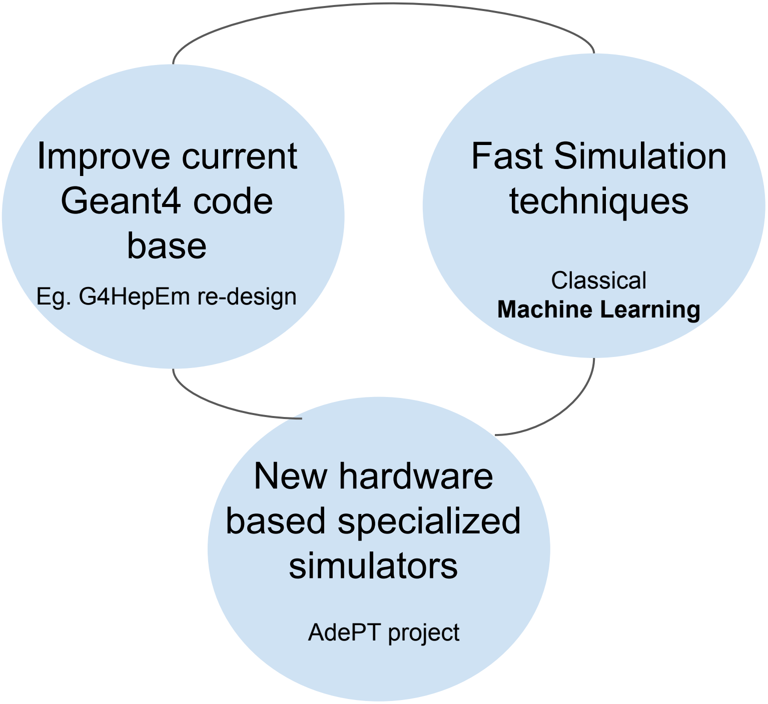
Multi-detector geometry modeling and Geant4 Integration

Anna Zaborowska, Witold.Pokorski & Dalila Salamani
CERN EP-SFT

This work benefited from support by the CERN Strategic R&D Programme on Technologies for Future Experiments (CERN-OPEN-2018-006)

ML4Jets Workshop 07/07/2021

Geant4 simulation R&D activities



How to fast simulate particles in Geant4?

Improve current Geant4 code base

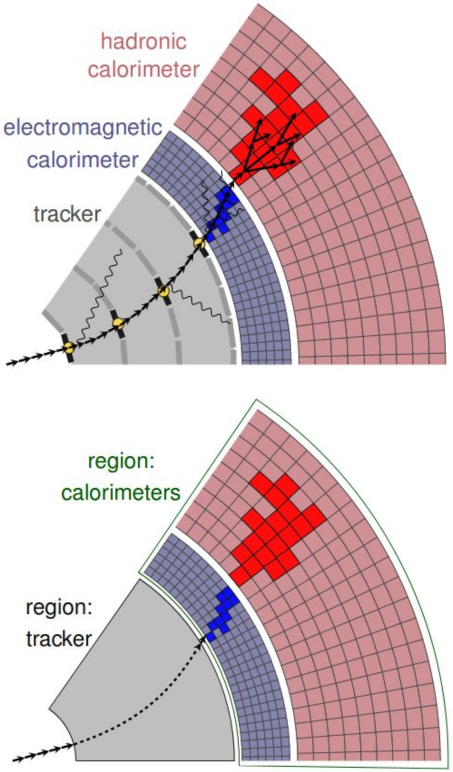
Eg. G4HepEm re-design

Fast Simulation techniques

Classical
Machine Learning

New hardware based specialized simulators

AdePT project

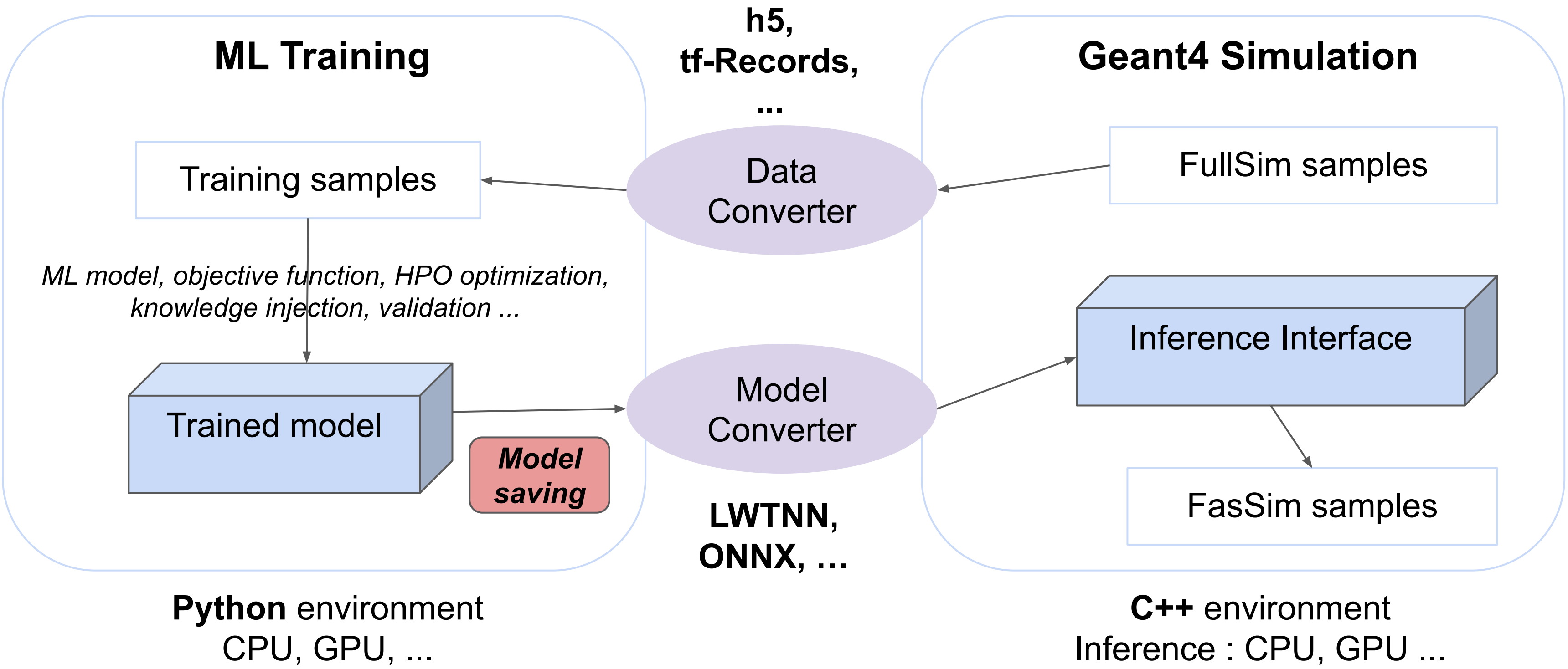


FullSim

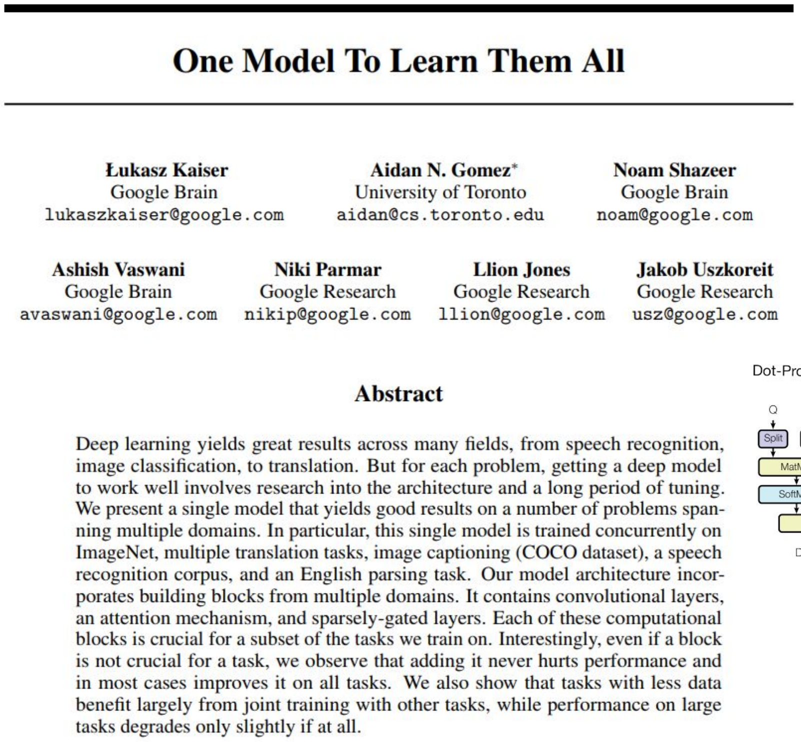
FastSim
GFlash, MLSim

Figures from Geant4 Advanced Course, Anna Zaborowska

From ML training to Geant4 fast simulation



One model to learn from different tasks/domains ?



One Model To Learn Them All

Łukasz Kaiser
Google Brain
lukaszkaiser@google.com

Aidan N. Gomez*
University of Toronto
aidan@cs.toronto.edu

Noam Shazeer
Google Brain
noam@google.com

Ashish Vaswani
Google Brain
avaswani@google.com

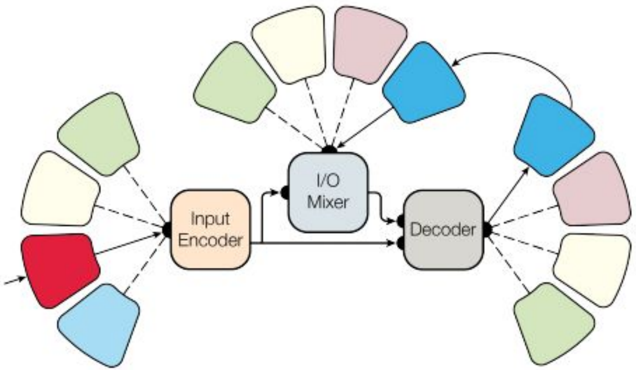
Niki Parmar
Google Research
nikip@google.com

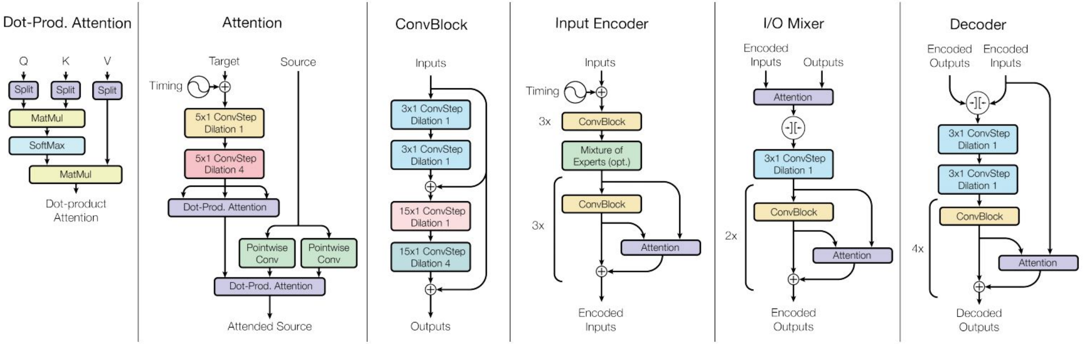
Llion Jones
Google Research
llion@google.com

Jakob Uszkoreit
Google Research
usz@google.com

Abstract

Deep learning yields great results across many fields, from speech recognition, image classification, to translation. But for each problem, getting a deep model to work well involves research into the architecture and a long period of tuning. We present a single model that yields good results on a number of problems spanning multiple domains. In particular, this single model is trained concurrently on ImageNet, multiple translation tasks, image captioning (COCO dataset), a speech recognition corpus, and an English parsing task. Our model architecture incorporates building blocks from multiple domains. It contains convolutional layers, an attention mechanism, and sparsely-gated layers. Each of these computational blocks is crucial for a subset of the tasks we train on. Interestingly, even if a block is not crucial for a task, we observe that adding it never hurts performance and in most cases improves it on all tasks. We also show that tasks with less data benefit largely from joint training with other tasks, while performance on large tasks degrades only slightly if at all.

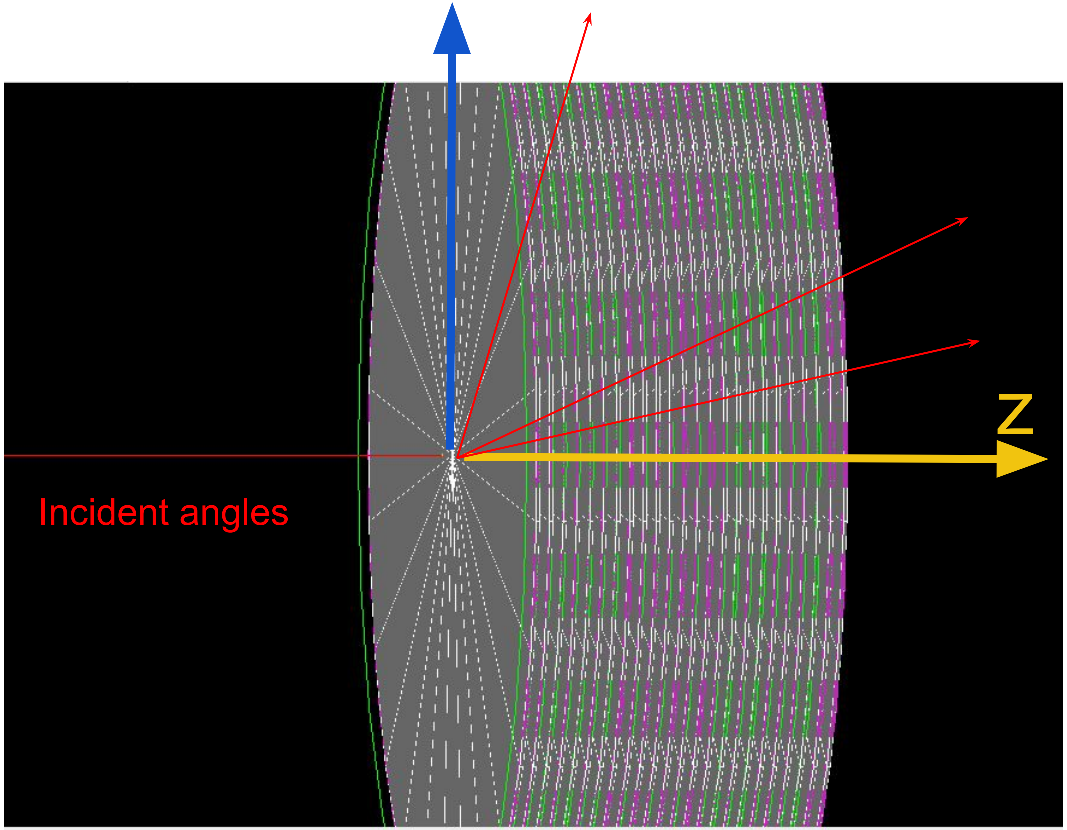




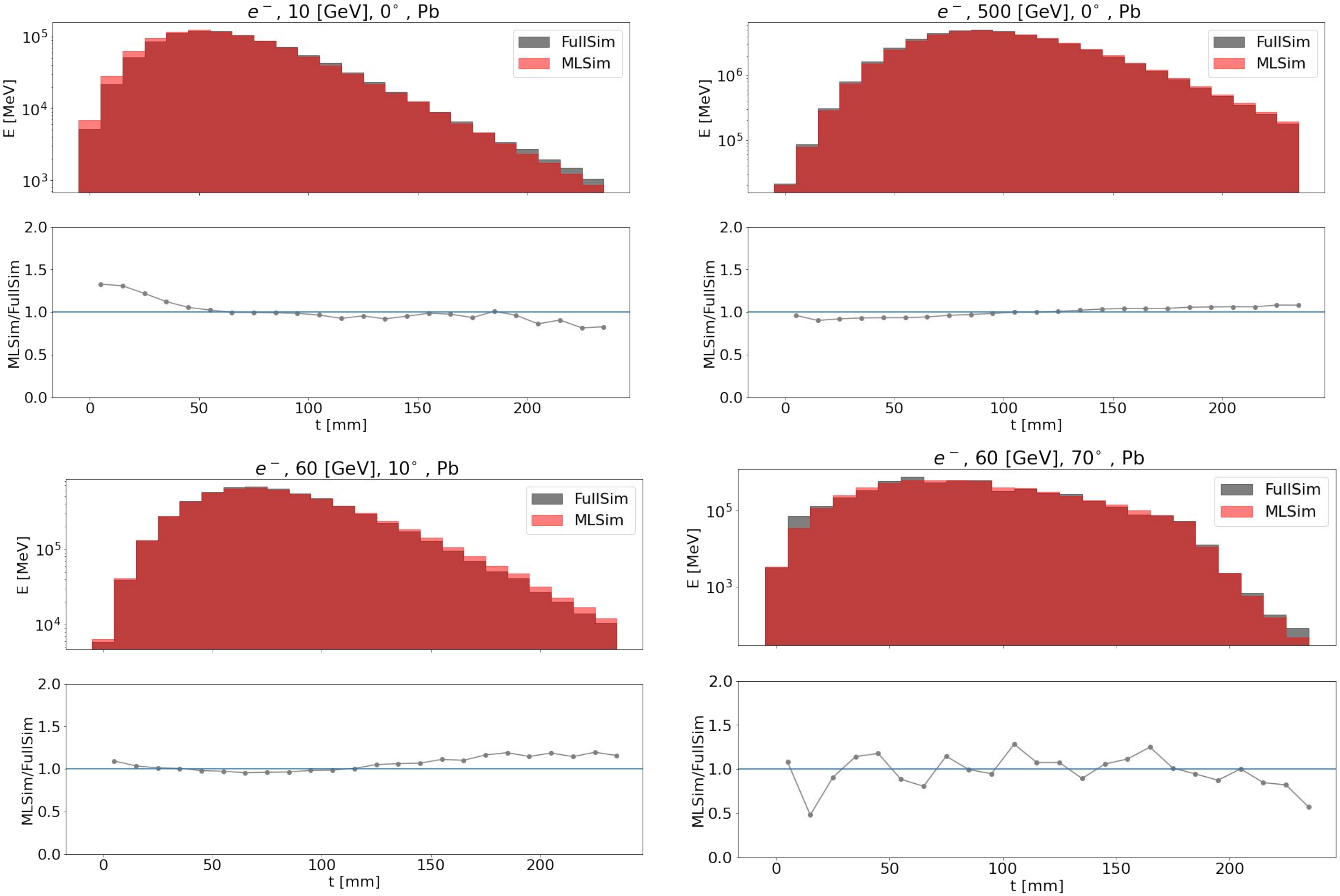
Geant4 samples

- 3D readout geometry, electromagnetic calorimeters
  - Lead tungstate ($PBWO_4$)
  - Silicon-tungsten (SiW)
- Flat energy samples 1-500 GeV
- Incident angle from 0° to 90°

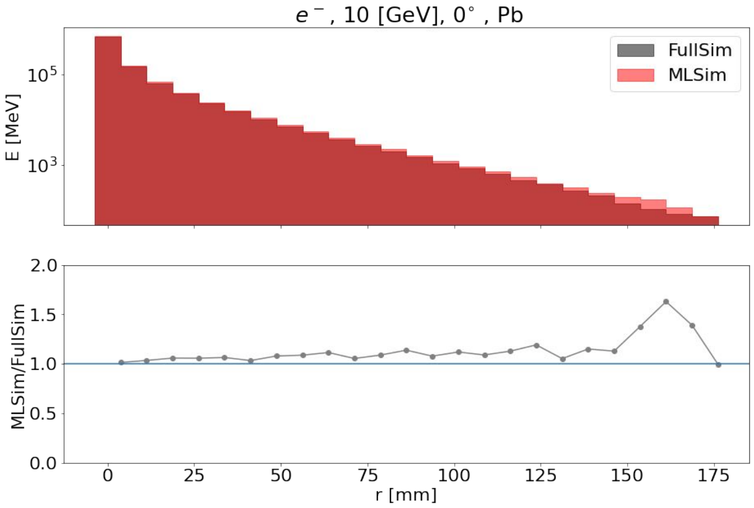
$$P\,(\,shower\,|\,energy\,,\,incident\,angle\,,\,geometry)$$

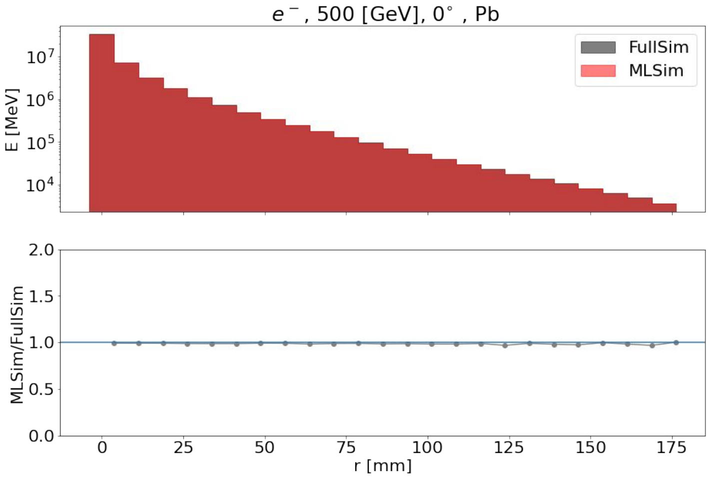


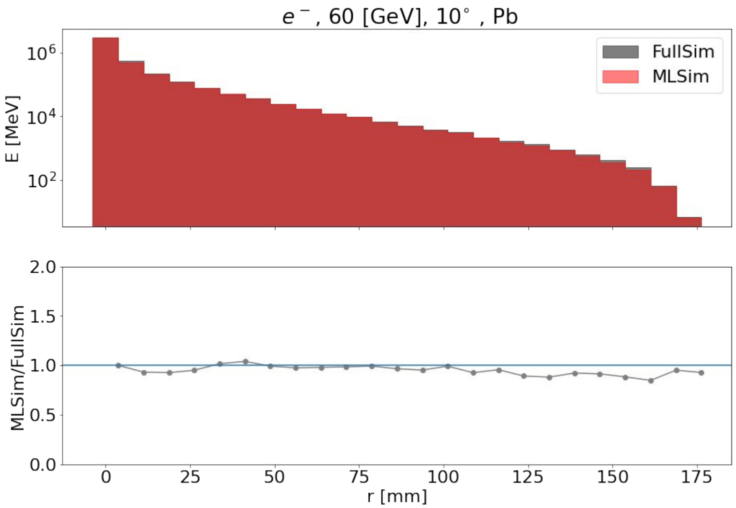
Longitudinal profiles $PBWO_4$

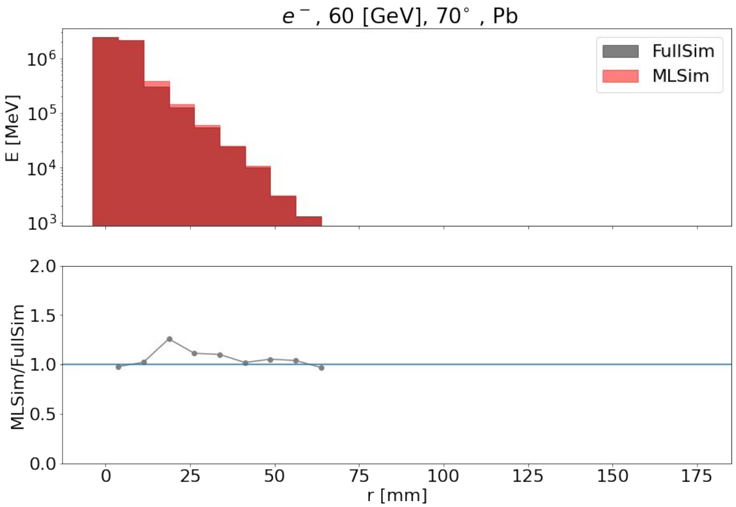


Transverse profiles $PBWO_4$

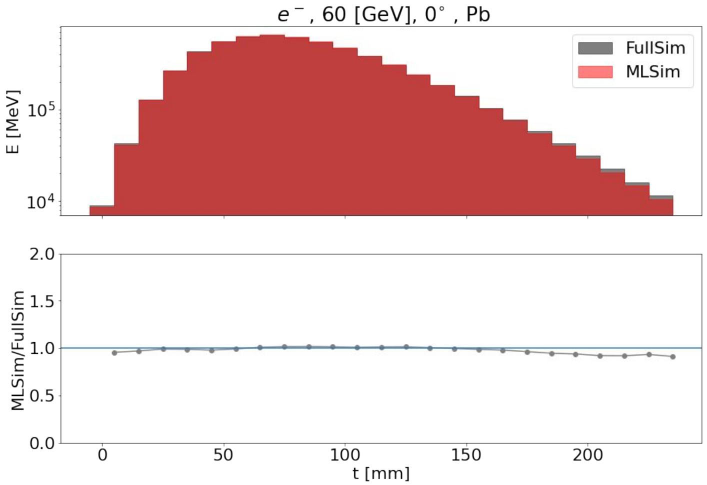




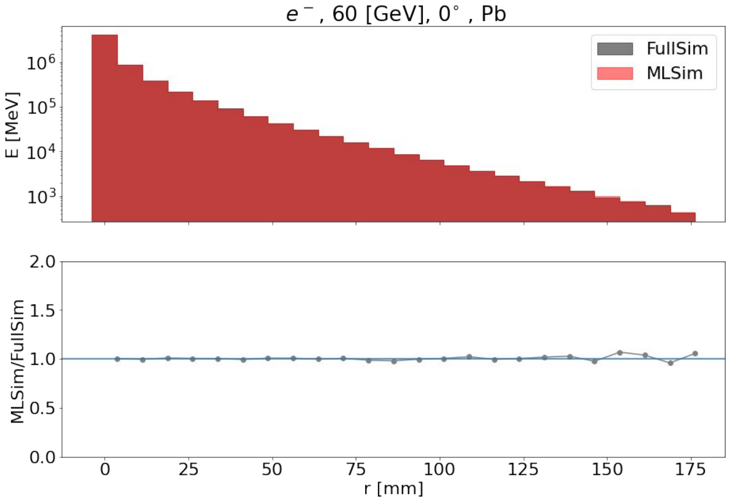




$PBWO_4$

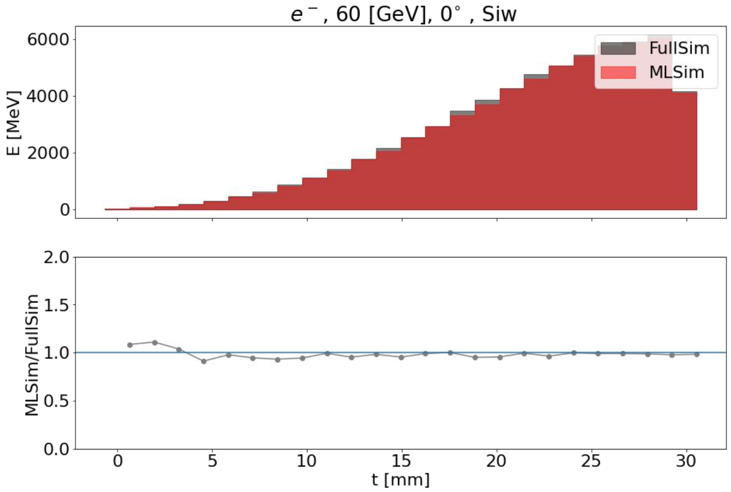


e^-, 60 [GeV], 0°, Pb
FullSim
MLSim
E [MeV]
MLSim/FullSim
t [mm]

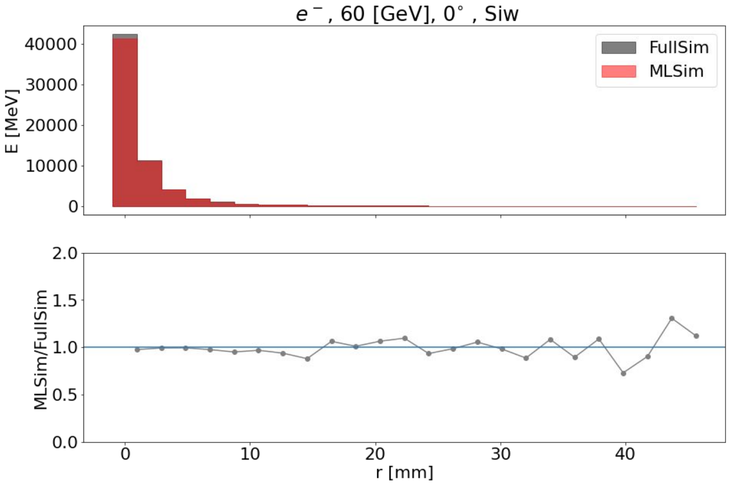


e^-, 60 [GeV], 0°, Pb
FullSim
MLSim
E [MeV]
MLSim/FullSim
r [mm]

SiW



e^-, 60 [GeV], 0°, Siw
FullSim
MLSim
E [MeV]
MLSim/FullSim
t [mm]



e^-, 60 [GeV], 0°, Siw
FullSim
MLSim
E [MeV]
MLSim/FullSim
r [mm]

Geant4 Inference Interface

Interface that allows to read in NN models, configure, and execute inference.

Two main functions :

```
void GetEnergies(std::vector<G4double>& aDepositsEnergies,
                 G4double aParticleEnergy);
```

Infer energies deposited in the detector

```
void GetPositions(std::vector<G4ThreeVector>& aDepositsPositions,
                  G4ThreeVector aParticlePosition,
                  G4ThreeVector aParticleDirection);
```

Calculate positions to corresponding energies in the detector

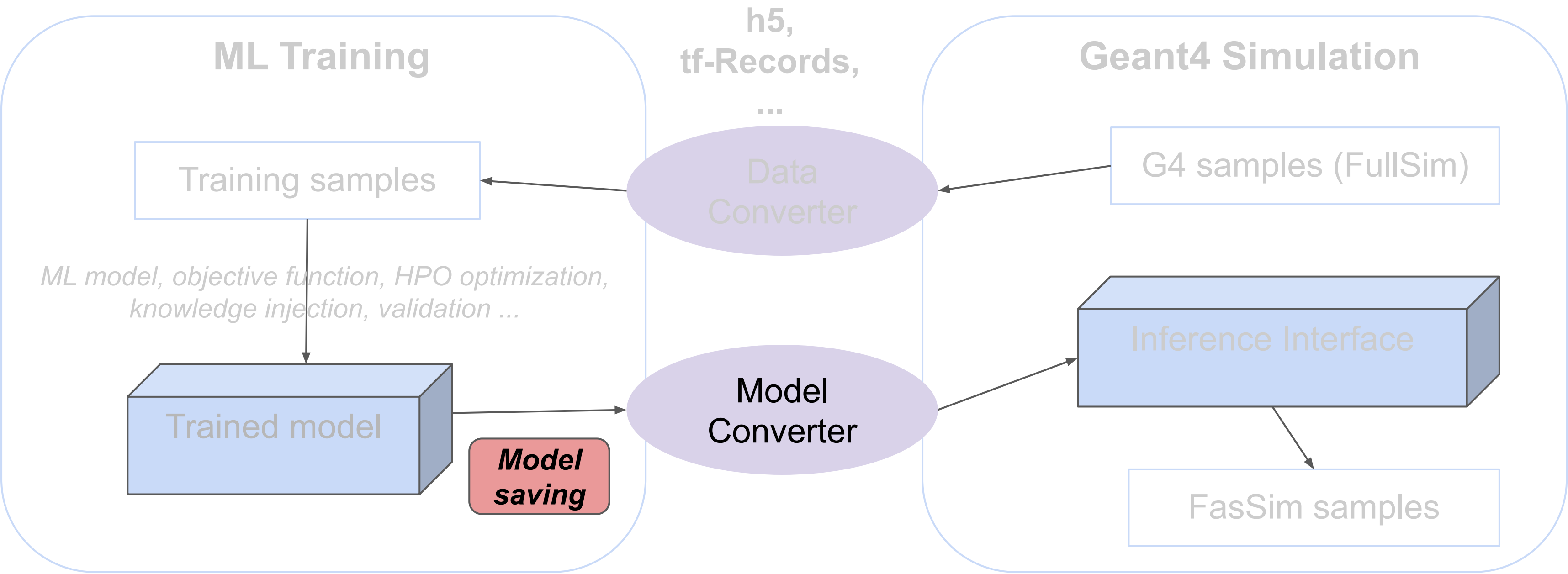
Inference libraries : LWTNN vs ONNX

	LightWeight Trained Neural Network (LWTNN) *Github*
Description	C++ library to apply NN Minimal dependencies : Eigen, Boost
Supported ML libraries	Sklearn and Keras models (it is possible to convert a Tensorflow model to a keras model)
Supported layers	All expect: CNN, Repeat Vector, Reshape.
Supported Activation functions	All except: Selu, PRelu
File format	JSON

Inference libraries : LWTNN vs ONNX

	LightWeight Trained Neural Network (LWTNN) *Github*	**Open Neural Network Exchange (ONXX)** *Github*
Description	C++ library to apply NN Minimal dependencies : Eigen, Boost	Open format to represent ML models ONNX Runtime: a cross-platform framework for ML model's inference and deployment
Supported ML libraries	Sklearn and Keras models (it is possible to convert a Tensorflow model to a keras model)	Saves models from (almost) all libraries
Supported layers	All expect: CNN, Repeat Vector, Reshape.	All
Supported Activation functions	All except: Selu, PRelu	All
File format	JSON	ProtoBuf

From ML training to Geant4 fast simulation

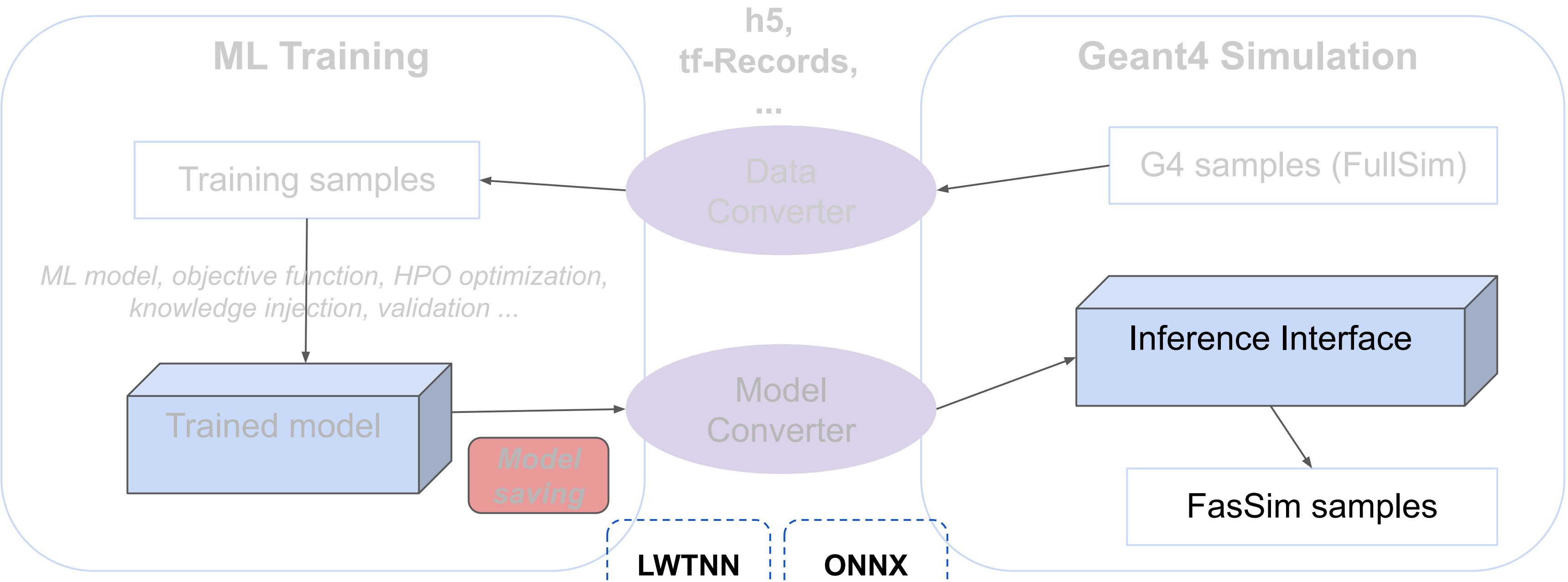


Disk space
Model weights (.h5 file) 28.3 MB
Architecture (.json file) 5.71 kB

LWTNN
.json
195 MB

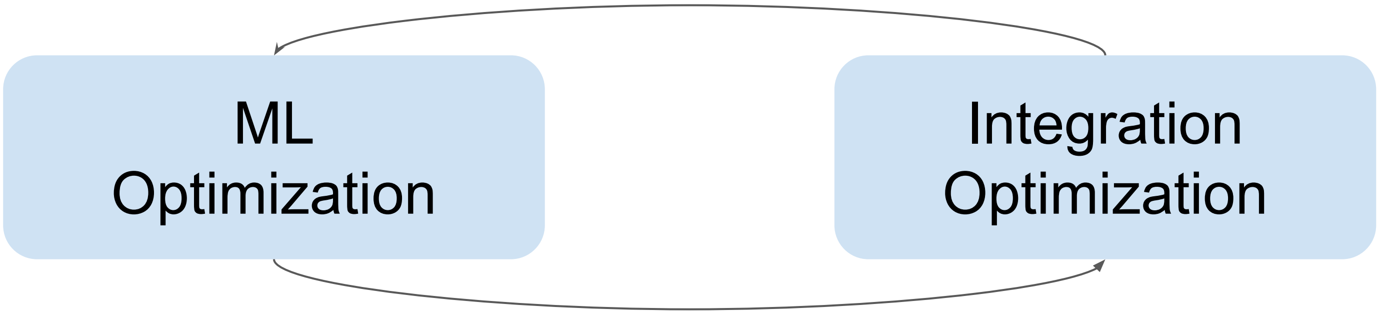
ONNX
.onnx
28.3 MB

From ML training to Geant4 fast simulation



	LWTNN	ONNX
Resident Memory	4GB	61MB
Virtual Memory	4GB	52MB

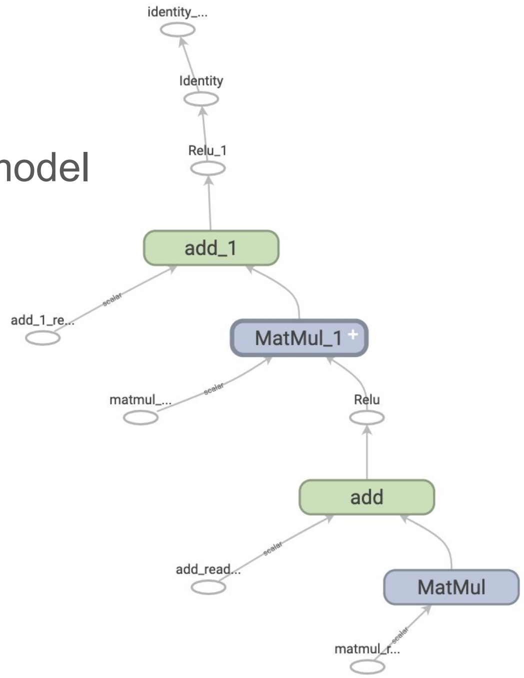
How to better optimize the memory footprint ?



- Using a highly granular calorimeter -> more inputs to the model -> larger network -> more parameters -> larger memory footprint
- The memory footprint can be optimized
  - ML optimization : to reduce the number of trainable parameters
  - Integration optimization : to reduce the complexity of the model representation

Integration optimization : Graph optimization

- Graphs : as data structures
- ONNX Runtime provides various graph optimizations to improve model performance.
- Graph optimizations graph-level transformations
  - **Basic Graph Optimizations** : remove redundant nodes and redundant computation
  - **Extended Graph Optimizations**: fuse nodes
- Graph optimizations can be performed
  - Online mode, the optimizations are done before the inference,
  - Offline mode, the runtime saves the optimized graph to disk.
- ONNX Runtime provides Python, C#, C++, and C APIs to enable different optimization levels and to choose between offline vs. online mode.



Integration optimization : Quantization

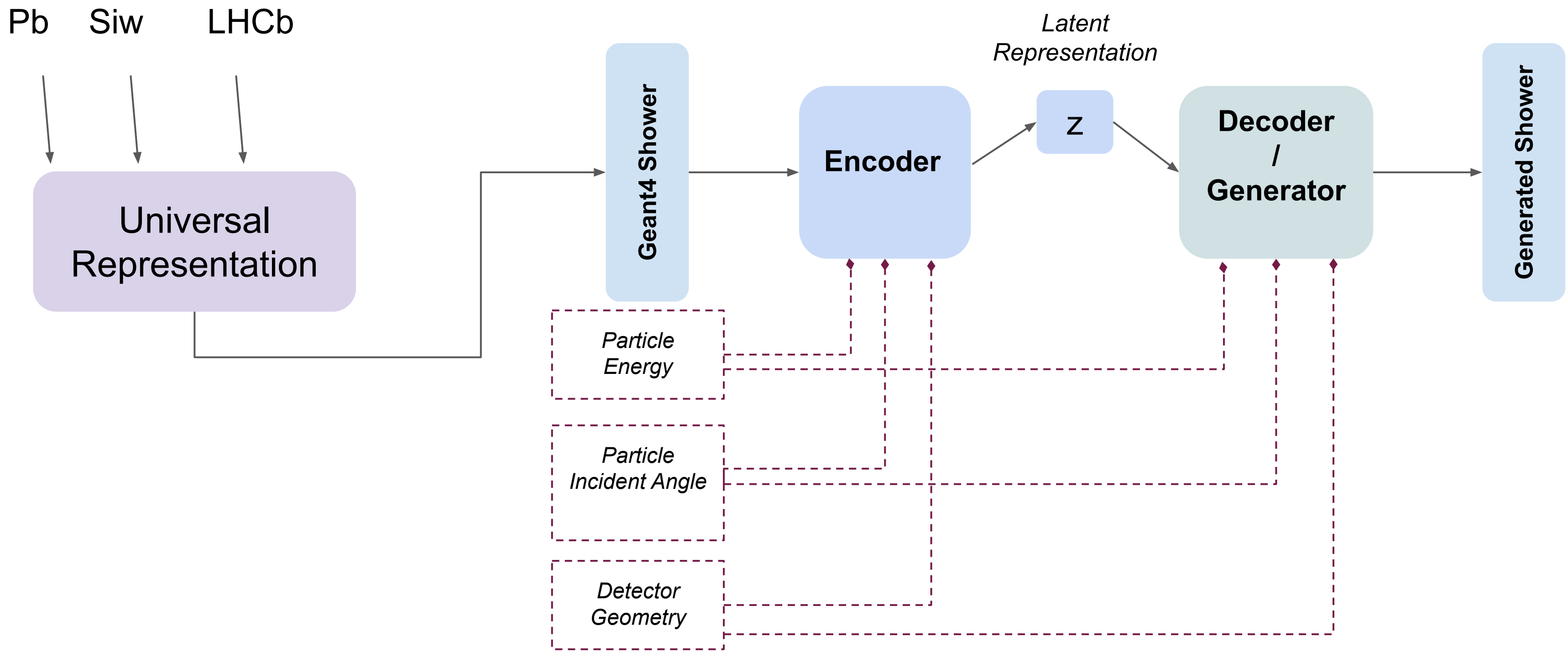
- Quantization in ONNX Runtime refers to 8 bit linear quantization
- Floating point real values are mapped to an 8 bit quantization space

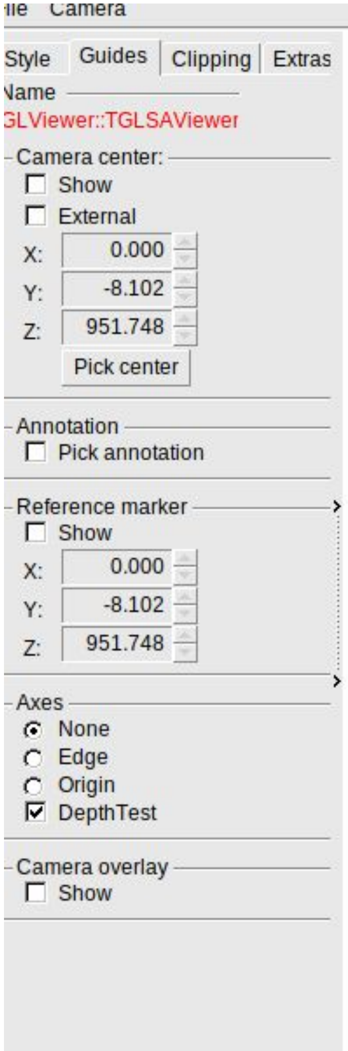
		Raw Model (without optimization)	Quantized Model
	Disk space (MB)	551	139
Loading + Inference	Resident memory (MB)	2265.34	650.414
	Virtual memory(MB)	3205.26	1339.22

Integration optimization : Graph optimization of a quantized model

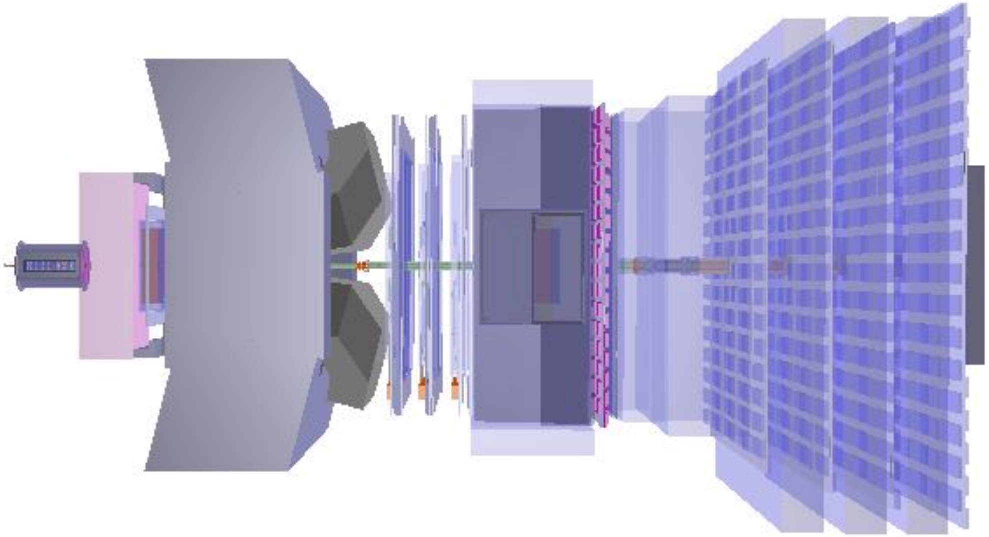
	Basic Optimization	Extended Optimization
Resident memory (MB)	650.414	555.828
Virtual memory (MB)	1339.22	1073.21

Multi-detector geometry modeling



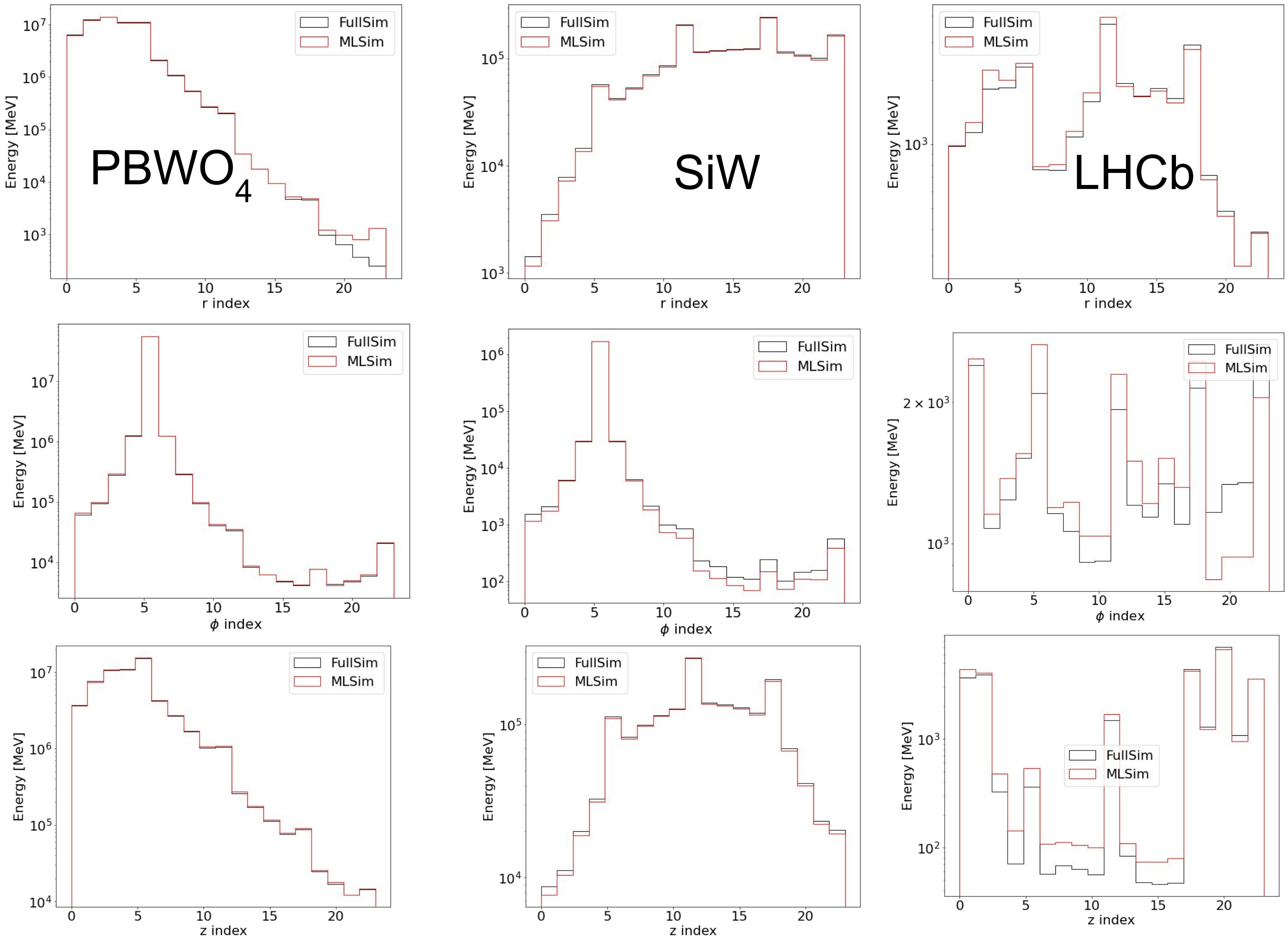


Multi-detector geometry modeling using an LHC experiment calorimeter



LHCb geometry loaded from a [GDML]() file

- Electron particles
- Energy : 60 GeV
- Incident angle : 40º



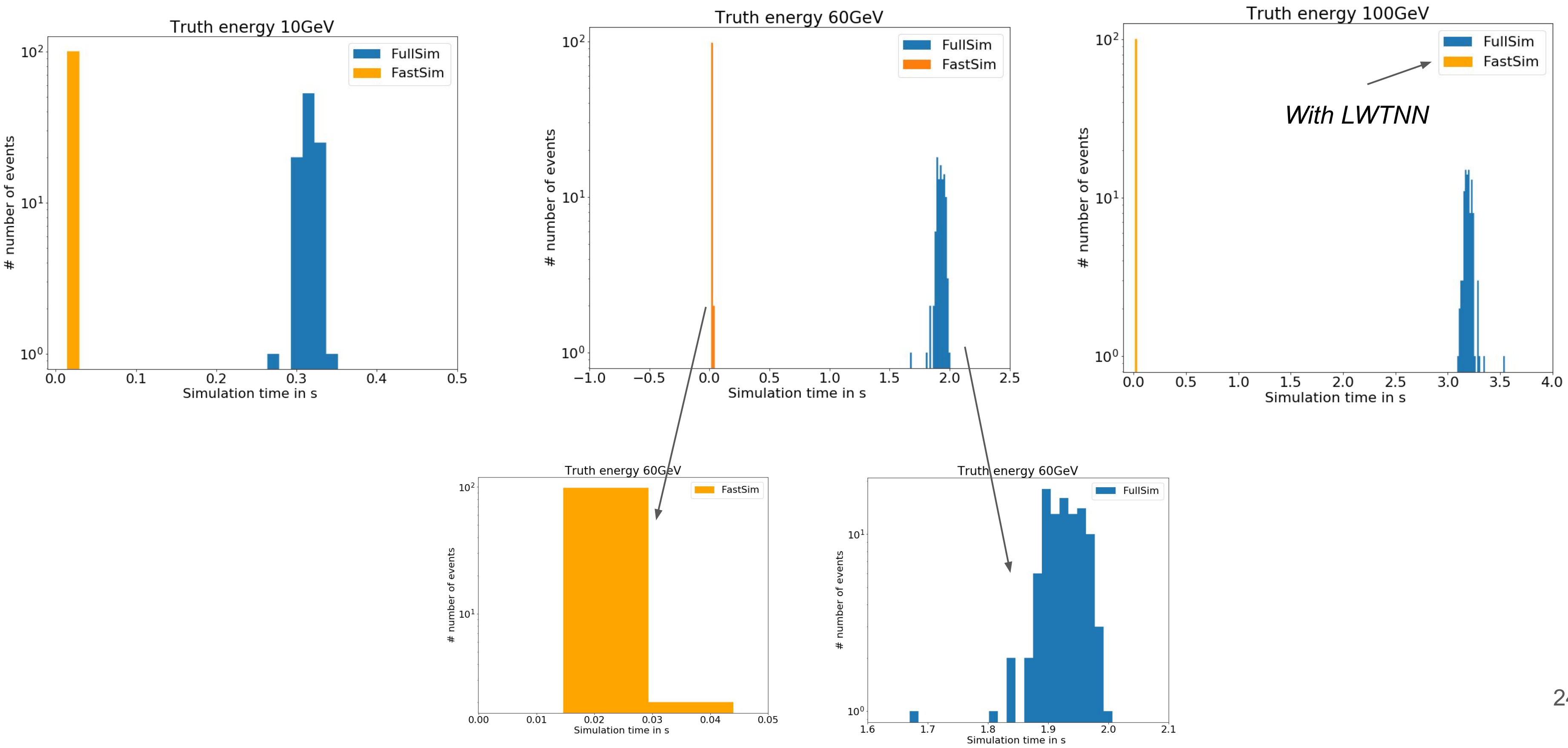
Summary

- Multi-detector geometry model
    - Conditioned on the geometry, energy and incident angle of the particle
    - First tested on 2 simplified geometries
    - Currently testing on a real LHC experiment detector
        - Many improvements are expected
        - More geometries will be tested and evaluated
- Geant4 inference integration
    - Provide G4 examples extending its simulation facilities to ML-based fast simulation
    - Compare inference libraries such as LWTNN, ONNX
    - To better optimize the memory footprint with ONNX
        - Graph optimizations
        - Qunatization

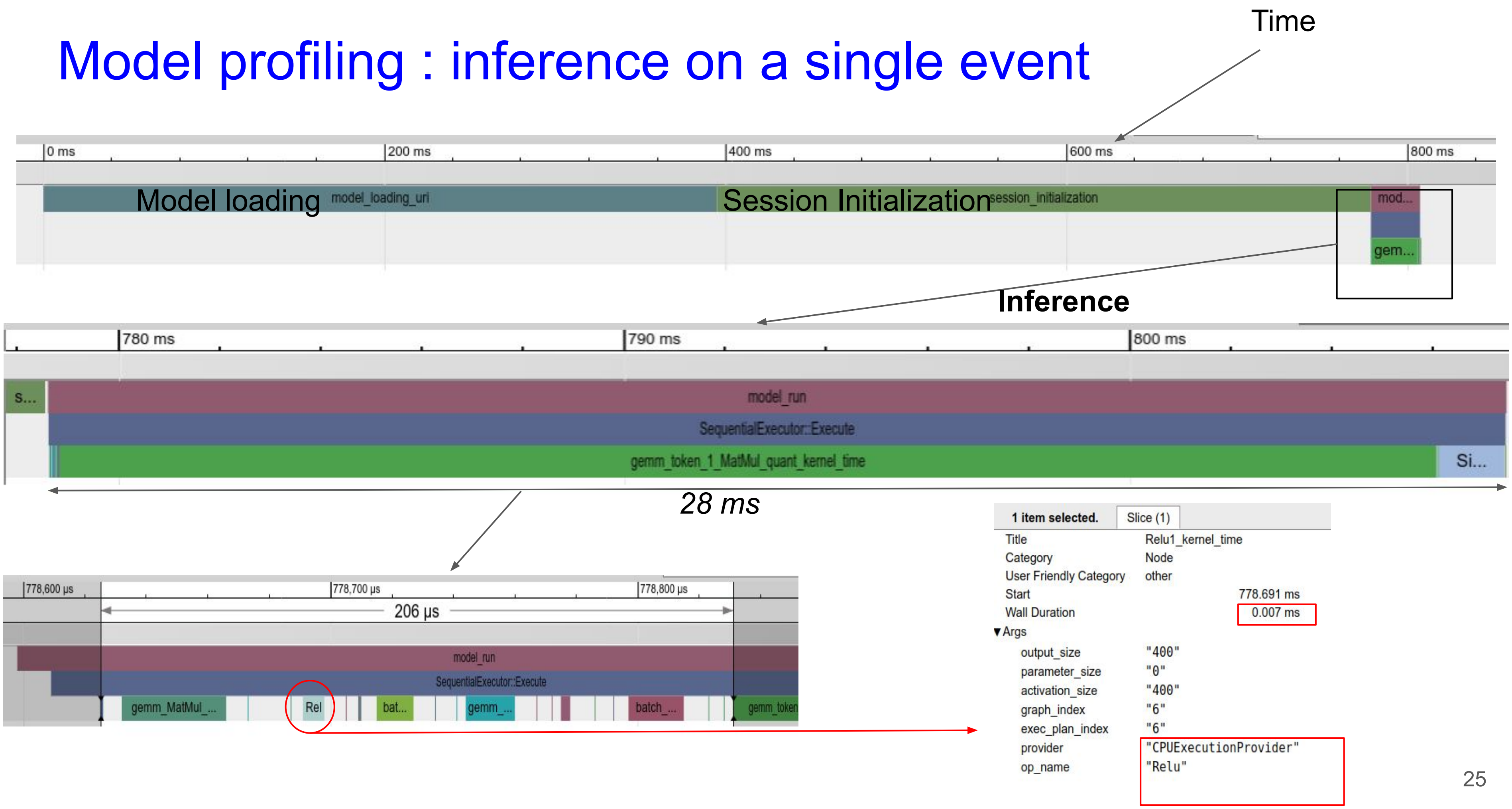
Thank you for your attention !

Backup

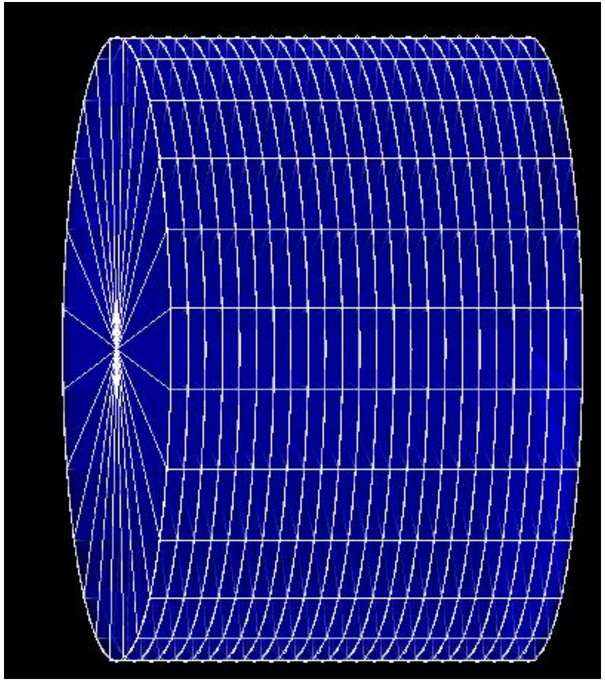
Geant4 Inference Interface : simulation time

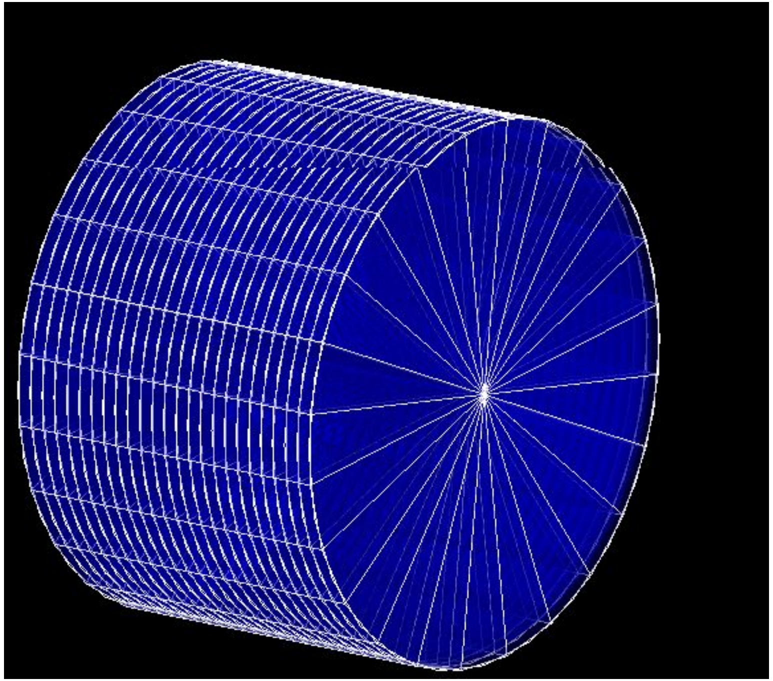


Model profiling : inference on a single event

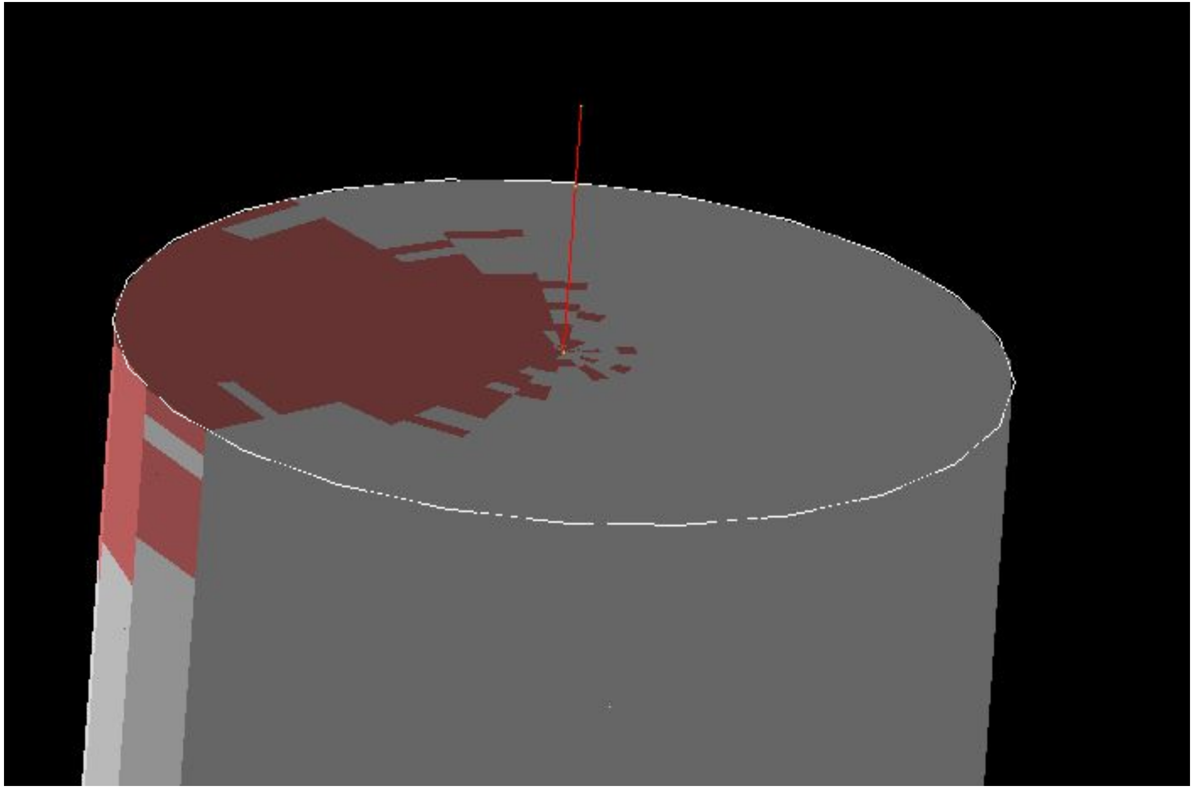


PBWO4 Geometry with 24x24x24 cell segmentation

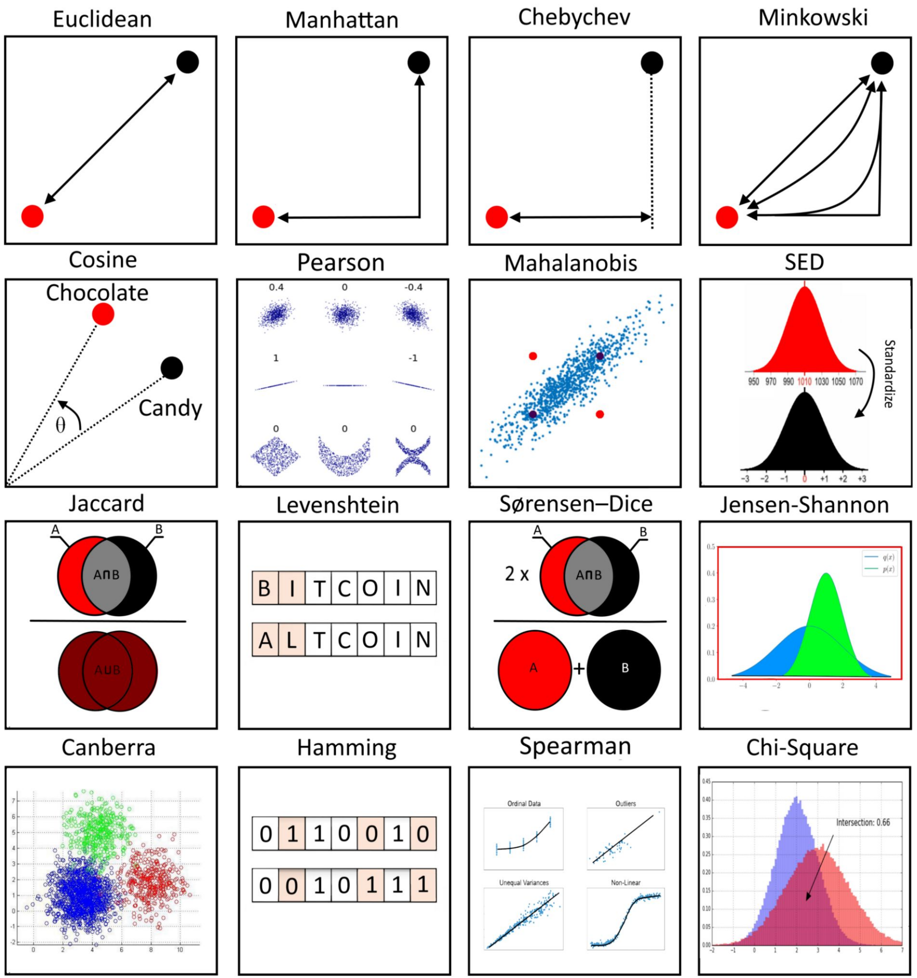




PBWO4 Geometry with 24x24x24 cell segmentation



Validation metrics



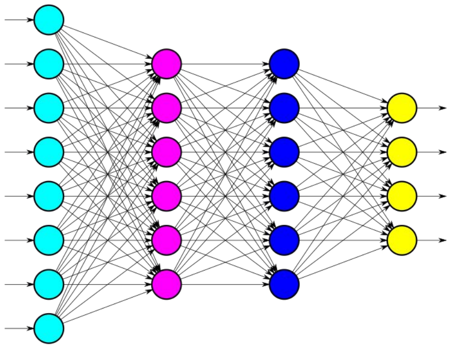
[Source](#)

ML optimization : from dense to convolutional layers

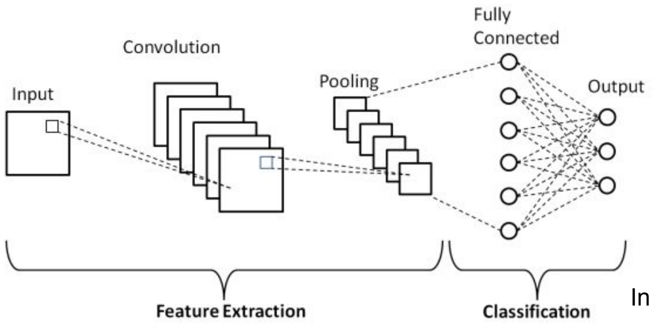
- With smaller input size 24x24x24 dense layers are easy to train , number of trainable parameters depends on the width and length of the NN
- Test 1: (50x48x120) with dense layers
- Test 2: (50,48,120) with Conv layers

\+ Reduce the number of trainable parameters

Dense layers



Convolutional layers



Memory footprint (CPU)

600 Mb (with integration optimization)

500 Mb (no integration optimization)

In CNN only last layer is fully connected