Efficient data movement for Machine Learning inference in heterogeneous CMS software

Christine Zeh[1,2], Lukasz Michalski[3], Emanuele Coradin[4], Leonardo Beltrame[5], Eric Cano[1], Felice Pantaleo[1], Davide Valsecchi[6], Markus Holzer[1]

on behalf of the CMS Collaboration

[1]CERN, [2]Imperial College London, [3]AMD, [4]Universitá di Padova, [5]Politecnico di Milano, [6]ETH Zurich

28th Conference on Computing in High Energy and Nuclear Physics

CERN | NexTGen Next Generation Triggers

Next-Generation Trigger Project (NGT)

5-year project (2024-2028): Collaboration between CMS, ATLAS, CERN Theory, IT

Motivation: New physics may be buried in the **background** → current triggers may discard signals

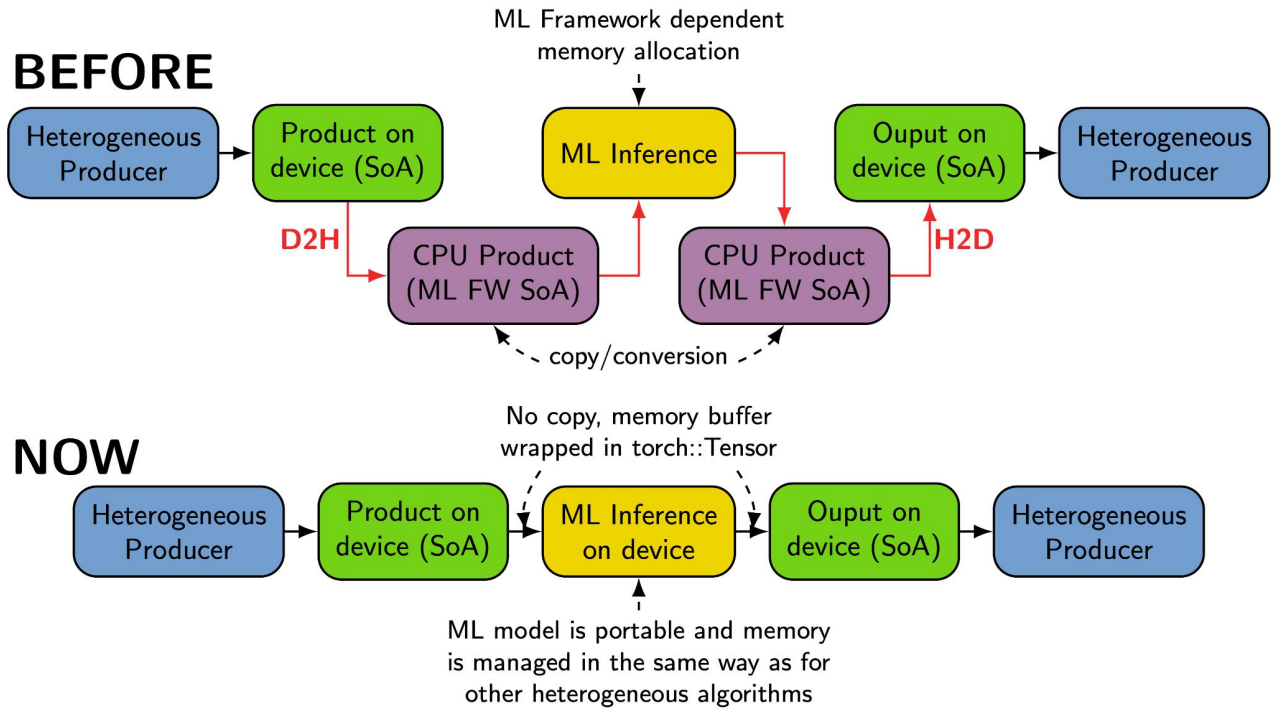
Goal: Explore innovative algorithmic and computational approaches to **expand trigger acceptance**

Task 1.7: Common software developments for heterogeneous architectures:

- Efficient usage of accelerator (GPU and FPGA) devices in software for High-Luminosity LHC
  - Efficient heterogeneous scheduling (Scheduling for Next Generation Triggers [Fila M.])
  - Efficient portable data structures (A Unified Interface for Different Memory Layouts [Chen J.])
  - Alternative programming languages (Exercising the novel and promising Mojo language in HEP frameworks [Naumann A.])
  - Common accelerated libraries
  - **Efficient accelerator interfaces to ML inference** (Yukti: A Unified Interface to ML Runtimes for Inference across Heterogeneous Architectures [Sengupta S.])

Motivation

- Increasing **migration** of algorithms to portable, **heterogeneous implementations**
- Adoption of efficient memory layouts such as **SoAs** and **PortableCollections (Efficient Data Layouts and heterogeneous data handling in CMSSW** [Holzer M.]**)**
- **No streamlined interface for ML inference** in *alpaka-based* code **(The alpaka C++ library for performance portability** [Bocci A.]**)**
- **Avoid suboptimal memory management** pipelines e.g., tensor conversions



Capability to directly interact with the memory layer torch::from blob()

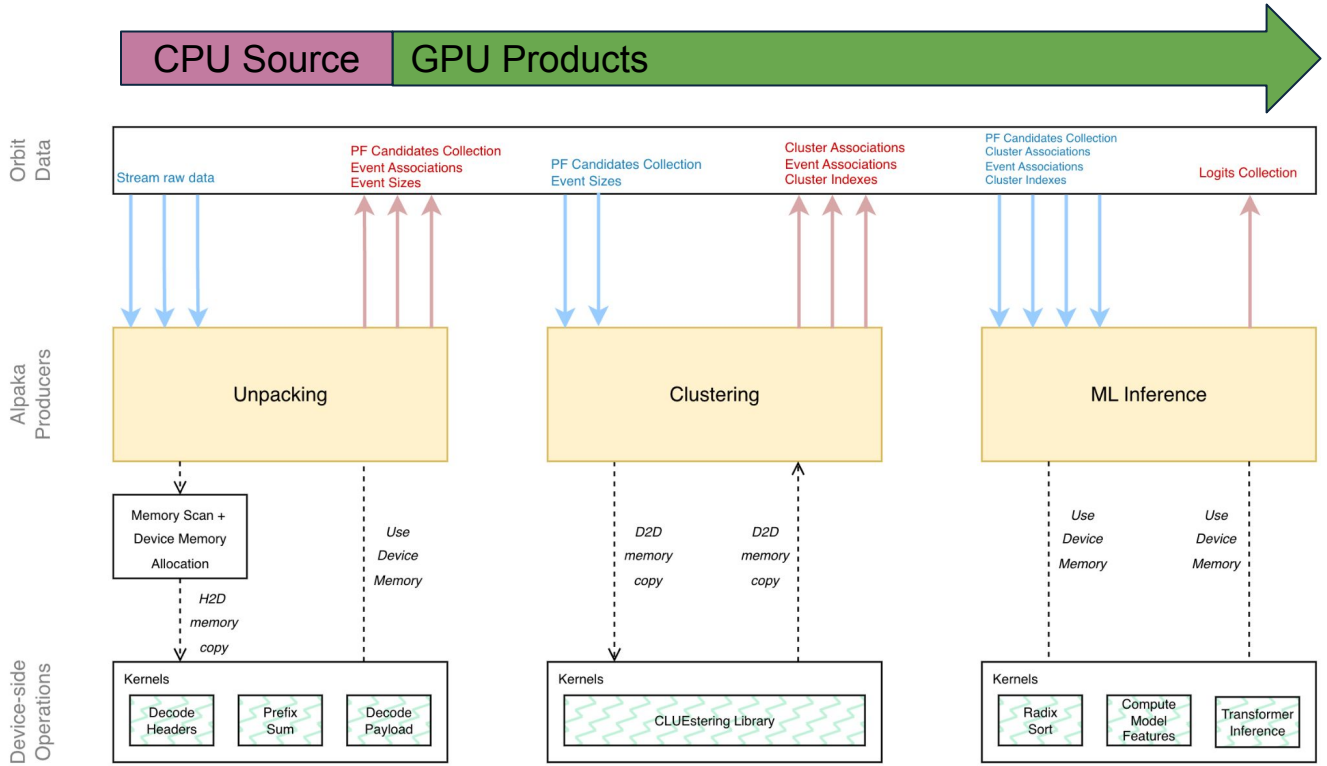
[1] C. Zeh, Efficient data movement for Machine Learning inference in heterogeneous CMS software, ACAT 2025

Motivation

- Heterogeneous Low pT Tau Tagging
- Accelerator-based input decoding
- CLUEstering algorithm[1]
- ML model for signal/background discrimination and pt regression

Problem:

- **Multiple SoA-based** (Structure-of-Arrays) **collections**, already allocated, should be used in the inference step
- **Expensive copies** of large data blobs (event-level batching)
- **No efficient solution** exists in CMSSW



- L1 Scouting: **Designing a Heterogeneous Computing Architecture for Level-1 Data Scouting in CMS at the HL-LHC** [Zago G.]
- Similar setup in the rest of the reconstruction code: **TICL: The Iterative CLustering Framework for the CMS Phase-2 Event Reconstruction** [Redjeb W.]

[1] S. Balducci et. al., CLUEstering, DOI: https://doi.org/10.5281/zenodo.20270689, 2026

Goal

Provide a robust solution that enables users to **seamlessly integrate** both **machine learning** and **GPU-accelerated algorithms** into CMSSW with minimal overhead.

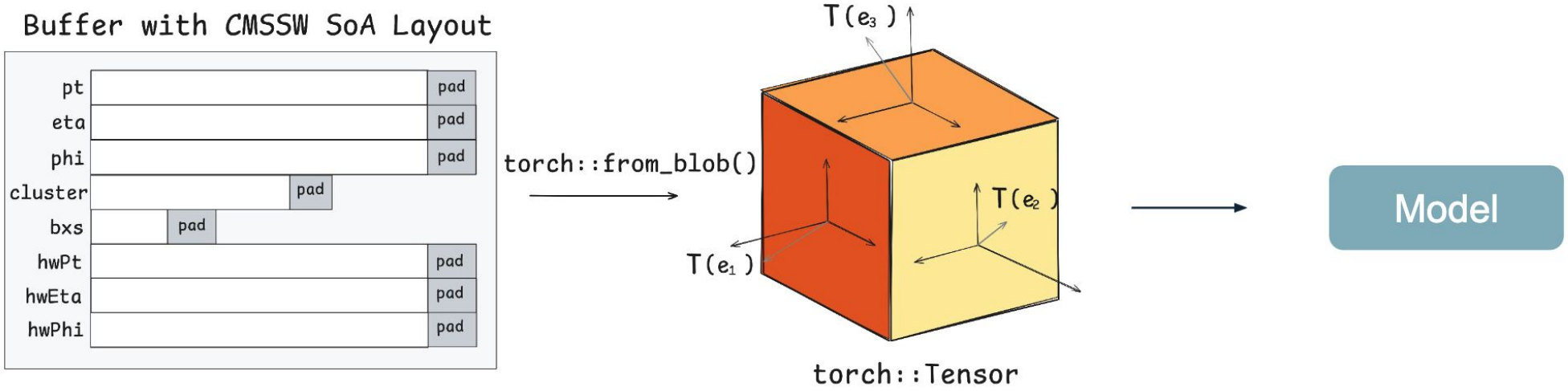
1. Define **integration** model for heterogeneous **alpaka and ML inference within CMSSW**:
   - Mapping PyTorch device constructs to Alpaka abstractions
   - Ensuring execution control and framework-aware resource usage
   - Shifting resource allocation responsibility from PyTorch to CMSSW

2. Enable **efficient event data memory reuse** for inference workflows:
   - Providing a lightweight interface for structured data manipulation via memory layouts and SoA patterns
   - Avoiding unnecessary memory transfers (H2D, D2D, D2H) during inference
   - Automating stride computation and tensor view definitions over pre-allocated memory blobs

Wrapping of SoA Structure

- **Machine learning data** stored **in SoA-based structure**, with element features as columns
  - Column-Major Stride
  - Eigen and Scalar supported
  - Datatype per column
  - Consistent element count
- PyTorch expects tensors in the form of ***Element* x *Feature***
- ***torch::from_blob*** builds **non-owning tensors** from existing data structures ➟ SoAs as basis

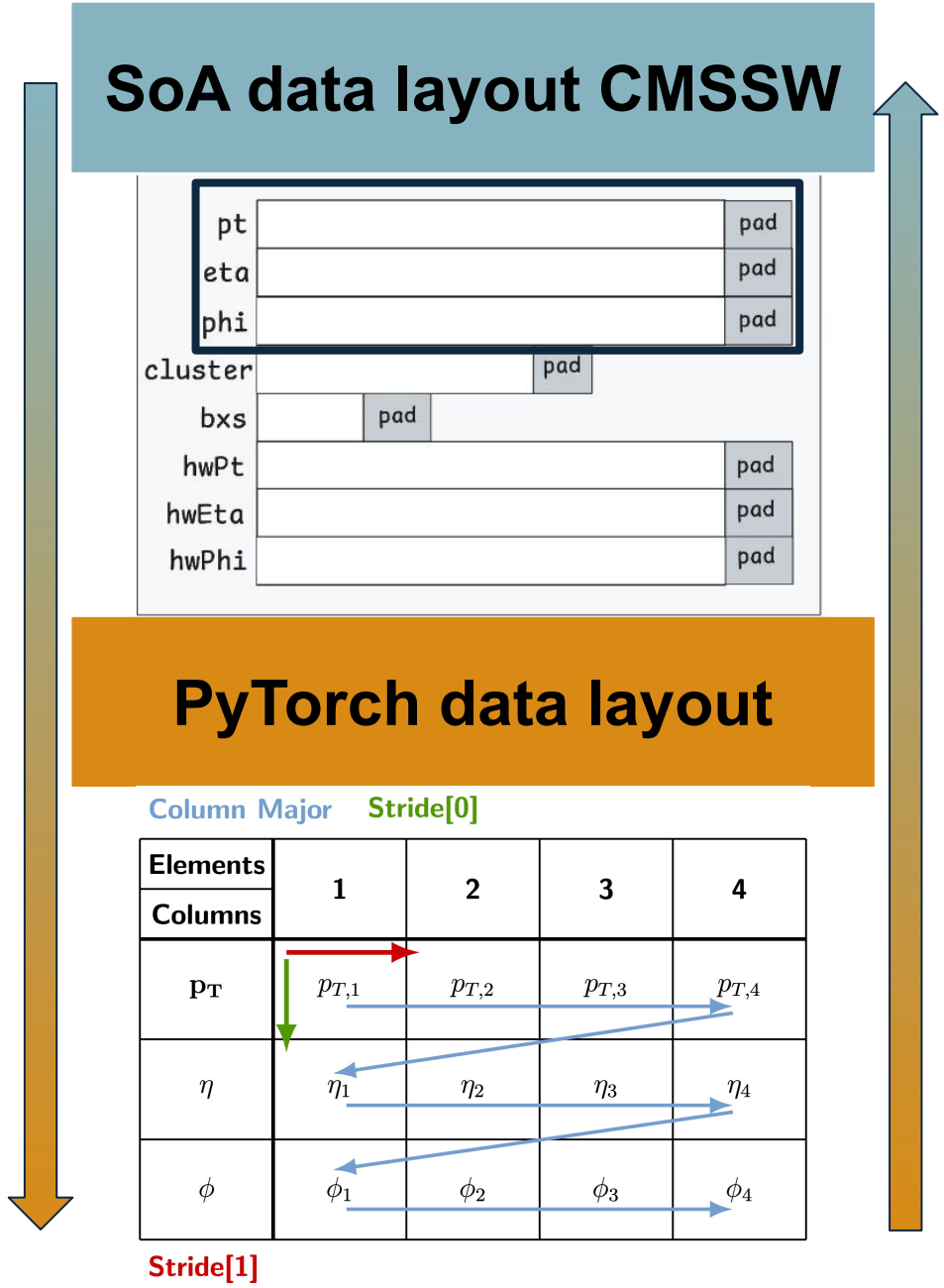
Columns of CMSSW's SoAs have the same #elements (except scalars) but can have different datatypes



Tensor Manipulation

- **PyTorch can adopt GPU memory directly**, provided the layout and alignment are known at compile time
- **Complexity** is **hidden behind** a streamlined **interface** built on CMSSW alpaka-ready collections

- Calculates **Stride and Size for each Block**
  - Converts raw **byte-based information to elements**
  - Memory layout is **padded to alignment boundaries**
  - Stride for Columns: *(1, #alignment)*
  - Automatic **wrapping for Eigen and Scalar** entries
  - **Data types** are **validated** at **compile time**
  - **Pointer validity** and positioning are verified at **runtime**



Example Usage

Retrieve input **SoAs from event** and **allocate output buffers** using node and edge size

Extract SoA records to **access** pointer locations to the **buffer** and their **datatype**

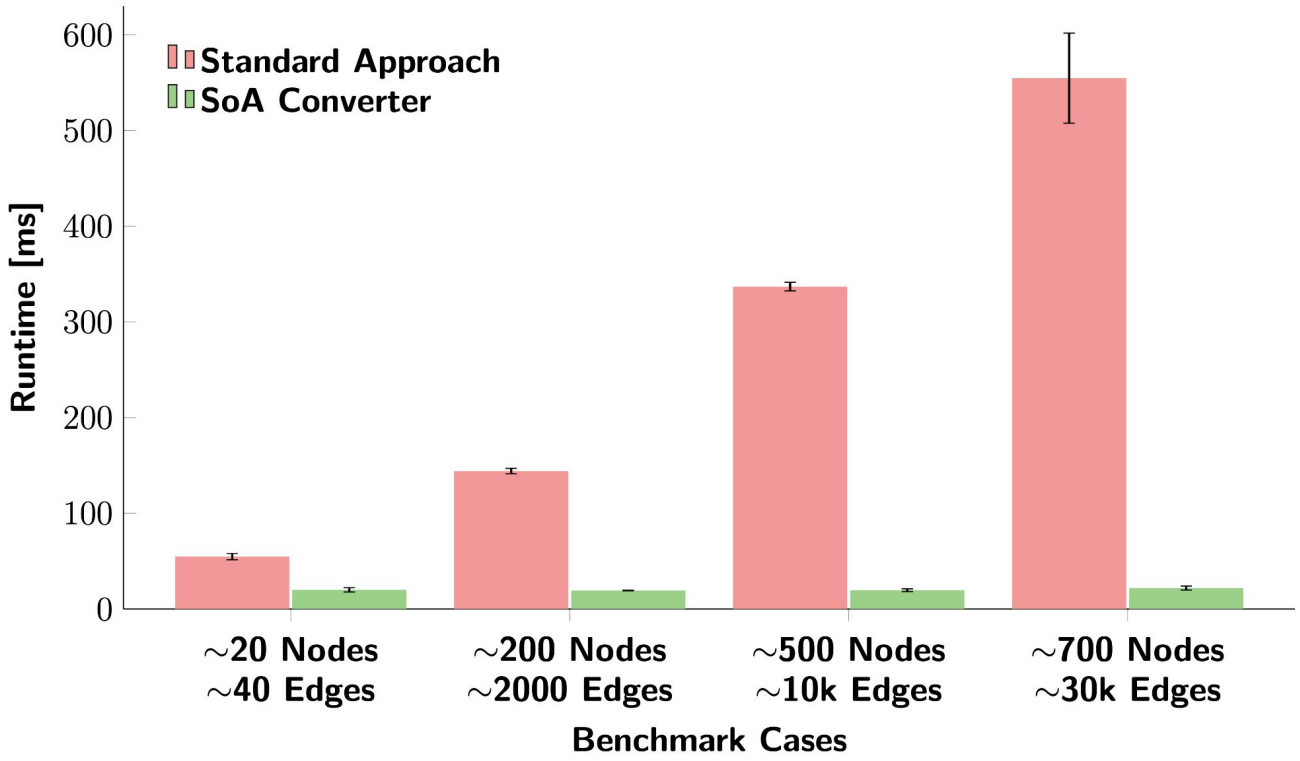
Collect columns from **different SoA-layouts** and append them to a ***TensorCollection***

Prepare output ***TensorCollection*** for inference results and **execute inference directly** on the queue

```
const auto &particles = event.get(particles_token_);
const auto total_size = particles.const_view().metadata().size();
auto regression_collection = SimpleNetDeviceCollection(event.queue(), total_size);


// SoA records
auto input_records = particles.const_view().records();
auto output_records = regression_collection.view().records();


// input tensor definition
TensorCollection<Queue> inputs(total_size);
inputs.add<ParticleSoA>("particles",
                        input_records.pt(),
                        input_records.eta(),
                        input_records.phi());


// output tensor definition
TensorCollection<Queue> outputs(total_size);
outputs.add<SimpleNetSoA>("regression_head", output_records.reco_pt());



model_.forward(event.queue(), inputs, outputs);
event.emplace(simple_net_token_, std::move(regression_collection));
```

Performance Results

- Benchmark using a single **NVIDIA H100 GPU** with 1 job and 1 stream
- Results **averaged over 100 runs**
- For **larger input sizes** costly memory **copies become** a significant **bottleneck**
- Up to **25 fold speedup** for largest benchmarked input size



Comparing the execution time on the GPU with and without the direct-input reading mechanism. In the standard approach, the data is read from the GPU, formatted for PyTorch, and moved back to the GPU for the model execution

Model Compilation for Low-Latency Inference

Just In Time

- Compiles models dynamically at runtime.
- Ideal for flexible development and quick iteration.
- Supports on-the-fly execution without build-time preparation.
- Fully supported by wrapper interface
- Unfeasible to use in a multithreaded environment

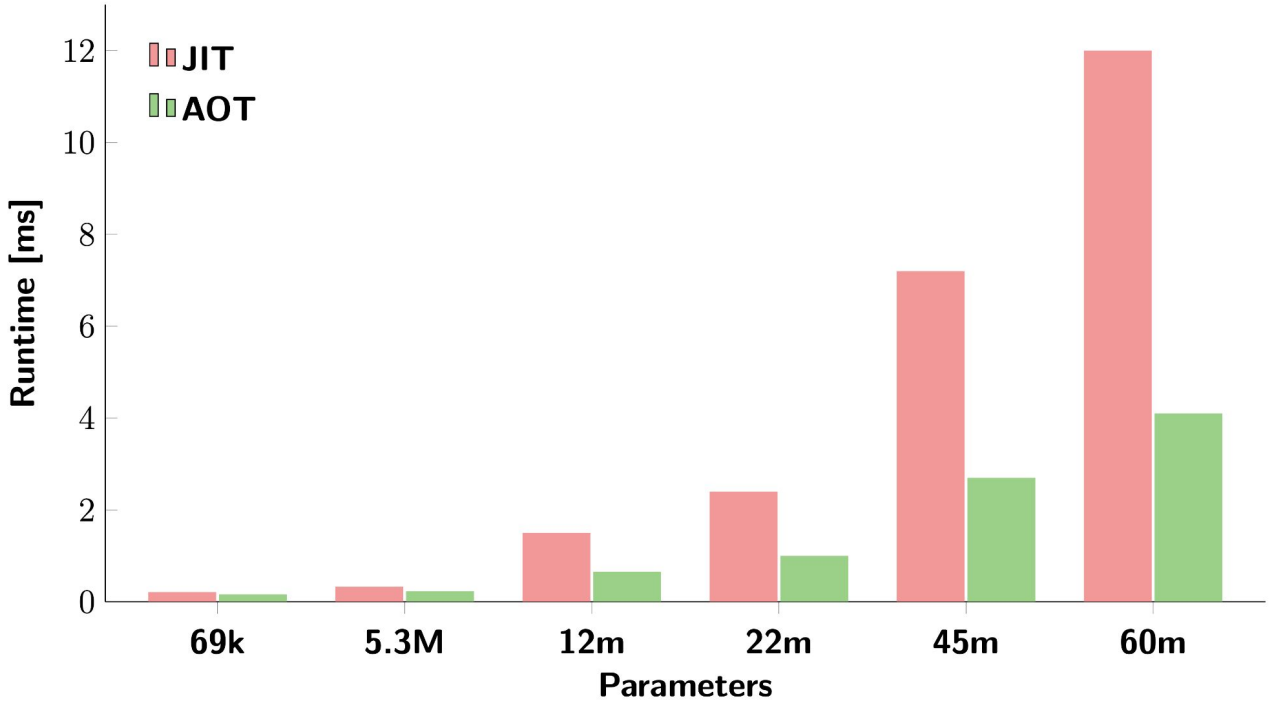
Ahead Of Time

- Uses *TorchInductor* and *Triton* to compile models into optimized binaries for specific hardware (e.g., NVIDIA GPUs).
- Targets specific hardware (e.g., NVIDIA GPUs) with precompiled kernels.
- Produces shared library, loaded at runtime using standard C++ interfaces.
- **Future work** to fully include in CMSSW

AOT should be used for production due to reduced runtime overhead, reproducibility, and maximum performance

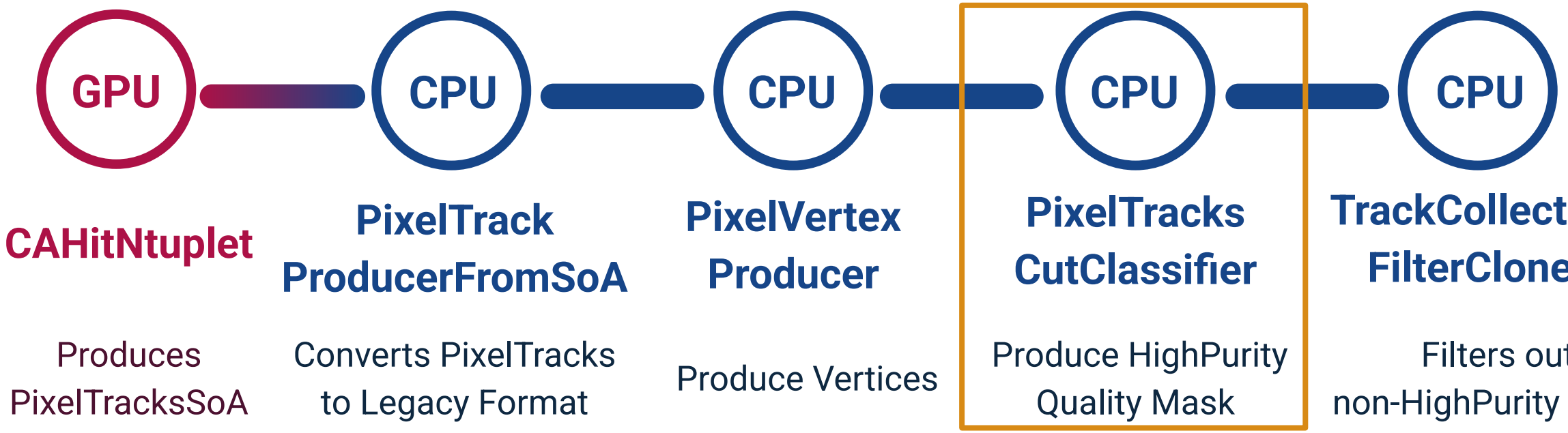
Model Compilation for Low-Latency Inference

- Benchmark using a single **NVIDIA T4 GPU** with 1 job and 1 stream
- Results **averaged over 100 runs**
- **Speedup using AOT** is especially **significant** for **larger models**
- Increasing the number of runs will gradually reduce the gap, since JIT optimizes the execution graph over time.



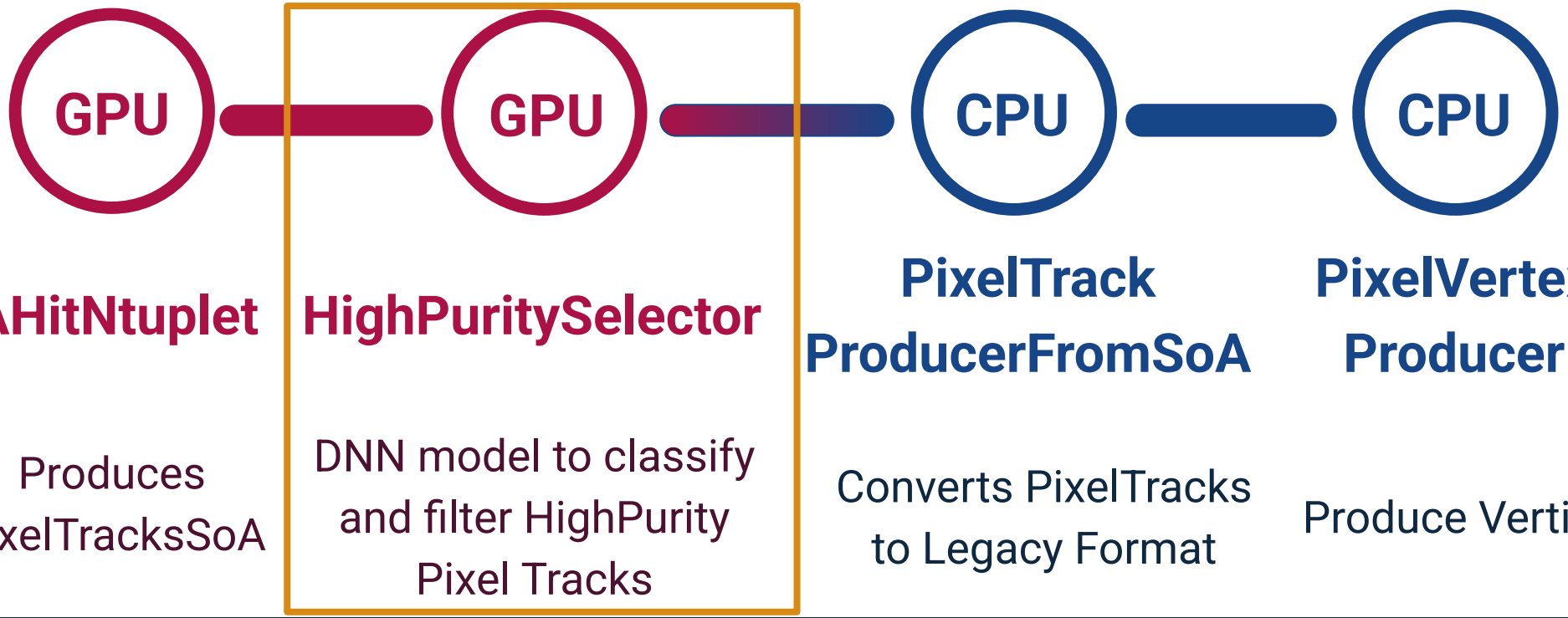
AOT increases the integration complexity of the model within the CMSSW workflow, but its improved reproducibility, compatibility with highly multithreaded environments, and performance gains make it a valuable step forward

Current HLT Phase-2 Pixel Tracks Sequence



Replace cut-based classifier by a DNN model to improve selection performance

Studied HLT Phase-2 Pixel Tracks Sequence



- **Successfully tested and benchmarked under production-like conditions**
- **Further developments of the Pixel Vertex Producer will allow to deploy the full sequence on GPU**

Conclusion

- Deliver an interface between **alpaka** and **PyTorch** to streamline **heterogeneous ML pipelines**.
- Exploiting **transparent PyTorch memory handling** ➟ seamless evaluation of ML models on data products already on the GPU without copies.
- Model outputs can be used for the next chain in the alpaka processing ➟ **no D2D, D2H, H2D copies!**
- Tested **AOT compilation of PyTorch models** ➟ beta version successfully for demonstration purposes
- First promising results for **HLT Phase-2 Pixeltracking** of the **direct ML pipeline** for future usage in production

Acknowledgements

This work has been partially funded by the Eric & Wendy Schmidt Fund for Strategic Innovation through the CERN Next Generation Triggers project under grant agreement number SIF-2023-004.


CERN
NexTGen
Next Generation Triggers