the set of multiplication operations, and A the set of sums/subtractions

And then the f_i values are used in the quantization operations of the weight inside the same loop of the QAT

Recap: quantize w_i means $w_i^Q = \dfrac{round(w_i \cdot 2^{f_i})}{2^{f_i}}$



After the training we can round f_i to be integer and then fine-tune

HLF JSC (CERNBox)

Implementation	Accuracy ↑	Latency [cycles]	LUT	DSP	FF
HGQ	74.5%	13 (20.4 ns)	3,152	0	2,941
HGQ	73.2%	11 (14.6 ns)	976	0	904
HGQ	72.4%	9 (9.9 ns)	623	0	642
QKeras [NMI'21] [22]*	74.8%	11 (55 ns)	39,782	124	8,128
QKeras [NMI'21] [22]*	72.3%	11 (55 ns)	9,149	66	1,781

C. Sun, Z. Que, et al. in Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '26), Seaside, CA, Feb 22–24, 2026, doi:10.1145/3748173.3779200

From trained model to firmware: the tool pipeline

hls4ml: converts the NN into HLS C++ (Conifer does the same for boosted decision trees) both targeting Xilinx/Intel FPGAs

HLS (High-Level Synthesis): a class of tools that automatically translates ordinary software code (typically C++) into a hardware description (RTL), letting you describe *what* the circuit should compute, instead of hand-designing it gate by gate

RTL (Register-Transfer Level): a description of the digital circuit in terms of how data moves between hardware registers each clock cycle, and what logic operations happen to it along the way (typically written in VHDL or Verilog)

Synthesis (Vivado): the process of turning an RTL description into an actual physical circuit on the chip, mapping the logic to real LUTs, DSPs, and BRAM blocks

UNIVERSITÉ DE GENÈVE

(Bonus part) Examples :

Davide Fiacco

CERN

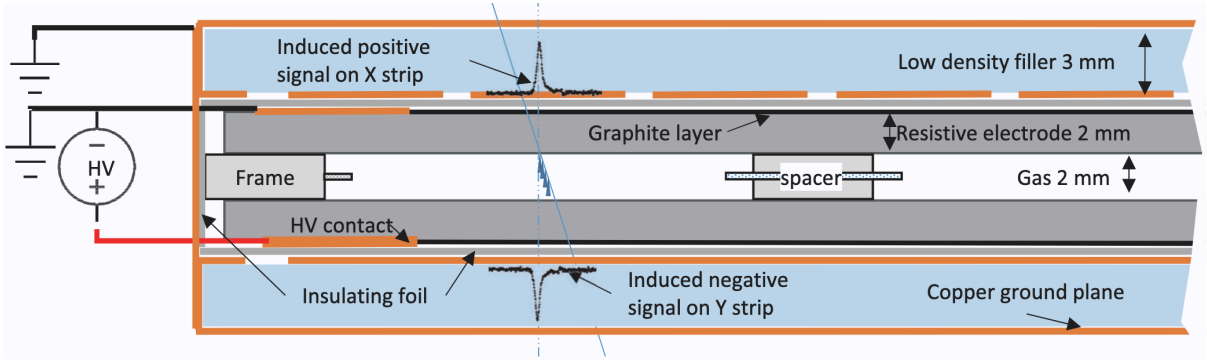
School on AI/ML Methods at the LHC · IBA Karachi · 16 July 2026

A first example: Muon Reconstruction in ATLAS

L1 muon trigger uses the RPC (Resistive Plate Chamber) detector in the barrel: 3 concentric doublet layers (RPC1/2/3), ~1 cm space / 1 ns time resolution

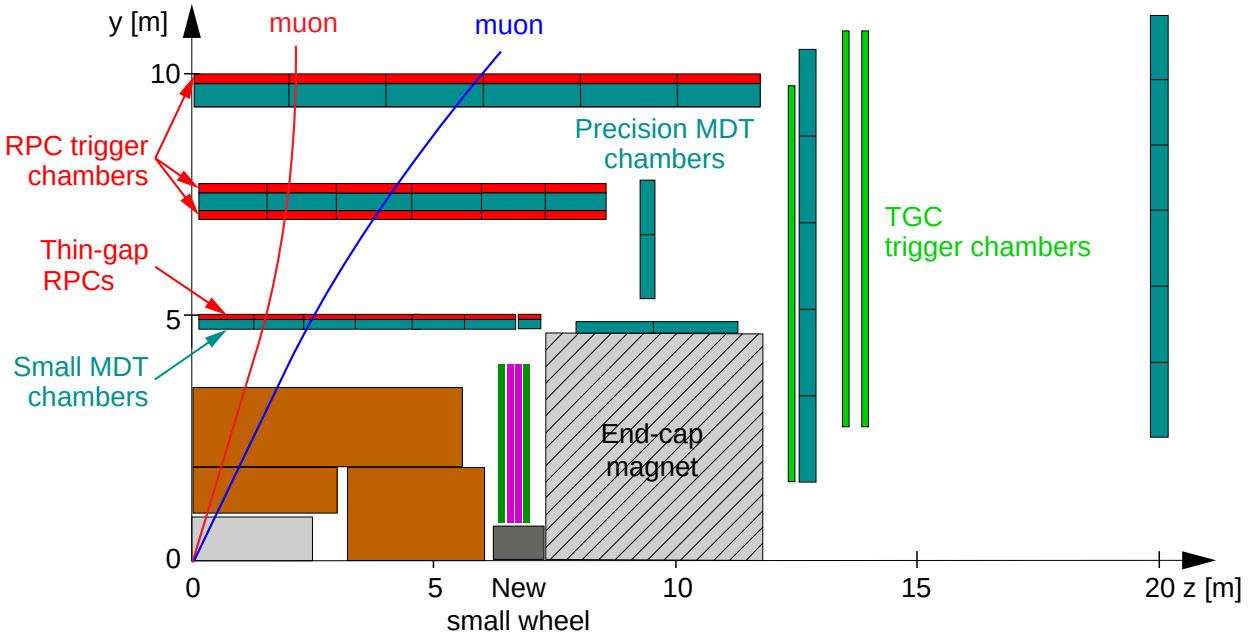
ATLAS Collaboration, JINST 16 (2021) P07029

RPC: a gaseous particle detector: two parallel bakelite electrodes (high-resistivity plastic, ~10^{10} Ω·cm) separated by a thin gas-filled gap; the charged particle ionizes the gas as it passes through; the resulting electron avalanche induces a fast, precisely-timed signal on external readout strips



The problem: the RPC's coarse strip granularity limits resolution, and this dominates the accepted background

The idea: a small NN regression model predicts muon q/p_T more precisely than today's hardware logic sharpening the trigger turn-on



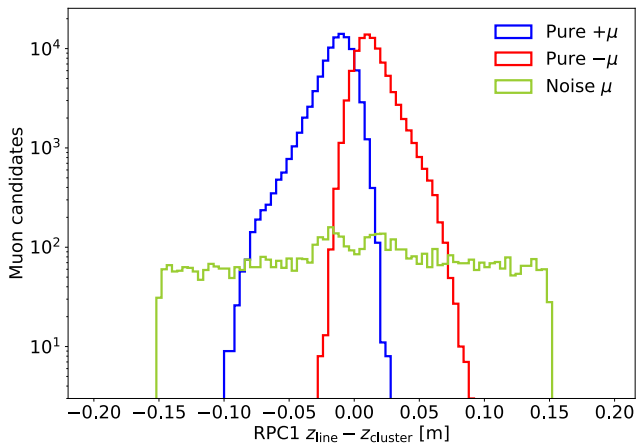
Ospanov et al., Eur. Phys. J. C 82, 576 (2022)

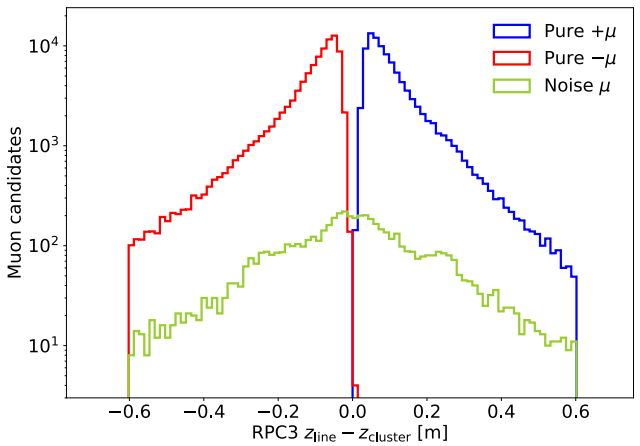
A lightweight regression network: 3 inputs, 3×20 ReLU layers

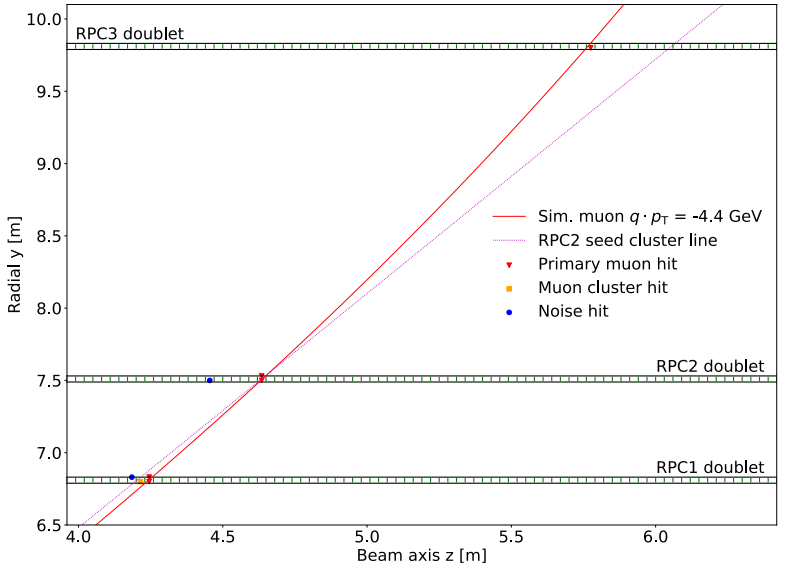
Simplified RPC simulation: muons at 3–30 GeV p_T, 40–85°, with realistic per-strip hit/cluster/noise probabilities calibrated to the real detector

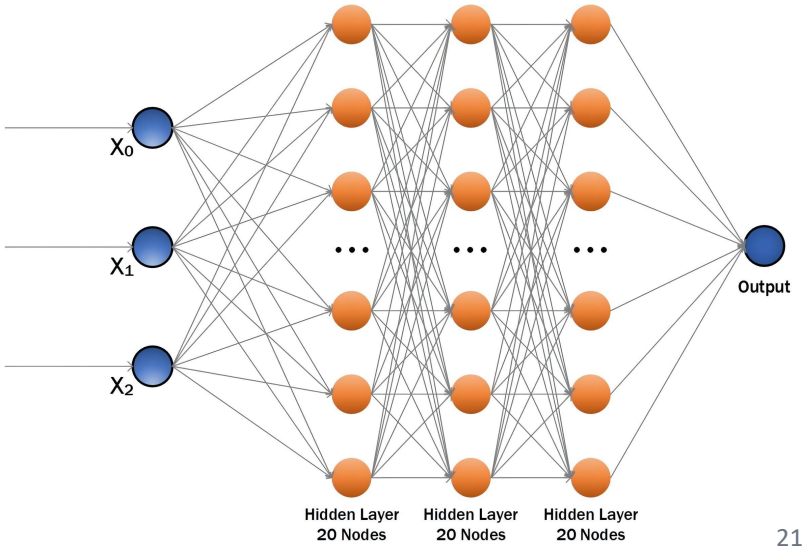
Hits in the RPC2 doublet are used as seeds to define the 3-hits inputs: $\Delta z_1, z_2, \Delta z_3$

Just 3 input variables: the z-position of the RPC2 seed cluster, and the z-differences between a straight-line extrapolation and the RPC1/RPC3 cluster positions









Architecture: 3 fully-connected hidden layers, 20 ReLU nodes each — chosen after testing several sizes; performance plateaus above ~20 nodes/layer

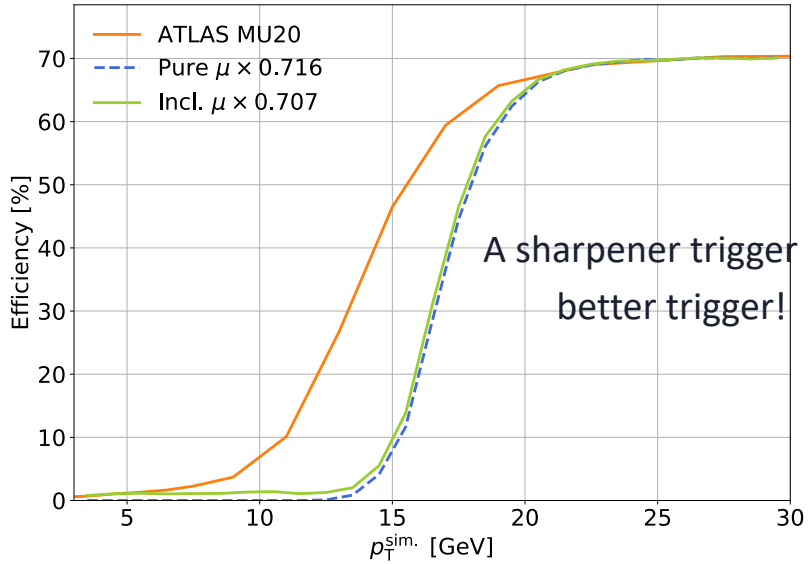
(Trained in PyTorch with an L1 (mean absolute error) loss on q/p^T)

Ospanov et al., Eur. Phys. J. C 82, 576 (2022)

Comparison with the Run3 efficiency

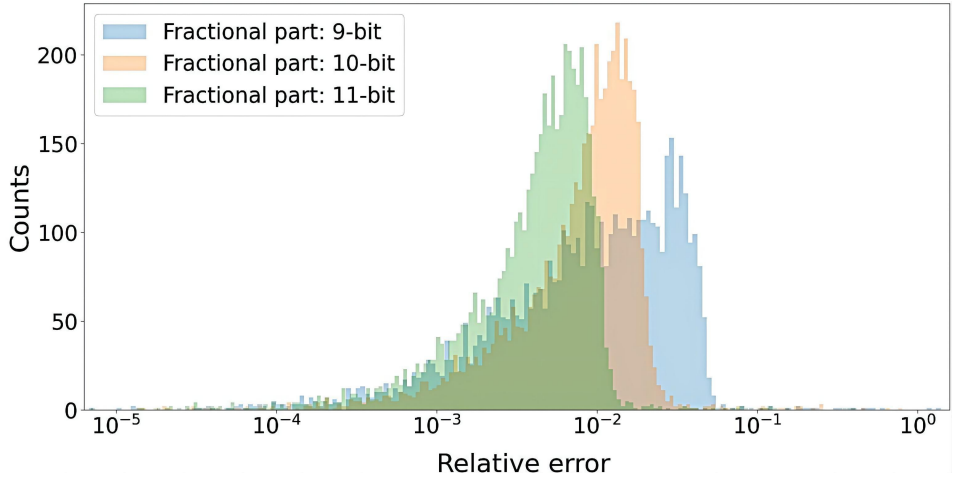
This was an attempt to attach a small NN to the RPC ATLAS trigger

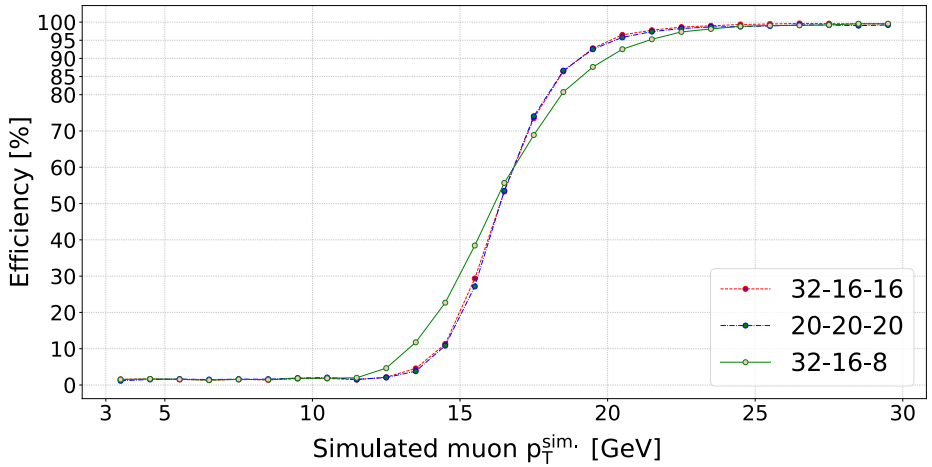
For each muon-candidate event with hits in all 3 RPC layers, the p_T is estimated from the 3 z variables, and the resulting efficiency curve is compared with ATLAS's real Run 2 trigger performance



A sharpener trigger is a better trigger!

They did an empiric evaluation of the error per different quantized models, varying the n-bits and the fractional part





The latency of $122\ ns$ with deadtime of $25\ ns$

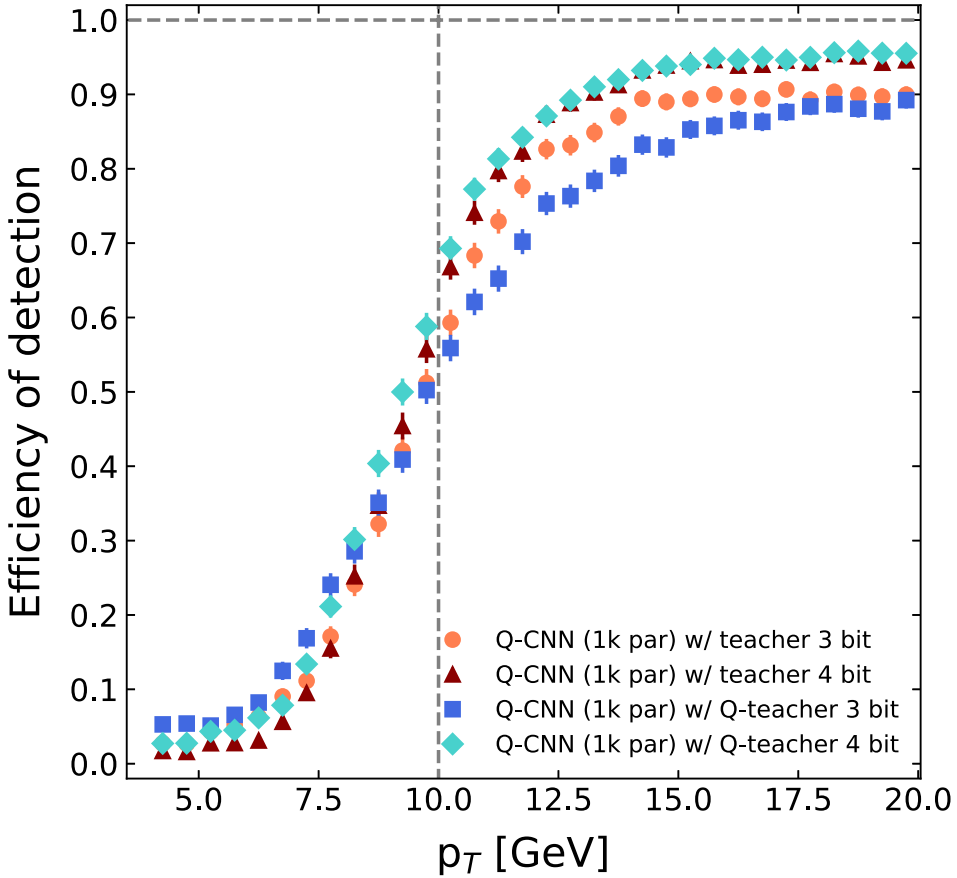
[Ospanov et al., Eur. Phys. J. C 82, 576 (2022)]

The HL-LHC new RPC system for the ATLAS experiment (work in progress)

More recent studies are focus on developing the new RPC trigger for HL-LHC era, but the required latency has to be $\sim O(100)\ ns$

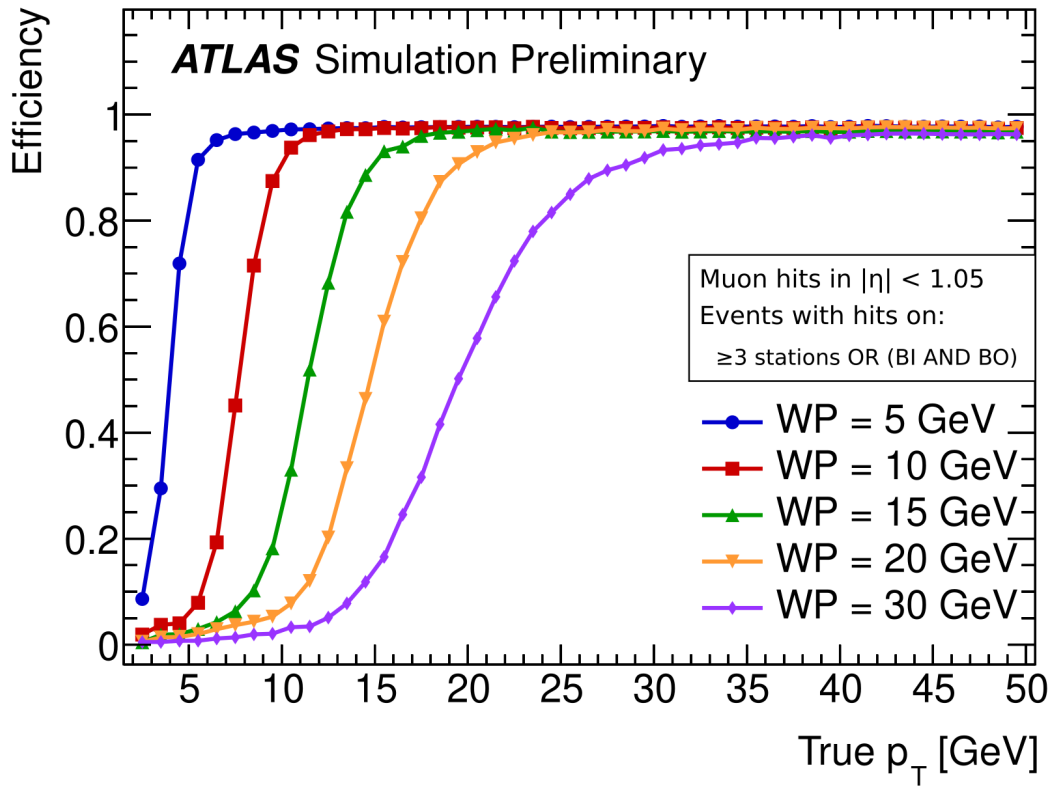
Different possible ML strategies: CNN, Fixed-graph GNN, Interaction Net… wait for future publications

S. Francescato, Eur. Phys. J. C 81, 969 (2021)



The latency of $435\ ns$ *with a frequency of* $f_{max} = 420\ MHz$

Implementation Optimisation under development



ATLAS public plot, "Accelerating Graph Neural Networks on FPGAs for Real-Time Level-0 Muon Triggering"

A second example: the CMS new strategy for jet tagging at L1

The CMS L1 Trigger for HL-LHC will exploit the jet constituents (clusters in the calorimeter) to do jet-tagging, previously only done only at HLT

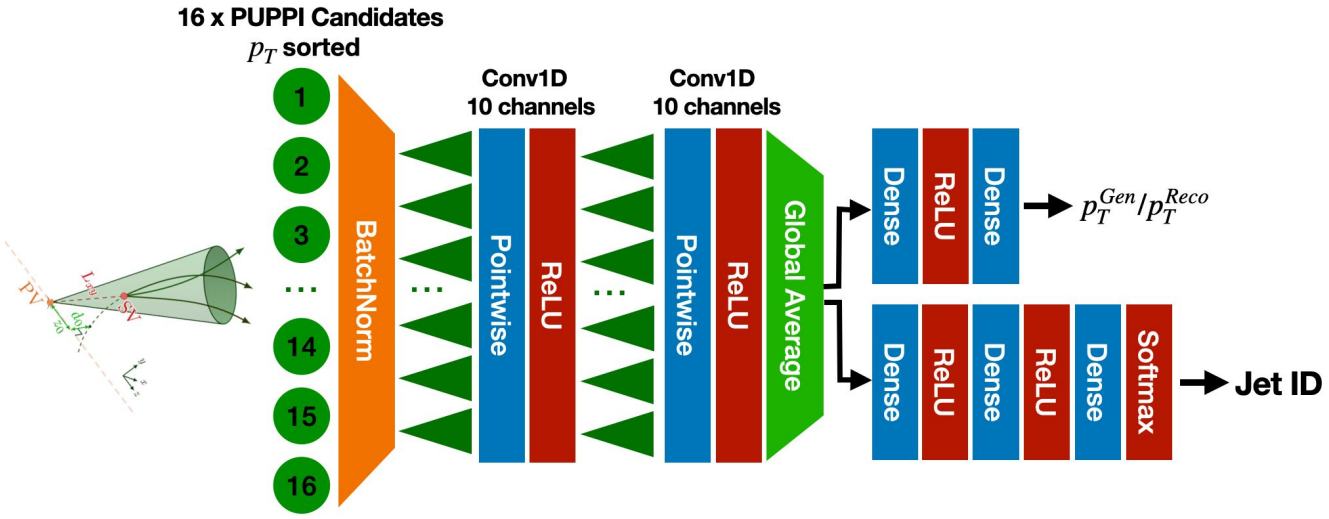
"Deep Sets", arXiv:1703.06114

The chosen model is Deepest, because if fast and constituent-permutation invariant (because of it's Sum operation)

Input: the features of the first 16 leading-p_T constituents of the jet candidate

Trained on ~21 million simulated jets (80% train / 10% test / 9% validation) with class weights to balance the categories

Two outputs to do regression of p_T and and jet identification

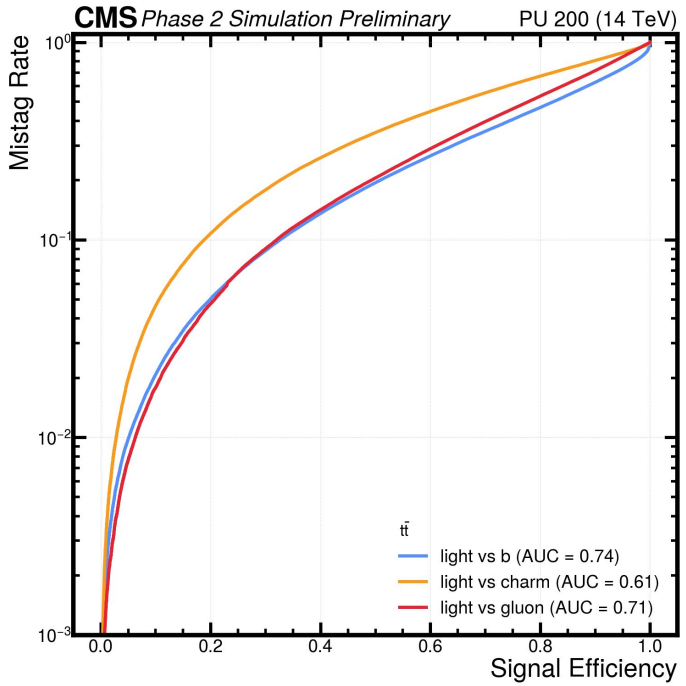


M. Schaefer et al. (CMS Collaboration), "Advancing the CMS Level-1 Trigger: Jet Tagging with DeepSets at the HL-LHC," SciPost Phys. Proc. (2025), arXiv:2509.24371

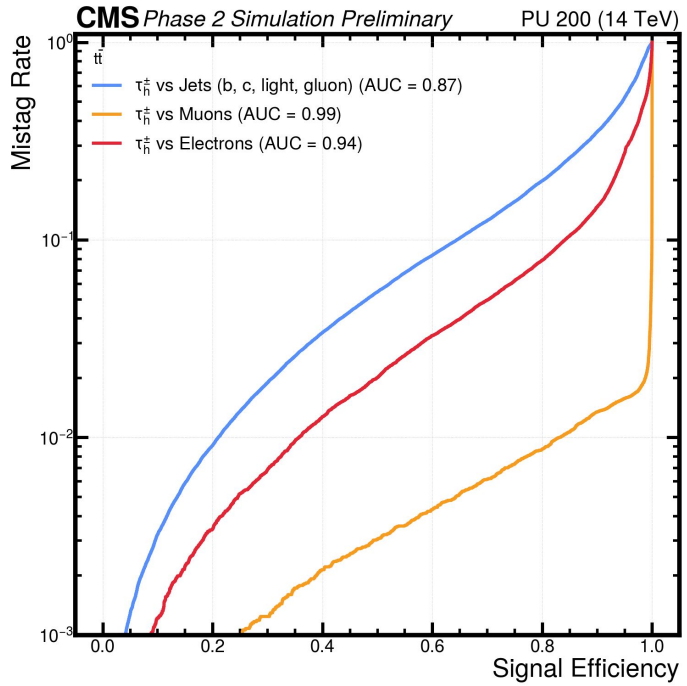
Eight classes in one model

Light quarks (uds)	charm	b	gluon	τ_h^+	τ_h^-	E	μ

Best performance on e and μ respect τ and b-iniziated jets, but the best way to test this model is to implement a trigger prototype



(a) ROCs one-vs-one: light-jets



(b) ROCs one-vs-one: τ_h

M. Schaefer et al. (CMS Collaboration), "Advancing the CMS Level-1 Trigger: Jet Tagging with DeepSets at the HL-LHC," SciPost Phys. Proc. (2025), arXiv:2509.24371

A test on a physics case: the rare HH production

The HH cross section production at LHC is expected by the SM but still unobserved, because it is a very rare process

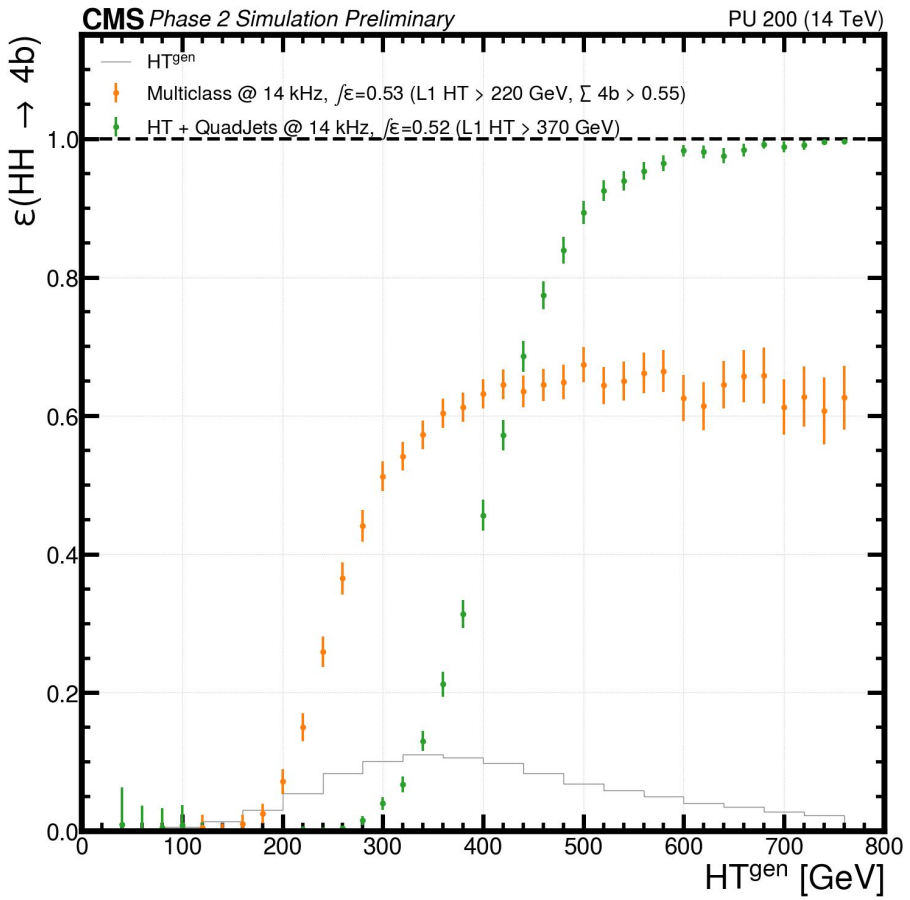
Enhancing the Level 1 trigger selection can improve the efficiency on these kind of rare events

Comparison between cut-based trigger and cut+b-tagging based:

- Green trigger requires at least 4 jets with $H_T(=\Sigma\ p_T)\ > 370\ GeV$, giving a 14 Hz rate
- Add an extra requirement of 4 b-tagged jets decrease the plateau, but with at the same rate we can go a lot lower in energy

Implemented with latency of $234\ ns$ *with a frequency of* $f_{max} = 360\ MHz$

<u>ML is not a priori better than traditional approaches, but gives flexibility…</u>
<u>mixing the two approaches we can do the union of both strategies</u>



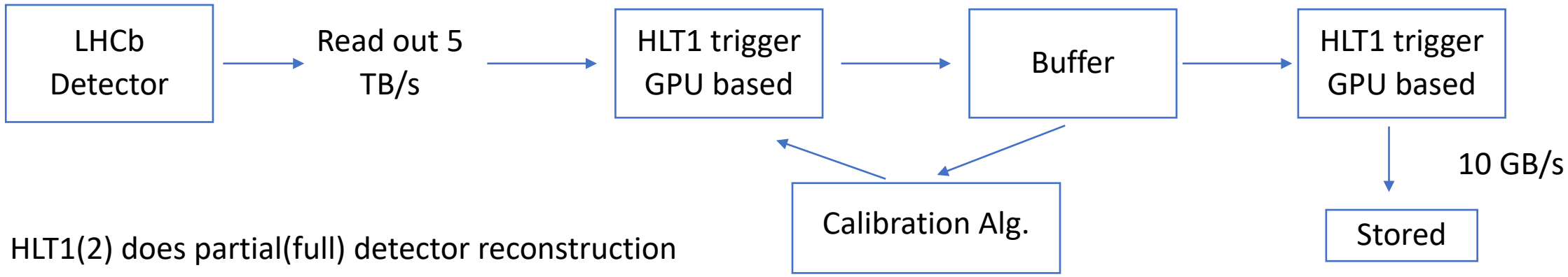
M. Schaefer et al. (CMS Collaboration), "Advancing the CMS Level-1 Trigger: Jet Tagging with DeepSets at the HL-LHC," SciPost Phys. Proc. (2025), arXiv:2509.24371

A different strategy: LHCb decided for no low level trigger for Run3 and Run4

LHCb's Run 1/2 hardware trigger (L0) had a hard rate ceiling of 1 MHz, and this was especially inefficient for fully hadronic decays

For Run3 LHCb removed Low level trigger entirely, every bunch crossing is read out and reconstructed in software

This was only possible because of two things covered earlier: LHCb's much smaller events than ATLAS/CMS, and genuine advances in GPU computing power making full software reconstruction at 30 MHz achievable for the first time



HLT1(2) does partial(full) detector reconstruction

Readout

~500 FPGA boards

FPGAs used here for formatting/ readout, NOT for trigger decisions

HLT1

~500 GPUs

Reduces the data volume by roughly a factor of 20

Buffer

disk

Real-time detector alignment and calibration happens HERE, before HLT2

HLT2

CPU (x86)

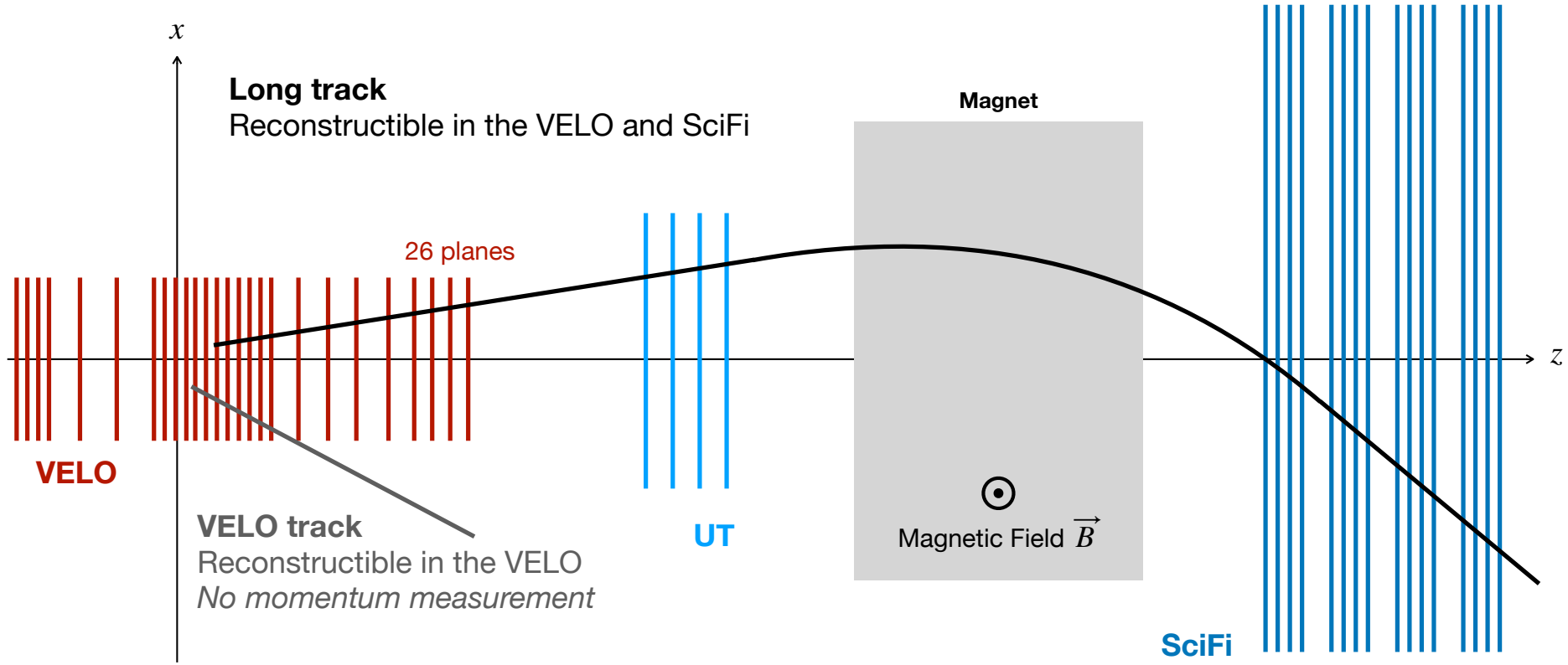
Offline-quality reconstruction; fully asynchronous, no hard latency requirement

LHCb Starterkit dataflow documentation: https://lhcb-starterkit-run3.docs.cern.ch/first-analysis-steps/dataflow/

An example for LHCb: ETX4VELO

The ETX4VELO goal is to turn raw hits left in the VELO detector into reconstructed particle tracks

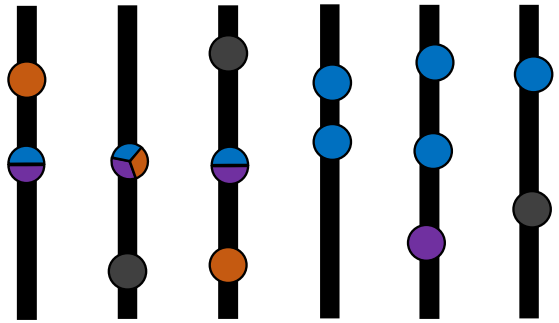
A Graph Neural Network that processes the hits left inside detector planes to connect them and build the tracks



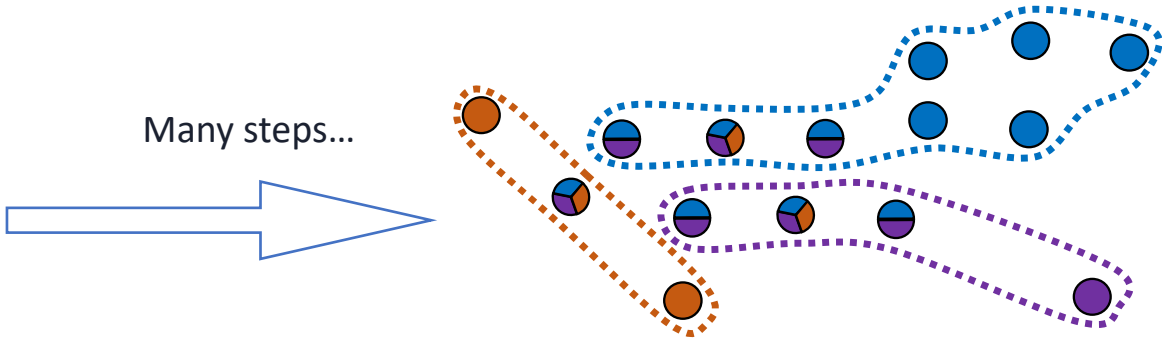
Source: Correia et al., "Graph Neural Network-Based Track Finding in the LHCb Vertex Detector," JINST 19 (2024) P12022; Giasemis et al

An example for LHCb: ETX4VELO

Build track from hits is a very difficult problem, the combinatorics could go crazy easily



Many steps…



We can simplify the chain as the following:

1 Graph construction

Embedding MLP

Learns an embedding space where hits from the SAME track land close together. Used to build a sparse candidate graph

2 Edge classification

Graph Neural Network

Takes the candidate graph and classifies each connection (edge) as a real or fake link between hits

3 Track building

Triplet-based method

Converts classified connections into final track objects

THIS is the piece ported to FPGA

Source: Correia et al., “Graph Neural Network-Based Track Finding in the LHCb Vertex Detector,” JINST 19 (2024) P12022; Giasemis et al

Hit Embedding by MLP

To enhance throughput, it is essential to minimise the graph size at this stage, otherwise the GNN will have too many

Because of the forward nature of the detector, each hit can only be connected with hits of the next 2 planes in the detector, and in general we want that hits from the same particle are near each other in the embedded space…

So we define a distance in the embedding space :

$$d^2(a,b) = \|e_a - e_b\|^2 = \sum_{i=1}^{n_{\dim}} \left(e_{a,i} - e_{b,i}\right)^2$$

And in the training the MLP has to minimise the following loss:

$$L = L_{fake} + w_{genuine} \cdot L_{genuine}$$

Where $L_{fake} = \frac{1}{|T_{fake}|} \Sigma_{(q,b)\in T_{fake}} max(0,\, m - d^2(q,b))$ It pushes fake pair to be bigger than the margin $(m = 1)$

And where $L_{genuine} = \frac{1}{|T_{genuine}|} \Sigma_{(q,a)\in T_{genuine}} d^2(q,a)$ It pushes true pair to be small as possible

Source: Correia et al., “Graph Neural Network-Based Track Finding in the LHCb Vertex Detector,” JINST 19 (2024) P12022; Giasemis et al

Conversion and comparison of the MLP on FPGA

Using hls4ml they converted the pythorch model in 8-bit quantised model (PTQ) and the they used Vivado-HLS for the implementation

They tested the model on 2 FPGAs (Alveo) and compared the energy consumption and latency with the performance on GPU (RTX3090)

Instead to compare pure latency they studied the Throughput (Events/s):

Accelerator Implementation	Alveo U50 <8,3>	Alveo U250 <8,3>	RTX 3090 TRT INT8
Throughput (Events/s $\times 10^6$)	0.55	1.10	0.82
Active Power Draw (W)	75	230	350
Energy per Event (μJ)	140	210	430
Energy Gain	3.1x	2.0x	1.0x
Price (USD)	3000	$\sim$ 10 000	1500

Smaller energy per event and higher Throughput but with an higher cost

For an higher level trigger there are many valuable options like CPU, GPU and FPGA, and it has to be studied case by case;

For an low level trigger that has to run on detector FPGA or custom ASIC are not an option, because we need deterministic latency and GPUs can't guarantee it

Source: Giasemis et al., arXiv:2502.02304

The journey we took

1 The problem

40 MHz collisions, ~1 PB/s of raw information — only 1 in ~30,000 events can ever be kept

2 The trigger

When and why is needed, and two different worlds: µs-scale hardware (L1) vs. ms-scale software (HLT)

3 Why ML fits

Exploits correlations no hand-cut can, adapts to new signals, enables anomaly detection

4 Proven at HLT

Nearly a decade of deployed deep NNs in Run 3

5 The push to L1

FPGAs, fixed resource budgets, and the techniques built to fit inside them: QAT, pruning, HGQ

6 Real examples

ATLAS muon trigger, CMS jet tagging, LHCb's triggerless GPU architecture and ETX4VELO

The key takeaways

1 **“Real-time” means something different at every level**

Milliseconds and CPU/GPU at HLT; nanoseconds and FPGAs at L1 — different hardware, different constraints, different techniques.

2 **Real-time ML is a valuable option**

ML makes triggers more flexible, adapting to signals no one explicitly programmed for and opening entirely new strategies

3 **There is still room for nice ideas**

QAT, pruning, knowledge distillation, HGQ, all these techniques already shown how impactful could be ideas to have “ a smaller model”

UNIVERSITÉ DE GENÈVE

Thank you! questions?

Davide Fiacco

CERN