Development of the Phase-2 CMS Overlap Muon Track Finder

Advancing Muon Reconstruction with HLS and Graph Neural Networks

Pelayo Leguina, Clara Ramón, Pietro Vischia, Santiago Folgueras

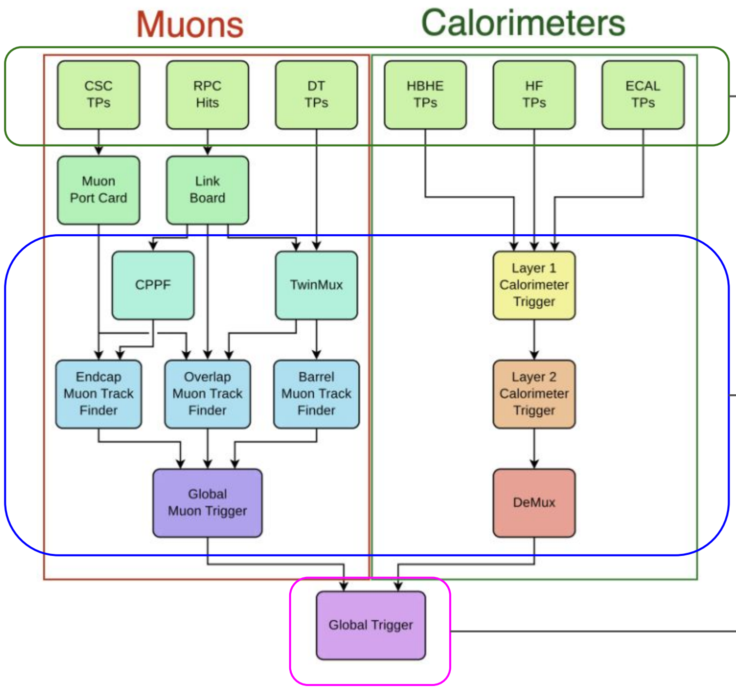
Summary

What is this about?

- CMS Level-1 Trigger System
- Phase-2 Upgrade
- OMTF Algorithm Status
  - Purpose
  - HLS approach
  - Firmware build pipeline
  - Board integration
- ML for triggering particles
- GNN Tracking
- GNN in OMTF
- Test case

CMS Level-1 Trigger System

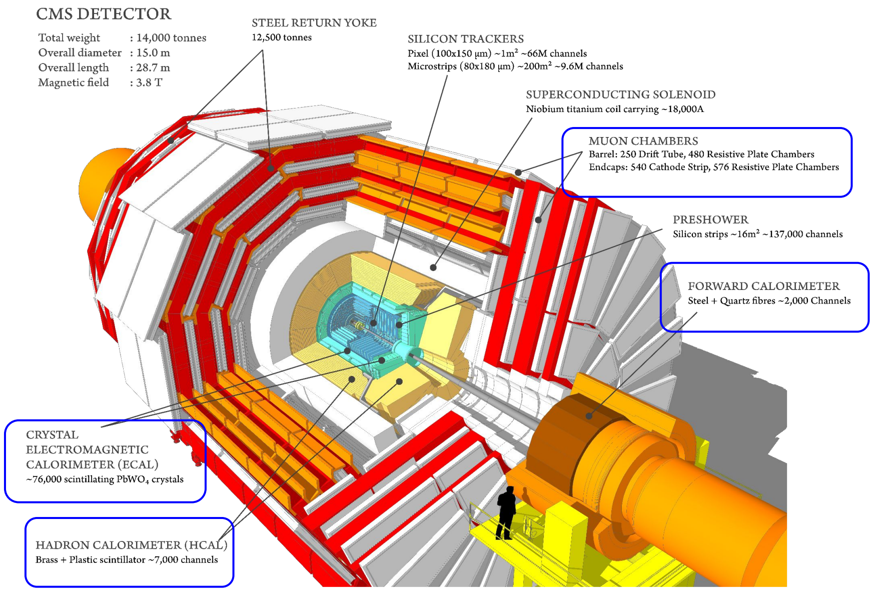
Today



Input Data

Physics Objects Reconstruction

Final decision



- **@40 MHz**
- **Short amount of time : < 3.8 us (latency)**
- **Large data volume**

Selecting physics with simple criteria implementable in hardware

Phase-2 upgrade

Key parameters

- **Increase bandwidth 100 kHz → 750 kHz**
- **Increase latency 3.8 us → 12.5 us**
- **Include high-granularity information**
- **Include tracking information**

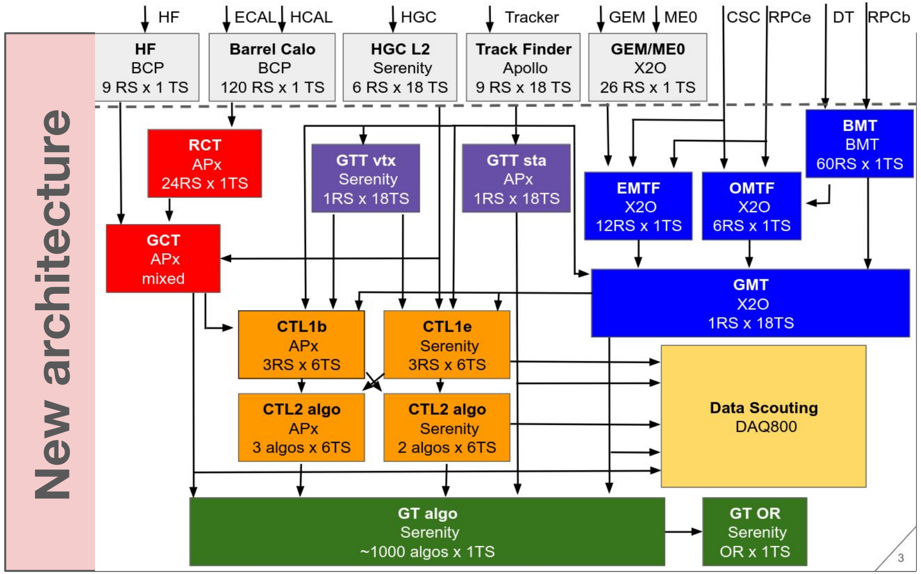
Requirements

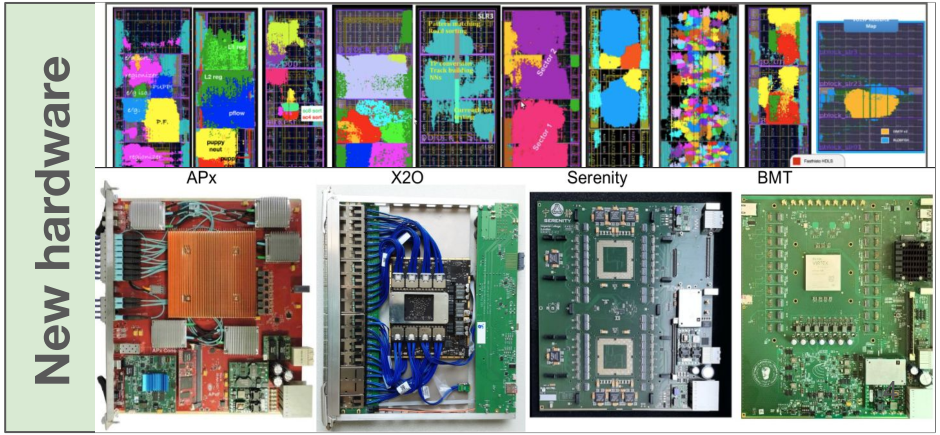
- **Cutting-edge hardware**
  - FPGA VU13P, 28 Gb/s links
- **Advanced Architecture**
  - ATCA standard, flexible & modular design

Firmware

- **Algorithm developed mostly in High Level Synthesis (HLS)**
  - Used successfully, much faster turn-around
- **Many tools available for Machine Learning inference**

hls4ml Conifer



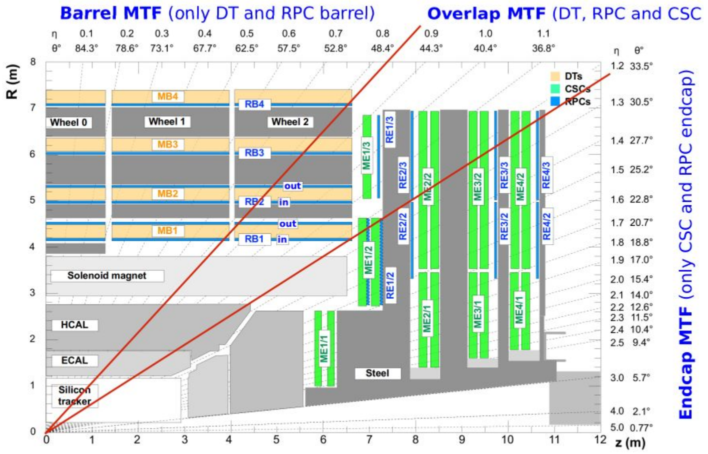


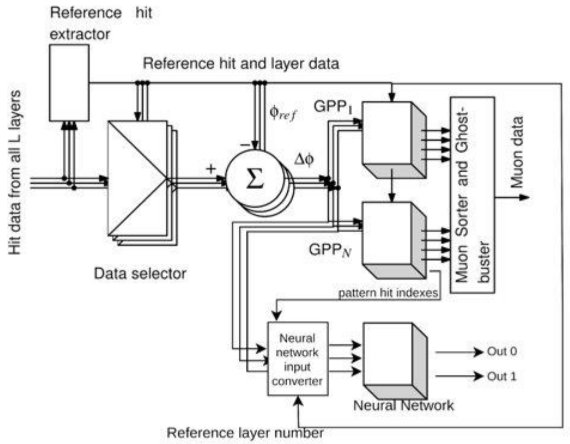
OMTF Algorithm

Overlap Muon Track Finder

- Designed to reconstruct muon trajectories in the barrel-endcap transition region of the detector.
- The algorithm evaluates how well the stubs correspond to expected patterns of muon tracks with specific pT.
- By calculating a similarity score between the observed stubs and these reference patterns, the algorithm identifies the most probable track candidate.

For Phase-2, we want to achieve a modular and maintainable design that can be easily adapted for future upgrades and more complex detector conditions.





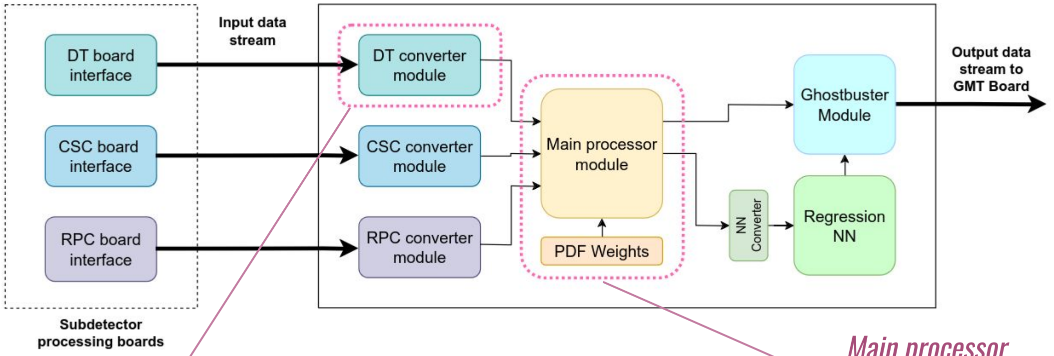
https://iopscience.iop.org/article/10.1088/1748-0221/11/03/C03004

Phase-2 OMTF

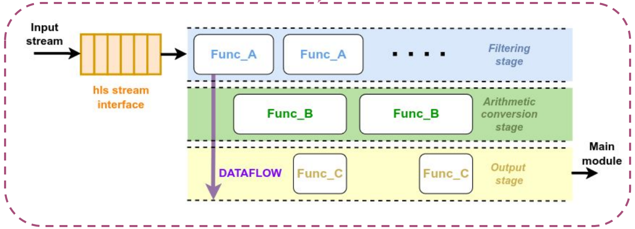
HLS approach

- Use of Vitis HLS to design each module.
- A direct adaptation from emulator code is made.
- Optimization techniques such as pipelining and memory arrangement.
- Individual testing for each module.
- Emulator - Hardware matching.
- Building pipeline.

General scheme

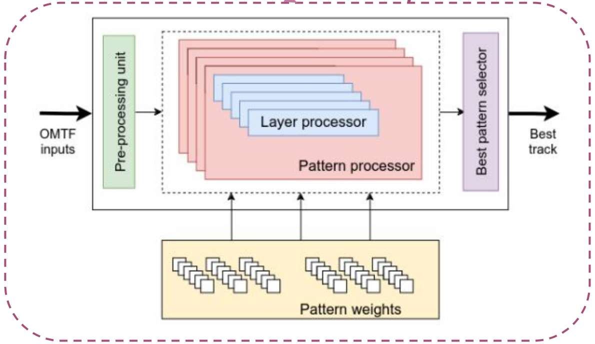


Input converting modules



- Manage streamed data.
- Parse data frames.

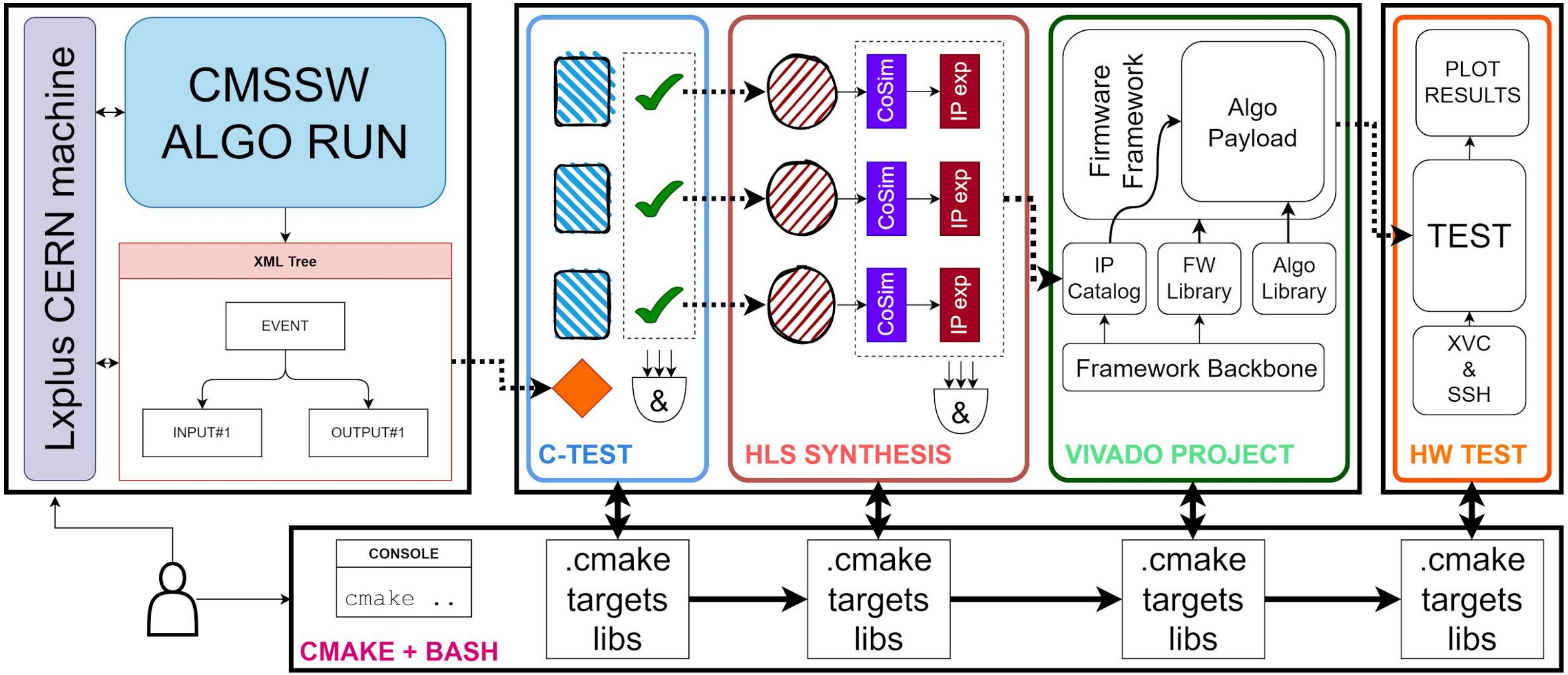
Main processor



- Parallel “golden pattern” processing.
- Stored weights reshaped for simultaneous availability.

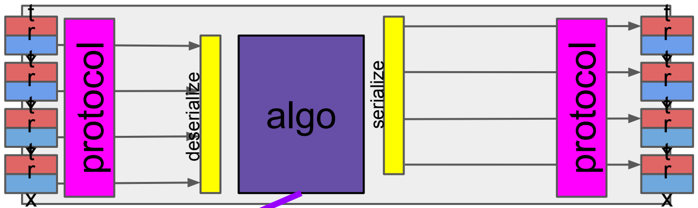
Phase-2 OMTF

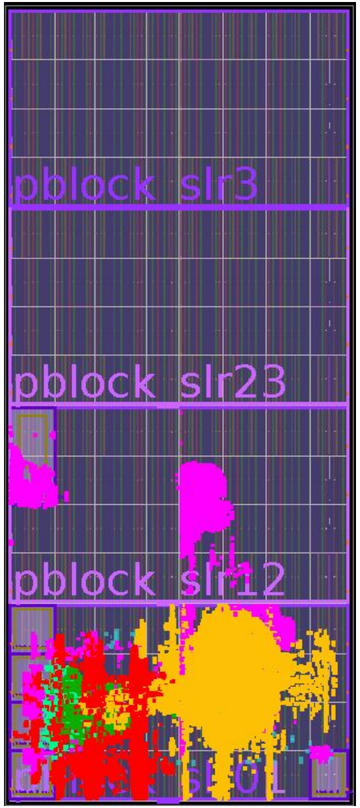
Firmware build pipeline

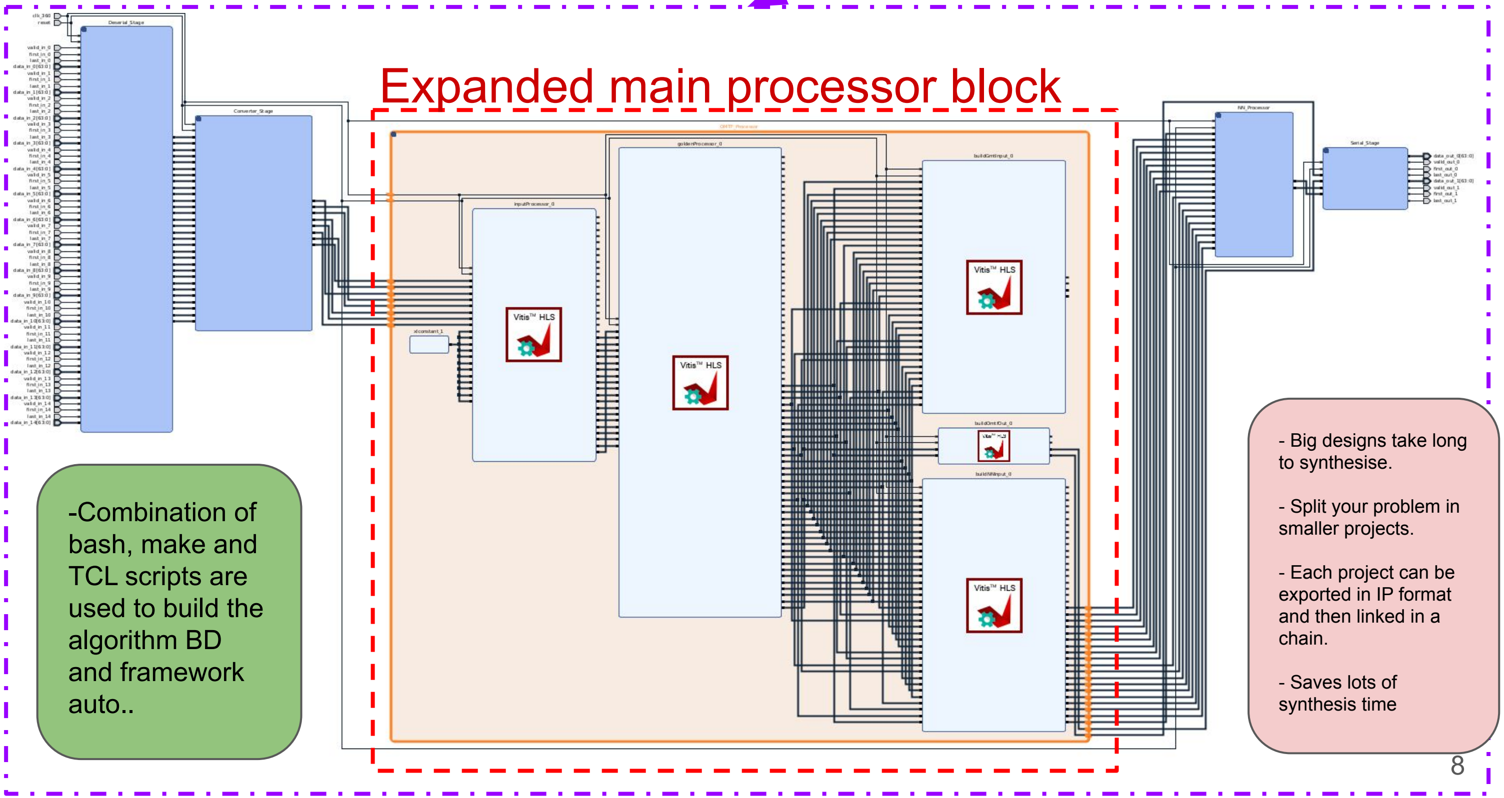


Board integration

Easy integration in vivado







-Combination of bash, make and TCL scripts are used to build the algorithm BD and framework auto..

- Big designs take long to synthesise.
- Split your problem in smaller projects.
- Each project can be exported in IP format and then linked in a chain.
- Saves lots of synthesis time

HLS
Extras

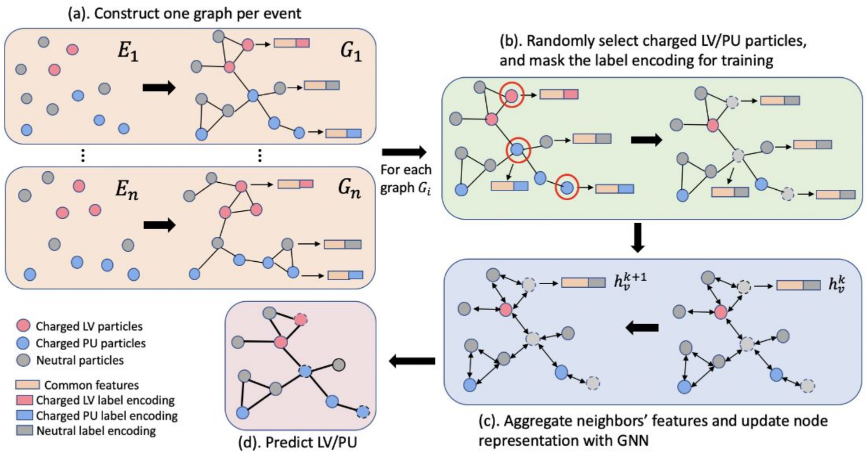
- You can use C++11 and higher constructs. Nice paper
- Read the list of pragmas and experiment a lot with them.
- HLS likes ternary operators !!

- Using C++ classes and template does not affect resource usage while improving code flexibility and ease of use. HLS-Classes-Templates

ML for track finding

Why is that?

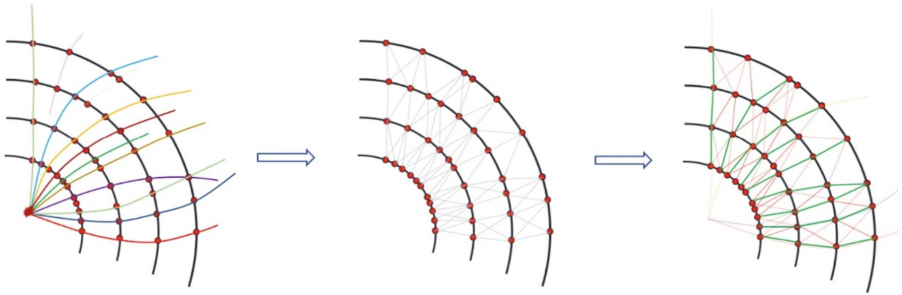
- Machine learning is growing in popularity, and the fields of HEP and LHC are no exceptions.
- In HEP, there is a growing trend towards utilizing larger and more complex machine learning models, along with increasing demands for computational power.
- Availability of modern hardware.



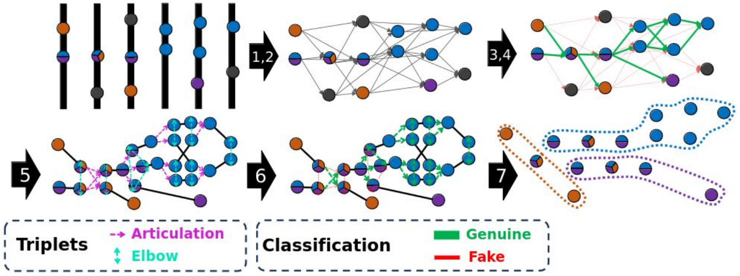
https://arxiv.org/pdf/2203.15823

GNN Tracking

- Tracking is an extremely challenging problem.
- The combinatorial complexity is vast and will only intensify over time.
- Graph Neural Networks (GNNs) offer promising solutions for tracking in the High-Luminosity Large Hadron Collider (HL-LHC).
- FLOPs and power efficiency are critical factors to consider.
- Pruning is a potential strategy to reduce complexity and resource demands.



LHCb exploring the use of GNNs



https://arxiv.org/abs/2407.12119

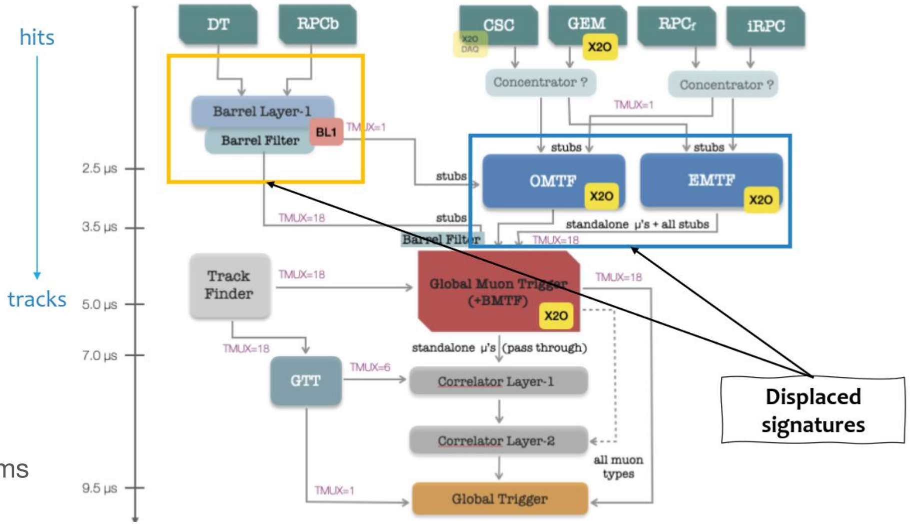
Level-1 Trigger Level - O(μs) Latency.

GNN in OMTF

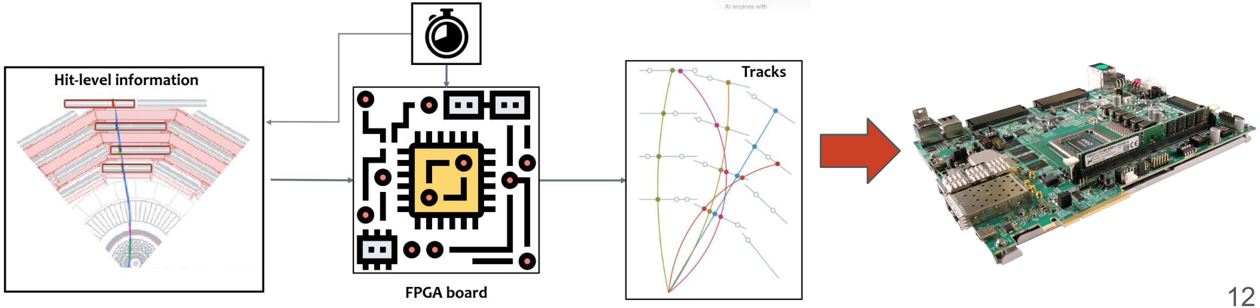
INTREPID project

INnovative**TR**igg**E**r techniques for beyond the standard model **P**hys**I**cs**D**iscovery at the LHC

- LLP signals might be easily overlooked or misinterpreted in LHC data.
- Enhancing muon triggers within the current architecture focuses on optimizing algorithms and refining data processing techniques to improve detection efficiency without requiring significant hardware upgrades.



Explore alternative technologies and ideas which could not be otherwise investigated that could potentially lead to a significant breakthrough.



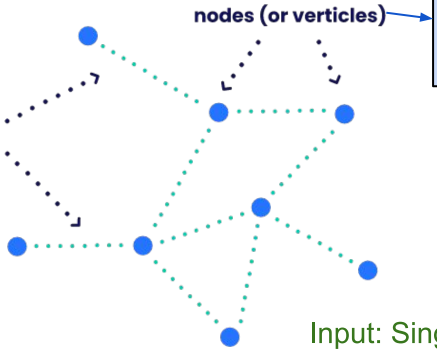
Software implementation

Designing a basic network

Current training:

- Muon gun phase-II sample, using only negative muons (symmetry)
- Batches of 64 graphs (events)
- Learning rate = 0.0005
- Weight decay = 0.75
- Epochs = 1000

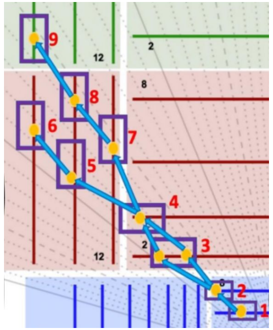
Stub detected in a station
Stub coordinates
Stub number
Stub type



Connection between two stubs
Difference in phi
DIfference in eta

Input: Single event stubs graph

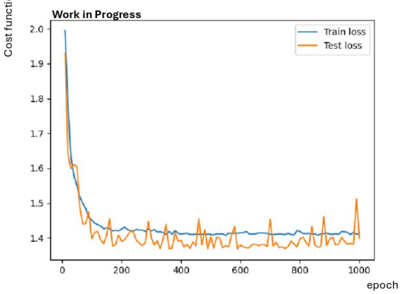
GNN



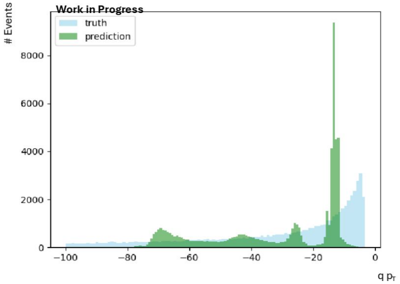
Output: Regress muon pT

- Using fully connected graphs for the moment.
- Using pyTorch geometric libraries.
- Current architecture based on two Graph Attention Layer (GAT) [arXiv:1710.10903] to process graph data, making use of the edge information. After each GAT layer, ReLU activation is applied. The model combines global mean and average pooling to aggregate node-level features into a graph-level representation.

Loss function



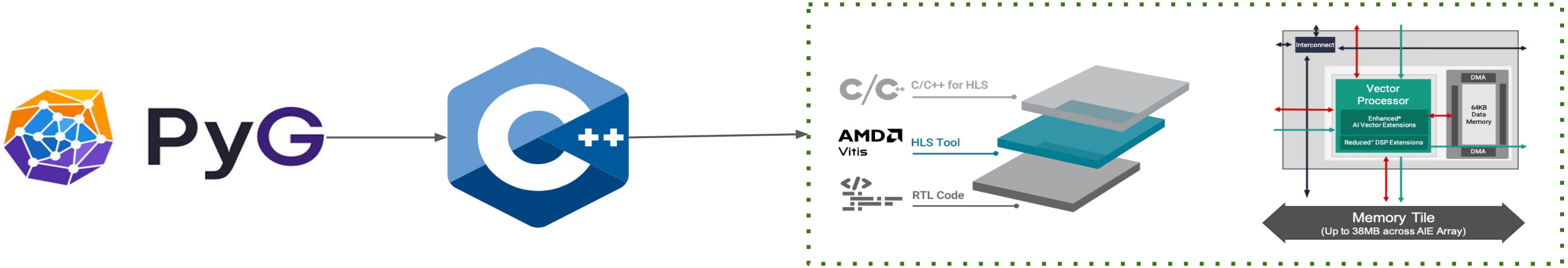
pT regression



Hardware implementation

First steps to design with versal architecture

- Build specific kernels, each processing a different layer of the net.
- This would require a direct implementation on C++, to build HLS kernels.
- The high-computational cost operations would be run in the versal AI cores.
    - Implement the model in C++ using vectorized operations that can leverage the parallel computing capabilities of the cores.
    - Develop AI Engine kernels for the most compute-intensive parts of your model. The AI Engine kernels are designed to run on the AI Engines, which are optimized for high-throughput, low-latency vector processing.
    - Integrate your AI Engine kernels with the rest of the system.
    - Profile the application to identify bottlenecks



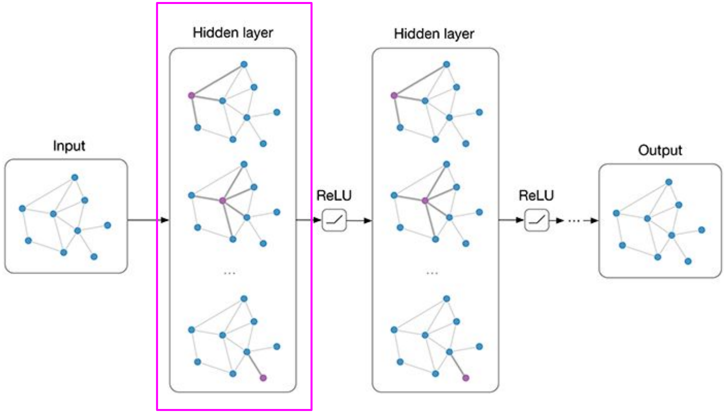
Test case

Implementing a simple GCNConv layer

- Graph Convolutional Networks extend the concept of convolution from grid-like data (like images) to graph-structured data.

$$\mathbf{X}' = \hat{\mathbf{A}}\mathbf{X}\mathbf{W} + \mathbf{b}$$

Aggregates feature information from a node's neighbors (including itself) and transforms it using learnable weights.



In PyG the conv layer is called **GCNConv**

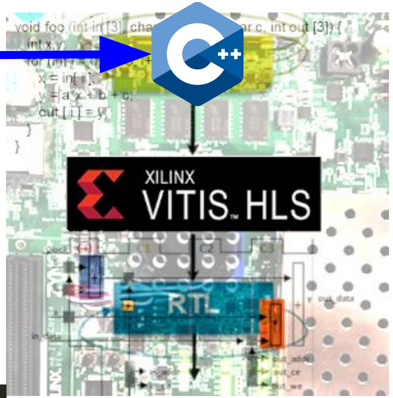
We dissect the layer into its main subfunctions:

Feat transform (Norm → Linear) → Message Passing (Aggreg. → Update) → Bias → Act. Funct.

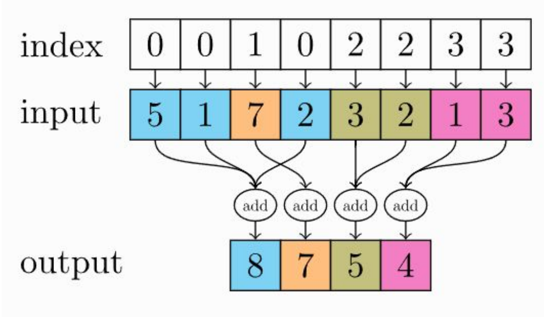
$\hat{A} = \hat{D}^{-1/2}(A + I)\hat{D}^{-1/2}$ $X' = XW$ $X''_i = \sum_{j \in \mathcal{N}(i)} m_{j \to i}$ $m_{j \to i} = \hat{A}_{ij} X'_j$ $X'''_i = X''_i + b$ $H_i = \sigma(X'''_i)$

Test case

Implementing a simple GCNConv layer



- We need C++ to import the layer into our device w/HLS!!



Aggregate example

$$m_{j \to i} = \hat{A}_{ij} X'_j$$

```python
def aggregate(
    self,
    inputs: Tensor,
    index: Tensor,
    ptr: Optional[Tensor] = None,
    dim_size: Optional[int] = None,
) -> Tensor:
```

```cpp
using Features = std::vector<std::vector<float>>;
```

```cpp
virtual Features aggregate(const Features &messages, const EdgeList &edge_index, int num_nodes)
{
    // Initialize a vector to store the aggregated messages for each node
    Features aggregated_messages(num_nodes, std::vector<float>(messages[0].size(), 0.0f));

    for (size_t i = 0; i < edge_index.size(); ++i)
    {
        int target = edge_index[i].second; // Target node index

        // Aggregate the message to the target node
        for (size_t j = 0; j < messages[i].size(); ++j)
        {
            aggregated_messages[target][j] += messages[i][j];
        }
    }

    // Optionally, implement other aggregation types (mean, max, etc.) based on your model's requirements

    return aggregated_messages;
}
```

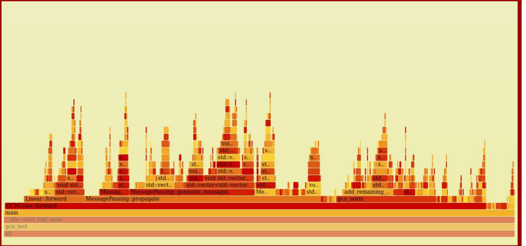
torch_geom_doc

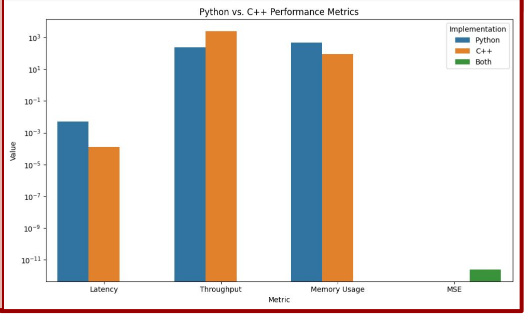
Test case

Implementing a simple GCNConv layer

Profiling

```
=== Profiling Summary ===
gcn_norm Total Time: 0.00556605 seconds
linear.forward Total Time: 0.017838 seconds
generate_messages Total Time: 0.00619323 seconds
aggregate Total Time: 0.00248693 seconds
update Total Time: 0.00160153 seconds
propagate Total Time: 0.143146 seconds over 1001 runs
propagate Average Time: 0.000143003 seconds
forward Total Time: 0.171707 seconds over 1001 runs
forward Average Time: 0.000171536 seconds
=========================
```

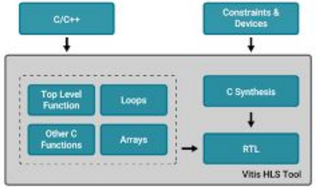




- After converting all the main elements, we test the performance of the layer and do profiling, so we can check the metrics and see what are the high-computational cost zones.

Then we build the HLS-like version

```
class MessagePassingHLS
{
public:
    // Top-level propagate function
    void propagate(
        const EdgeFeature edge_index[NUM_EDGES],
        const feature_t x[NUM_NODES][FEATURE_SIZE],
        const feature_t edge_weight[NUM_EDGES],
        feature_t updated_features[NUM_NODES][FEATURE_SIZE]
    );
```

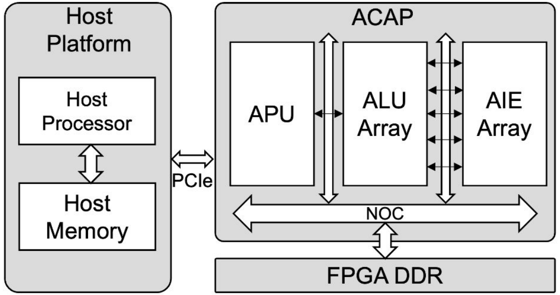


Test case : next steps

Implementing a simple GCNConv layer

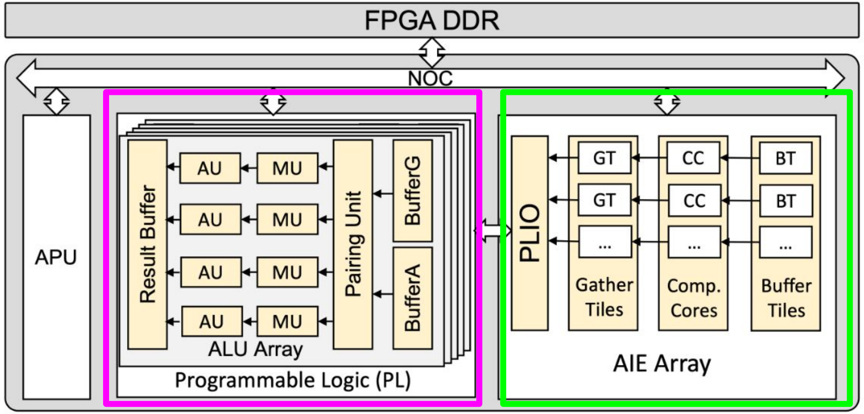
Matrix computation in the PL.

Matrix computation using AIE cores.



Implement the kernel cores into the versal device.

Experiment with different architectures for the matrix computations.



- The main idea is to develop an hybrid system that allows us to achieve maximum performance for the GNN inference.

To conclude…

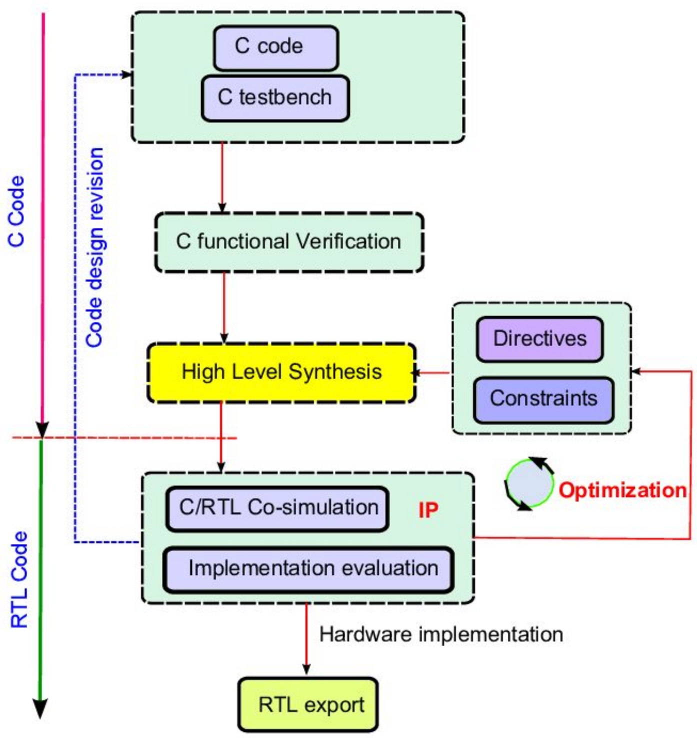
What have we seen so fart?

- The main ideas behind the CMS Level-1 trigger and its upgrade.
- The Overlap Muon Track Finder behaviour and some ways of implementing it on hardware.
- The increase of use of ML algorithms for tracking purposes.
- Some steps in the quest of developing a trackfinder by using a GNN architecture.

Thanks for your attention, see you tonight :)

Backup slides

HLS design flow



- We want to design a build framework that allows us to follow the usual HLS design flow.
- While this is easy for a single module, it becomes dirtier once your system grows:
  - Lots of HLS projects.
  - Lots of different testbenches for similar payloads.
  - Reusable code.

C test - Algorithm test library

Class CLIUtils

- Passes args:

(Bx, Processor, Events file, module)

Class TestData

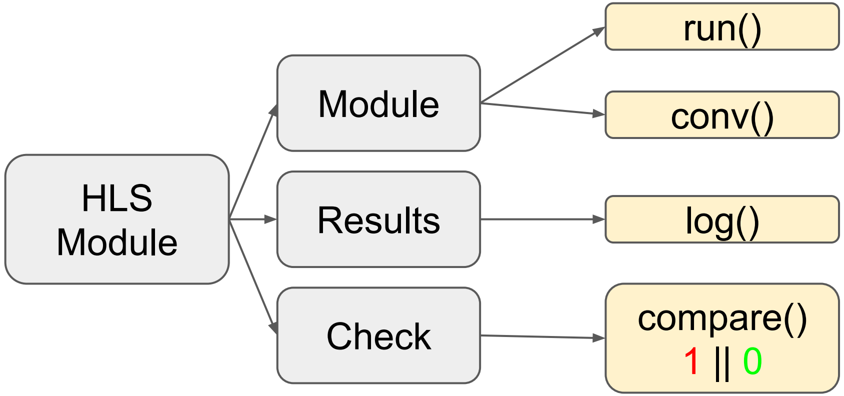
Class iResult

Class iCheck

Class iModule

```cpp
class IModule
{
public:
    virtual std::unique_ptr<IResult> run(const testData_t &data, std::unique_ptr<IResult> prevResult = nullptr) = 0;
    virtual ~IModule() {}
};
```

Maybe some modules require a previous module result as an Input **

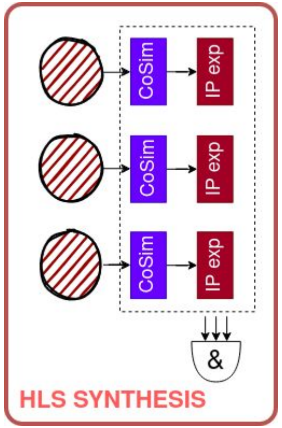


Test structure

1. Parse args
2. Load testData
3. addModule
4. runModule
5. logResults
6. checkResults

- For each new module, new class is created.

HLS Synthesis - Design Flow



COMMON SYNTH PARAMS.

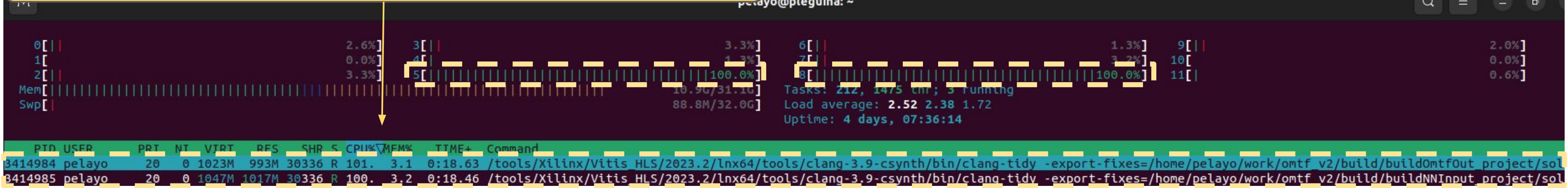
- Vitis HLS library.
- HLS global params.
  - FPGA part
  - Clk period
  - CSIM/COSIM/EXP
- Export path
- Flow

Build TCL script

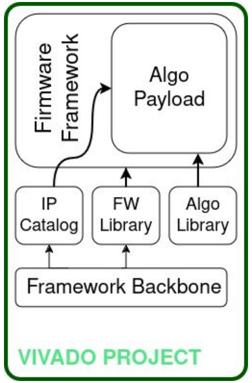
Module Sources → Apply params → Make target → CoSIM + Flow → Export IP+Token

Timing check
Unzip
To IP catalog

Parallel Vitis HLS process execution is possible **



Vivado Synthesis



COMMON VIVADO CMAKE

- Check vivado environment
- Check framework token
- Set global params:
  - Project name
  - Payload file y or n

```cmake
# Define the path to the marker file
set(MARKER_FILE ${CMAKE_BINARY_DIR}/framework_built)

# Custom target to build the framework
# Only define build_framework if marker does not exist
if(NOT EXISTS ${CMAKE_BINARY_DIR}/framework_built)
    add_custom_target(build_framework
        COMMAND ${CMAKE_COMMAND} -E chdir ${PROJECT_SOURCE_DIR}/blobfish4omtf bash scripts/build_framework.sh
        COMMAND ${CMAKE_COMMAND} -E touch ${CMAKE_BINARY_DIR}/framework_built
        COMMENT "Building the framework and setting up marker..."
        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}
    )
else()
    message(STATUS "Marker file found. Skipping framework build.")
endif()
```