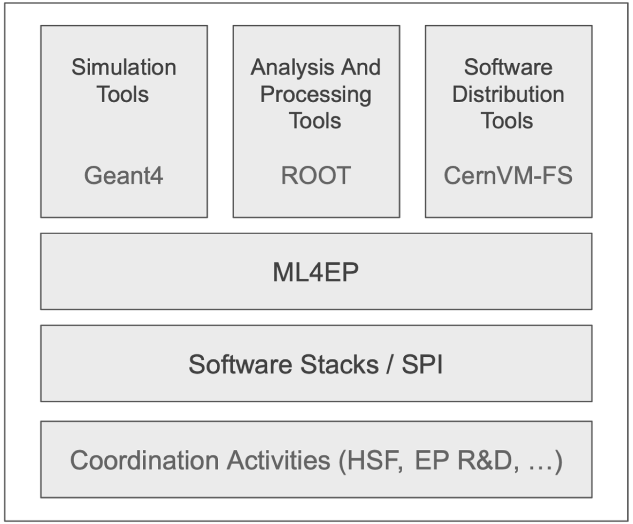
Machine Learning Activities in SFT

Lorenzo Moneta

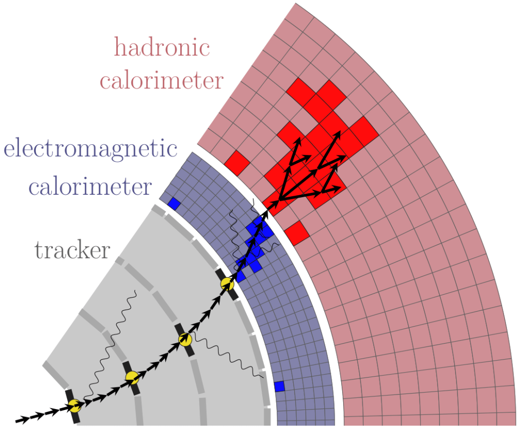
Introduction

- New project in SFT for common ML activities
  - **ML4EP**: provide service and support to the experiment on common ML issues
- Initiated by building on existing ML activities:
  - **ML for fast simulation**
  - ML software in ROOT
    - **SOFIE** (ML inference)
    - **Batch generator**

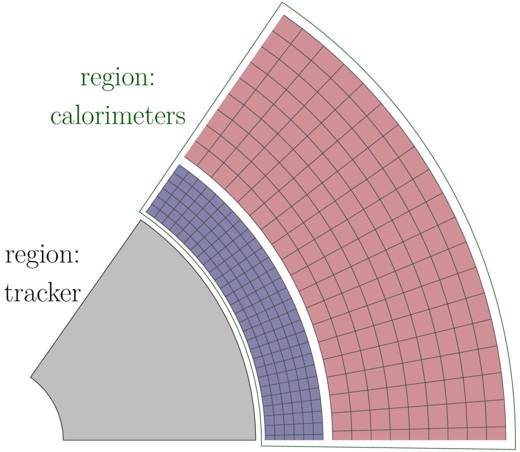


Stakeholders
ALICE
ATLAS
CMS
LHCb
EP R&D
IT projects
HSF
FCC
…

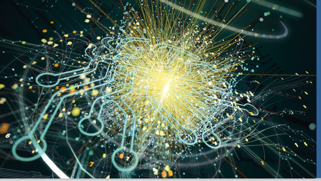
Fast Simulation Activities



FullSim



FastSim



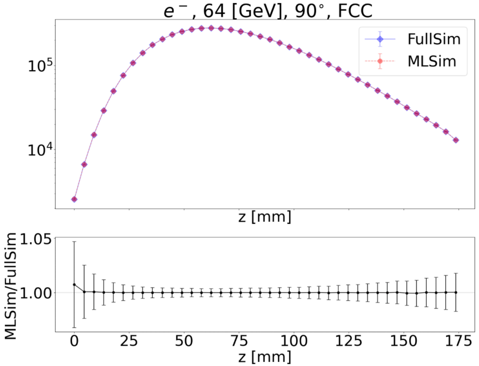
Fast Simulation Activities

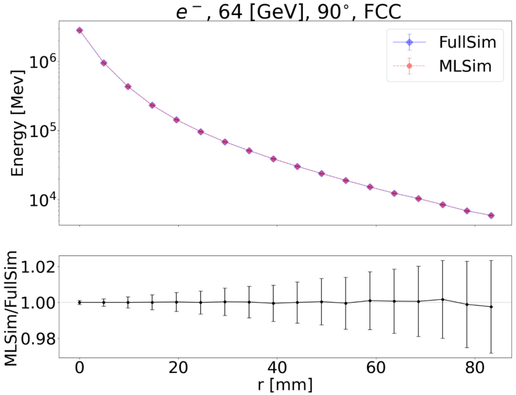
ML Models developed in CERN EP-SFT group

(*Anna Zaborowska, Piyush Raikwar, Peter Mckeown, Renato Paulo Da Costa Cardoso*)

Variational Autoencoder

- published with Geant4 release in example Par04
- small and quick to train
- reproduces well average shower variables (total energy, profile and moments)
- but blurry deposits

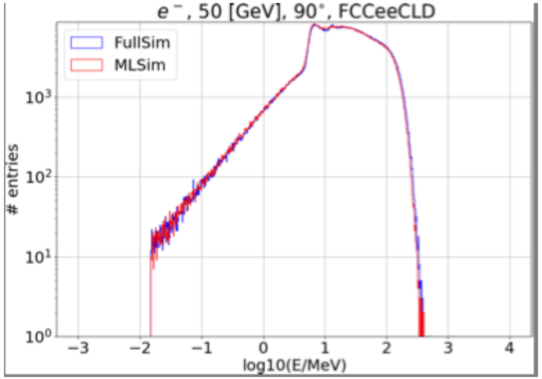


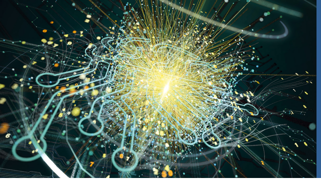


Fast Simulation Activities

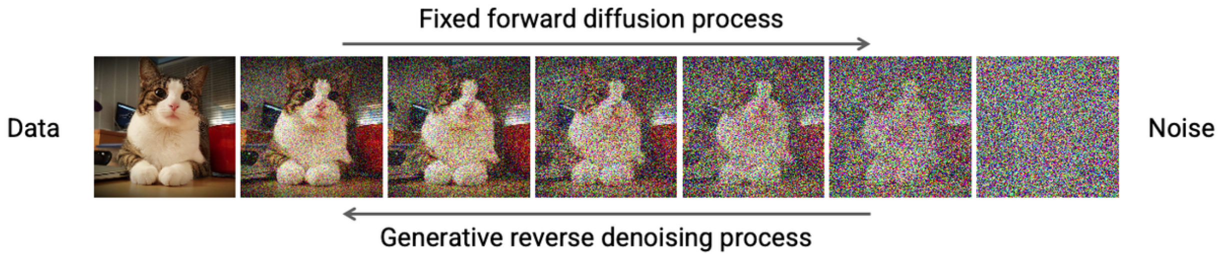
Transformer based models

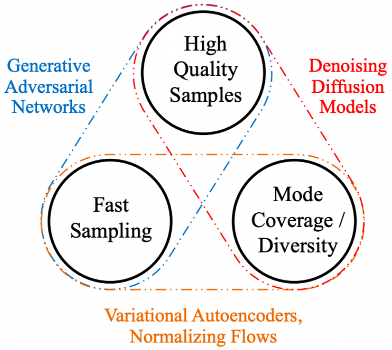
- focusing on modelling well cell-level variables as well as exploring generalisation power to multiple detectors
- Vector-quantized VAE + autoregressive: (see [CHEP 2023] presentation)
- More promising diffusion model, **CaloDiT** (see [ACAT 2024] presentation)





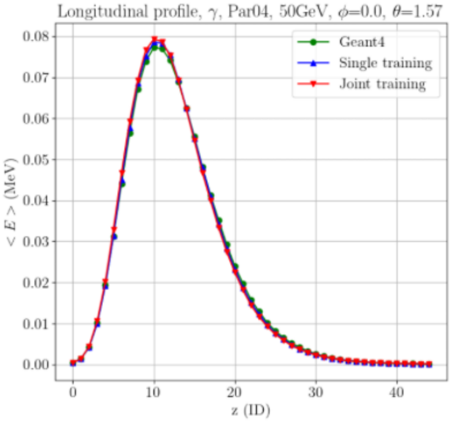
Diffusion Models for Simulation





Use of diffusion models for simulation

- Investigating transformer based architecture
  - A generalised architecture working with any type of data
  - Modelling long-range dependencies (attention mechanism)
- but longer time to evaluate the model

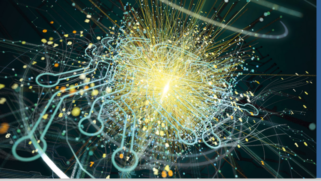


CaloDiT ACAT24

Plans for Fast Simulation Developments

- Continue development of transformer based models
  - aim to have best single-geometry diffusion model
  - work on inference optimisation
  - extend to different geometries and test its adaptation capabilities
- Work in collaboration with experiments
  - ATLAS: test VAE and transformer based modes
  - CMS: test transformer based model on HGCal
  - LHCb: develop best working model for hadronic showers
- Community effort: CaloChallenge and Open Data detector

Machine Learning Inference



Motivation

- **Fast Evaluation of Machine Learning models** is more and more relevant
- ML tools like Tensorflow/PyTorch have functionality for inference
    - can run only for their models
    - usage in a C++ environment can be cumbersome
    - require heavy dependence
- A standard for describing deep learning models:
    - ONNX ("Open Neural Network Exchange")
    - cannot describe all possible deep learning models (e.g. GNN) fully
- ONNXRuntime: an efficient inference engine based on ONNX
    - can work in both C++ and Python
    - supporting both CPU and GPU
    - can be challenging to integrate in the HEP ecosystem
        - control of threads, dependencies, etc..
        - not optimised for single-event evaluation

Idea for Inference Code Generation

- An inference engine that…
  - **Input**: trained ONNX model file
    - Common standard for ML models
    - Supported by PyTorch natively
    - Converters available for Tensorflow and Keras
  - **Output**: Generated C++ code that hard-codes the inference function
    - Easily invokable directly from other C++ project (plug-and-use)
    - Minimal dependency (on BLAS only)
    - Can be compiled on the fly using Cling JIT

- **SOFIE : S**ystem for **O**ptimised **F**ast **I**nference code **E**mit

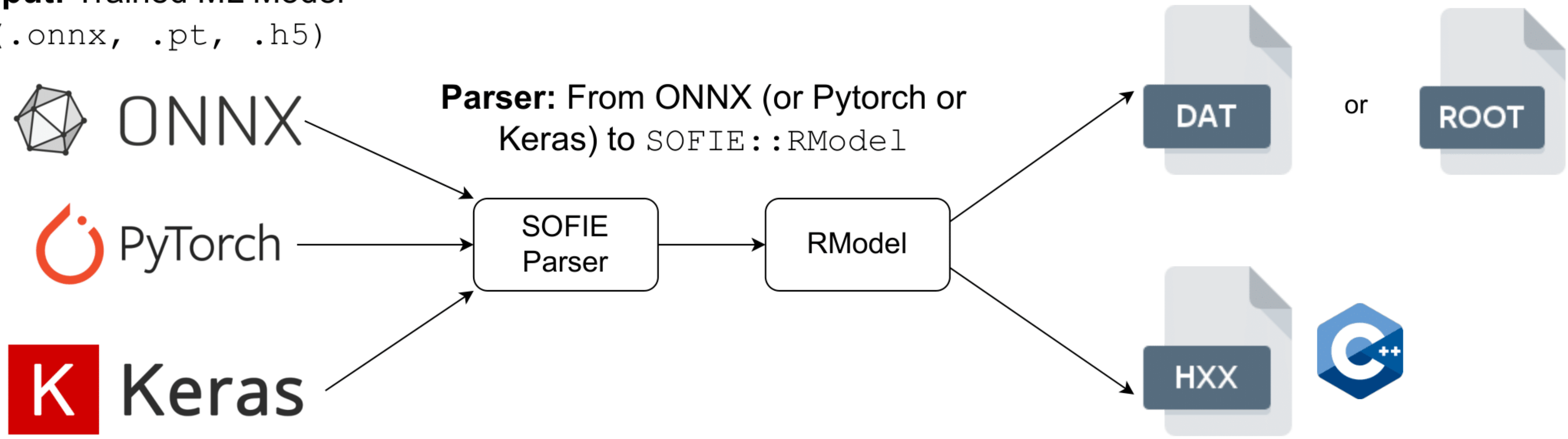
SOFIE

Outputs

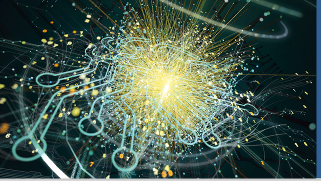
1. Weight File

Input: Trained ML Model

`(.onnx, .pt, .h5)`



2. C++ header file



Code Generation

- **Parser**: from ONNX to `SOFIE::RModel` class
  - `RModel`: intermediate model representation

```
using namespace TMVA::Experimental::SOFIE;
RModelParser_ONNX parser;
RModel model = parser.Parse("Model.onnx");
```

- **Code Generation**: from `RModel` to a **C++ file** (`Model.hxx`) and a weight file (`Model.dat` or `Model.root`)

```
// generate text code internally
model.Generate();
// write output header and data weight file
model.OutputGenerated();
```

Generated code has minimal dependency

- only linear algebra library (BLAS) and no ROOT dependency
- can be easily integrated in any project

C++ code

```
namespace TMVA_SOFIE_Model{

struct Session {

  Session(std::string filename) {
    ..........
  }
  std::vector<float> infer(float* input)
  {
     ..........
    //— implementation of all operators
    ..........

    return output_tensor;
   }
};
}
```

weight files

Using the Generated code: in C++

- SOFIE generated code can be easily used in compiled C++ code

```
#include "Model.hxx"
// create session class
TMVA_SOFIE_Model::Session ses("model_weights.dat");
//-- event loop
for (ievt = 0; ievt < N; ievt++) {
   // evaluate model: input is a C float array
   float * input = event[ievt].GetData();
   auto result = ses.infer(input);
   ….
}
```

1. include generated Model header file
2. Create session class (read weight data file)
3. Evaluate the model calling `Session::infer` function

See full Example tutorial code

Using the Generated code: in Python

- Code can be compiled using ROOT Cling and used in C++ interpreter or Python

```
import ROOT
# compile generate SOFIE code using ROOT interpreter
ROOT.gInterpreter.Declare('#include "Model.hxx"')
# create session class
s = ROOT.TMVA_SOFIE_Model.Session('model_weights.dat')
#-- event loop
.......
# evaluate the model , input can be a numpy array
# of type float32
  result = s.infer(input)
```

Compile at run-time SOFIE generated code using Cling

See full Example tutorial code

SOFIE Integration with RDataFrame

- **SOFIE Inference** code provides a Session class with this signature:

  `vector<float> ModelName::Session::infer(float* input);`

- **RDataFrame**(RDF) interface requires a functor with this signature:

  `FunctorObj::operator()(T x1, T x2, T x3,….);`

- Have a generic functor class adapting SOFIE signature to RDF: **`SofieFunctor<N,Session>`**
  - supporting multi-thread evaluation, using the RDF slots

```
ROOT::RDataFrame df("tree", "inputDataFile.root");
auto h1 = df.DefineSlot("DNN_Value",
SofieFunctor<7,TMVA_SOFIE_higgs_model_dense::Session>(nslots),
{"m_jj", "m_jjj", "m_lv", "m_jlv","m_bb","m_wbb","m_wwbb"}).
Histo1D("DNN_Value");
h1->Draw();
```

See full Example tutorial code in C++ or Python

GPU Extension of SOFIE

- Extend SOFIE functionality to produce GPU code using SYCL

```
// generate SYCL code internally
model.GenerateGPU();
// write output header and data weight file
model.OutputGeneratedGPU();
```

model.hxx

```
namespace TMVA_SOFIE_Linear_event{

struct Session {

Session(std::string filename ="") {
   if (filename.empty()) filename =
"Linear_event.dat";
   std::ifstream f;
   f.open(filename);
   // read weight data file
   ………………..
}
std::vector<float> infer(float*
tensor_input1){
```

with SYCL code

```
#include "Model.hxx"
// create session class
TMVA_SOFIE_Model::Session ses("model_weights.dat");
//-- event loop
for (ievt = 0; ievt < N; ievt++) {
   // evaluate model: input is a C float array
   float * input = event[ievt].GetData();
   auto result = ses.infer(input);
   …..
}
```

Inference code needs to be linked against oneAPI MKL libraries and compiled using SYCL compiler

GPU Implementation

Performance considerations

- **Minimise overhead of data transfers** between host and device
  - implement all on GPU and transfer data only at the beginning and at the end of the computation
- **Manage buffers efficiently**, declaring them at the beginning
- Use libraries for **GPU Offloading**:
  - GPU BLAS implementation from Intel oneAPI and portBLAS for other GPUs
- **Fuse operators** when possible (e.g. a layer op. with activation) in a single kernel
- **Replace conditional check** with relational functions
  - ensure work items do not execute different paths

ONNX Supported Operators

Operators implemented in ROOT	CPU	GPU
Perceptron: Gemm	✓	✓
Activations: Relu, Selu, Sigmoid, Softmax, Tanh, LeakyRelu	✓	✓
Convolution (1D, 2D and 3D)	✓	✓
Recurrent: RNN, GRU, LSTM	✓	
Pooling: MaxPool, AveragePool, GlobalAverage	✓	✓
Deconvolution (1D,2D,3D)	✓	✓
Layer Unary operators: Neg, Exp, Sqrt, Reciprocal, Identity	✓	✓
Layer Binary operators: Add, Sum, Mul, Div	✓	✓
Reshape, Flatten, Transpose, Squeeze, Unsqueeze, Slice, Concat, Reduce, Gather	✓	✓
BatchNormalization, LayerNormalization	✓	✓
Custom operator	✓	

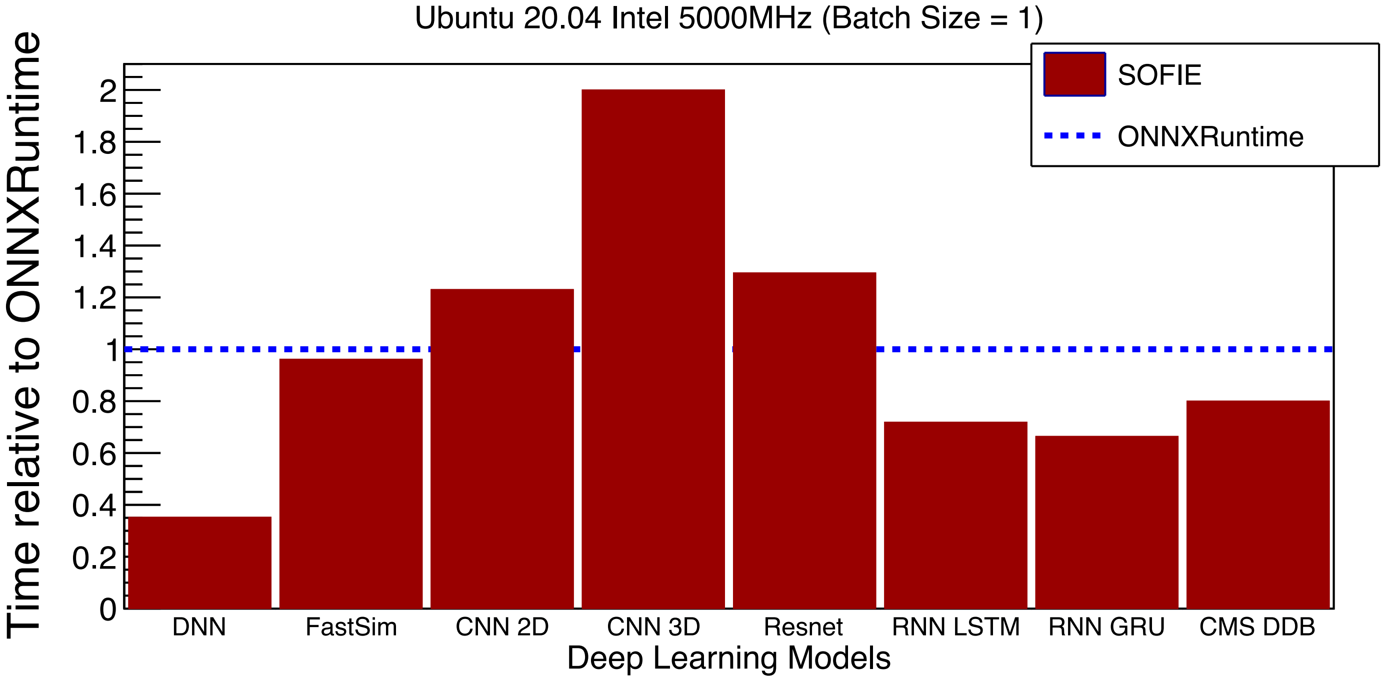
- current CPU support available in ROOT 6.30
- GPU/SYCL is implemented in a [ROOT PR]

CPU Benchmark for Different Models

- Test CPU event performance of **SOFIE** vs **ONNXRuntime**

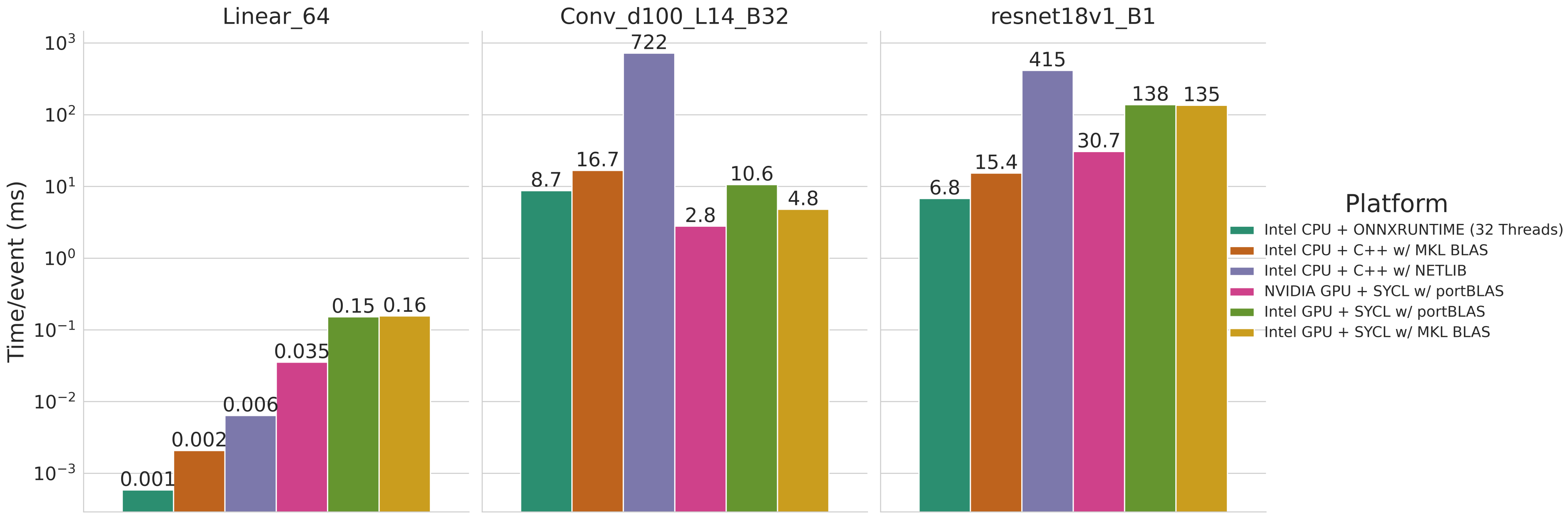
(using batch size = 1)

Smaller = Better

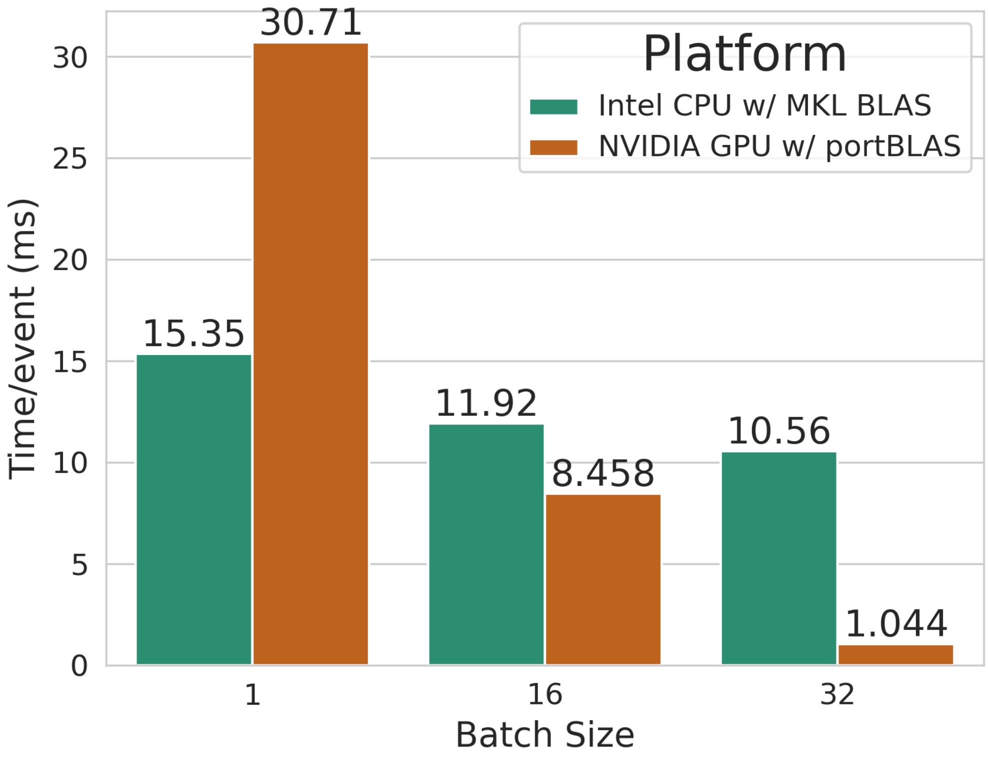


CERN

Performance on CPU vs GPU

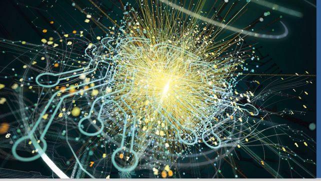


Performance on GPU vs CPU (ResNet)



Using **ResNet model**
(rather heavier model,
> 10 conv. layers with images sizes ~ 200x200)

Varying Batch size



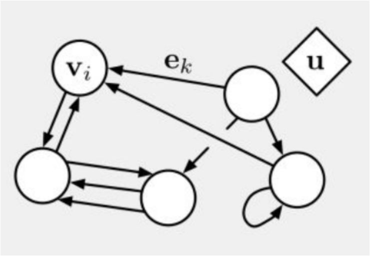
SOFIE for Graph Networks

Added SOFIE **support GNN models**

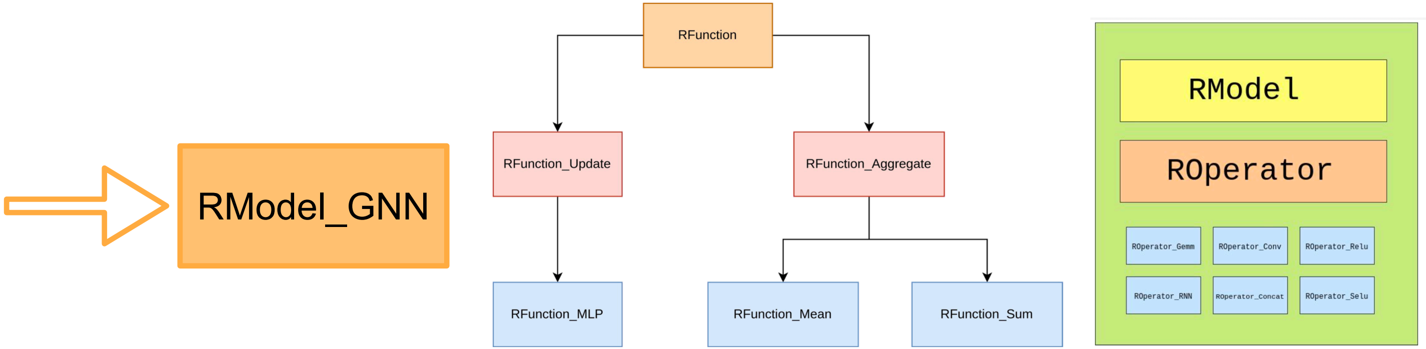
- Initiated with a network developed by LHCb:
  - Message Passing GNN built and trained using the DeepMind's **Graph Nets** library
    - model plan to be used in LHCb trigger using full event interpretation (*see ACAT2024* contribution)
    - important to have efficient implementation and with minimal dependencies
- **Available now in ROOT** from version 6.32
  - supporting a dynamic number of nodes/edges

SOFIE GNN Support

- Developed **C++ classes** for representing **GNN structure**.
  - based on SOFIE **`RModel`** and the **`ROperator`** classes developed for supporting ONNX.
  - SOFIE classes provide the functionality to generate C++ inference code
- **Python code** (based on PyROOT) for initialising SOFIE classes from the Graph Nets models



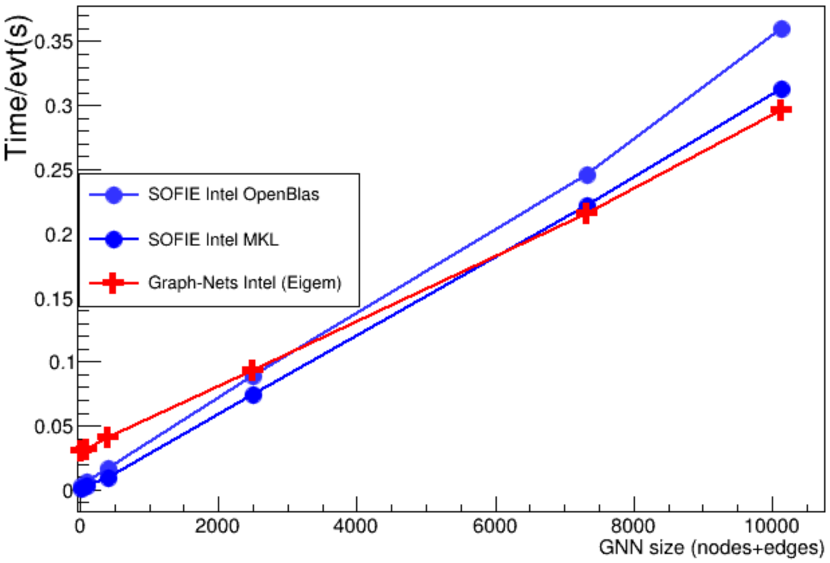
Graph Nets GNN



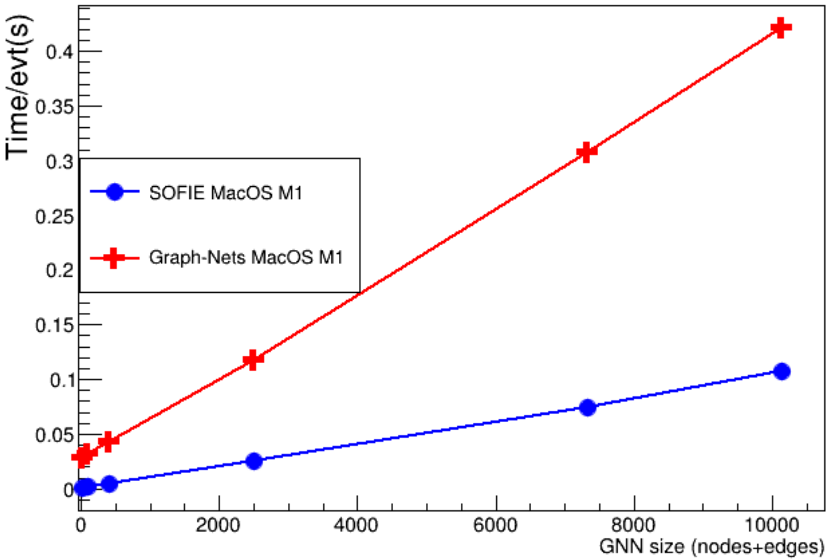
Benchmark of SOFIE GNN

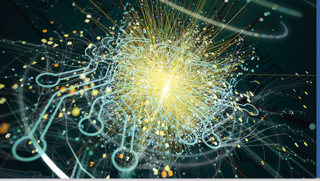
- Test inference performance of a toy architecture from LHCb
    - scaling number of nodes and edges

Intel Linux Desktop



MacOS M1





Future Steps

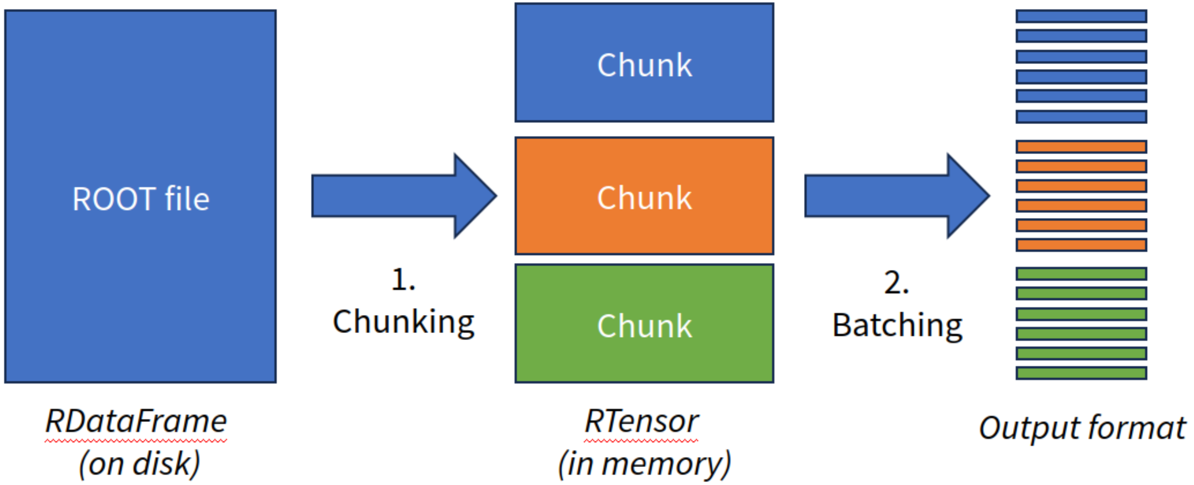
- **Integrate SOFIE in fast simulation pipelines**
  - supporting first VAE model
  - looking also at FastCaloGAN in ATLAS
- Future developments (e.g. new operators) according to **user needs** and the **received feedback**
  - starting developments to support transformer models
- Continuing the **support for different types of GPUs**
  - plan to extend to ALPAKA (used by CMS) given some interest to deploy SOFIE in GPU-based trigger systems
- Want to support inference for ML models of the experiments in cases that are difficult to implement or require heavy dependencies
  - don't want to compete with existing industry tools for training
- Develop a **complete benchmark** (CPU time and memory) for models used by experiments and fast simulations
  - will guide experiments to choose the optimal tool for their used models

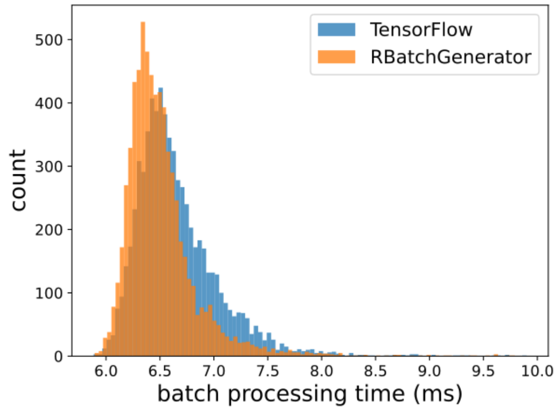
Other Activities

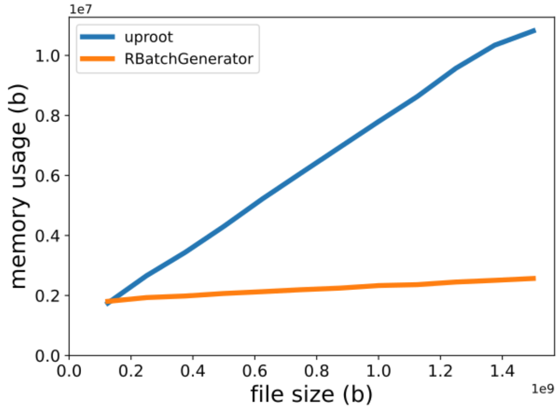
RBatchGenerator: Batching ROOT files

Serving tensors to ML training pipelines (ongoing R&D)

- **Generate batches directly from a ROOT file**
- As fast as traditional ML software
- Scales to very large file sizes
- Easy to add to workflow



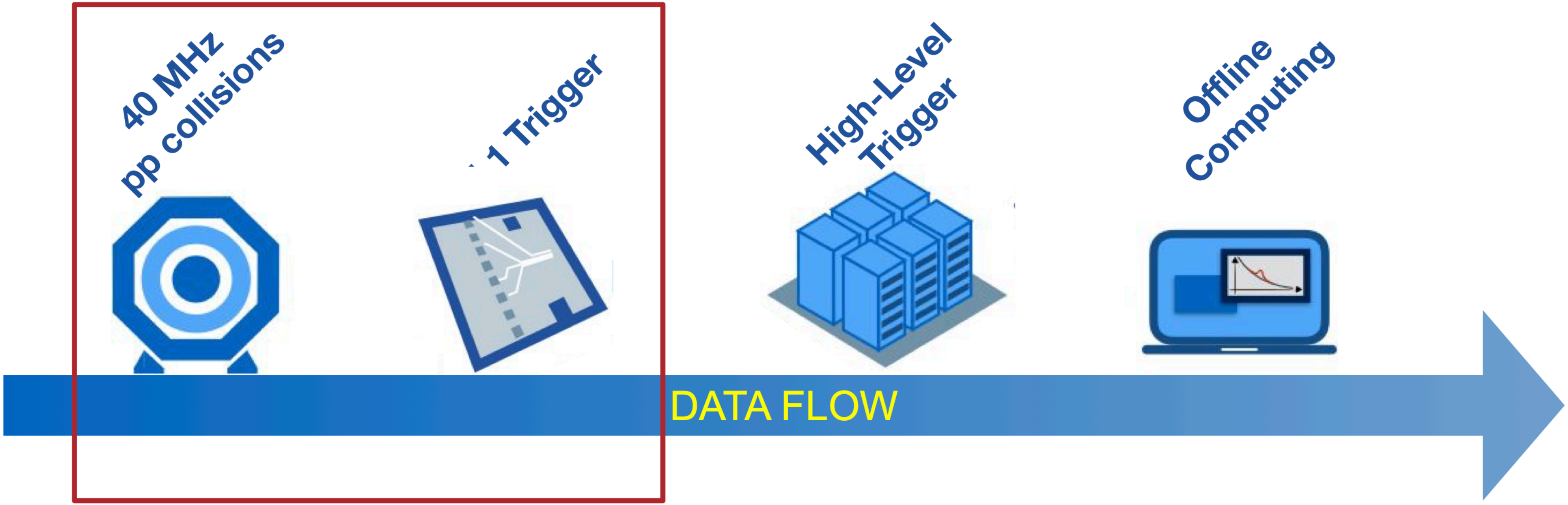




Next Gen Trigger Activities

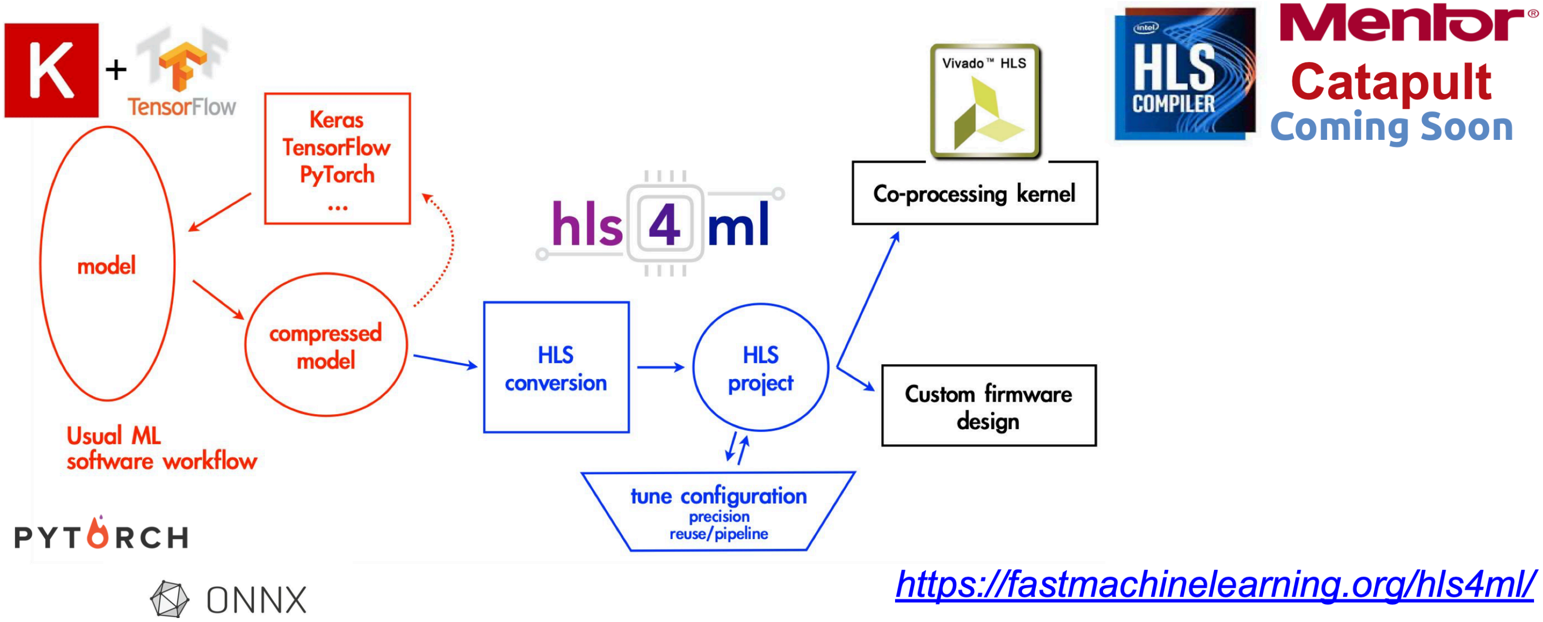
- SFT is hosting common activities of Next Gen Trigger projects
  - Work on tools such as **hls4ml** (for DL) and **Conifer** (BDT) to develop ML to FPGA model synthesis tools, addressing the needs of the experiments.
  - Develop the software infrastructure needed to enable **hardware-aware neural network training workflows**. This work will enable the development and deployment of hardware-optimal AI-based real-time algorithms.

LHC Experiment Data Flow



- ML in trigger and sensor applications must be implemented in FPGAs or custom ASICs
- Must be robust to noise and radiation and meet high throughput low latency requirements

high level synthesis for machine learning



https://fastmachinelearning.org/hls4ml/

from V. Lonchar at 24th IEEE Real-Time Conference

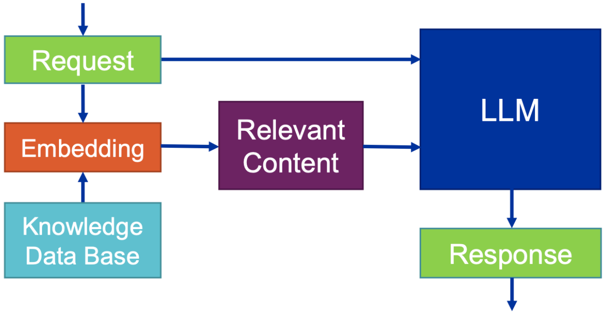
Using Large Language Models (LLM)

AccGPT: A CERN Chatbot

- aim to be better than ChatGPT for specific CERN use case
- being developed in collaboration between CERN IT and ATS

The AccGPT pipeline:

- Retrieval Augmented Generation (RAG).



F. Rehm (Applied AI WS)

Based on two models:

1. Embedding model:
   - A pretrained open source model (e5-large).
   - Retrieves „relevant content“ from database.
2. Large Language Model (LLM):
   - A pretrained open source GPT model (LLaMA 2 13B).
   - Formulates responses using the „relevant content“.

Accompanied by a self-created knowledge data base.

last IML meeting (April, 9) dedicated to LLM

Summary

- AI/ML is fundamental for experiments
- New ML4EP project provides a place for **sharing common AI/ML expertise** within SFT and its stakeholders
  - Avoiding duplicating efforts
  - Can focus on supporting main activities and integrate new ones (e.g HLS4ML funded by NGT project)
  - Will foster the collaboration with IT and the AI/ML group of ATS

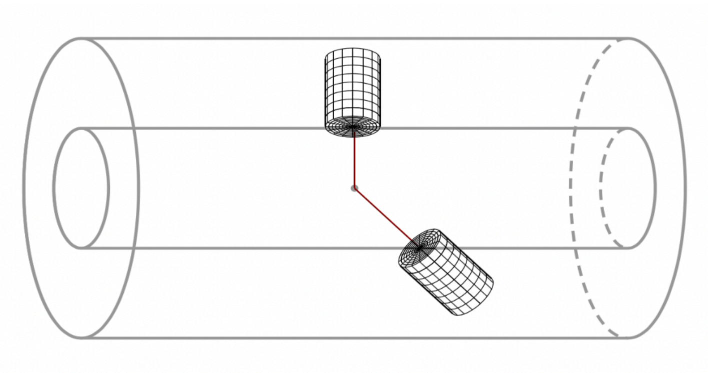
Backup Slides

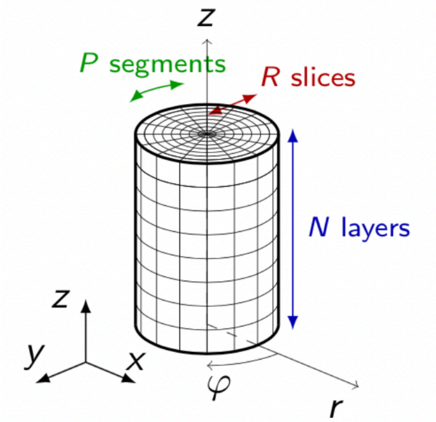
SOFIE: Example Notebooks and Tutorials

- Example notebooks on using SOFIE:
  - https://github.com/lmoneta/tmva-tutorial/tree/master/sofie
- Tutorials are also available in the tutorial/tmva directory
- Link to SOFIE code in current ROOT master in GitHub
- Link to PR implementing SOFIE to SYCL code generation
- Link to benchmarks in *rootbench*

Dataset

We utilize a dataset similar[1] to “CaloChallenge Dataset 3”. (Talk at CHEP’23)





$r \times \varphi \times z = 18 \times 50 \times 45$

For the shown preliminary results, we use the following subset (~100k samples):

- Angle of incident e^- = 70°, 80°, 90°
- Energy of incident e^- = 64, 128, 256 GeV
- Sampling calorimeter with silicon and tungsten layers[2] (SiW)

from P. Raikwar (CHEP 2023)

[1]More incident angles and discrete energy spectrum
[2]Layer thickness: 0.3 mm + 1.4 mm for Si & W respectively