# New developments of TMVA/SOFIE: Code Generation and Fast Inference for Graph Neural Networks

*Ahmat Hamdan, Lorenzo Moneta, Sanjiban Sengupta*

ROOT
Data Analysis Framework
https://root.cern

# Motivation for Fast Inference

- Deployment of models (inference) is often neglected, more focus on training
- Tensorflow/PyTorch have functionality for inference
  - can run only for their own models
  - usage in C++ environment is cumbersome
  - require heavy dependence
- Standard for describing deep learning models:
  - **ONNX** (“*Open Neural Network Exchange*”)
  - cannot describe all possible deep learning models (e.g. GNN) fully
- ONNXRuntime: a efficient inference engine based on ONNX
  - can be difficult to integrate in HEP ecosystem
    - control of threads, used libraries, etc..
    - not optimised for single event evaluation

# Idea for Inference Code Generation

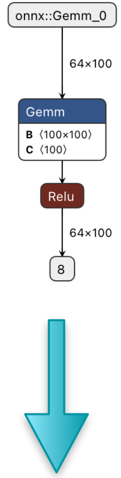
- An inference engine that…
  - **Input**: trained ONNX model file
    - Common standard for ML models
    - Supported by PyTorch natively
    - Converters available for Tensorflow and Keras
  - **Output**: Generated C++ code that hard-codes the inference function
    - Easily invokable directly from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


- **SOFIE : System for Optimised Fast Inference code Emit**

# Code Generation

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

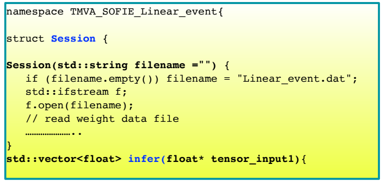
```
using namespace TMVA::Experimental::SOFIE;
RModelParser_ONNX parser;
RModel model = parser.Parse("Model.onnx");
```




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

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

C++ code




- Generated code has minimal dependency
  - only linear algebra library (BLAS) and no ROOT dependency
  - can be easily integrated in your project

# Other SOFIE Parsers

- Parser exists in SOFIE also for :
  - native **PyTorch** files (*model.pt* files)

```
SOFIE::RModel model = SOFIE::PyTorch::Parse("PyTorchModel.pt");
```

  - native **Keras** files (*model.h5* files)

```
SOFIE::RModel model = SOFIE::PyKeras::Parse("KerasModel.h5");
```

- Based on the PyMVA interface (in `libPyMVA.so`)
  - Limited operator support:
    only dense layer and convolutional layers

- See TMVA tutorials TMVA_SOFIE_PyTorch.C and TMVA_SOFIE_Keras.C

# Using the Generated code: in C++

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

```cpp
#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:

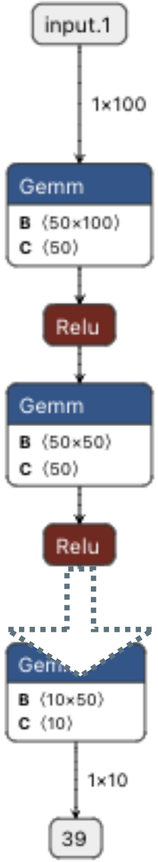
  `FunctorObj::operator()(T x1, T x2, T x3,….);`

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

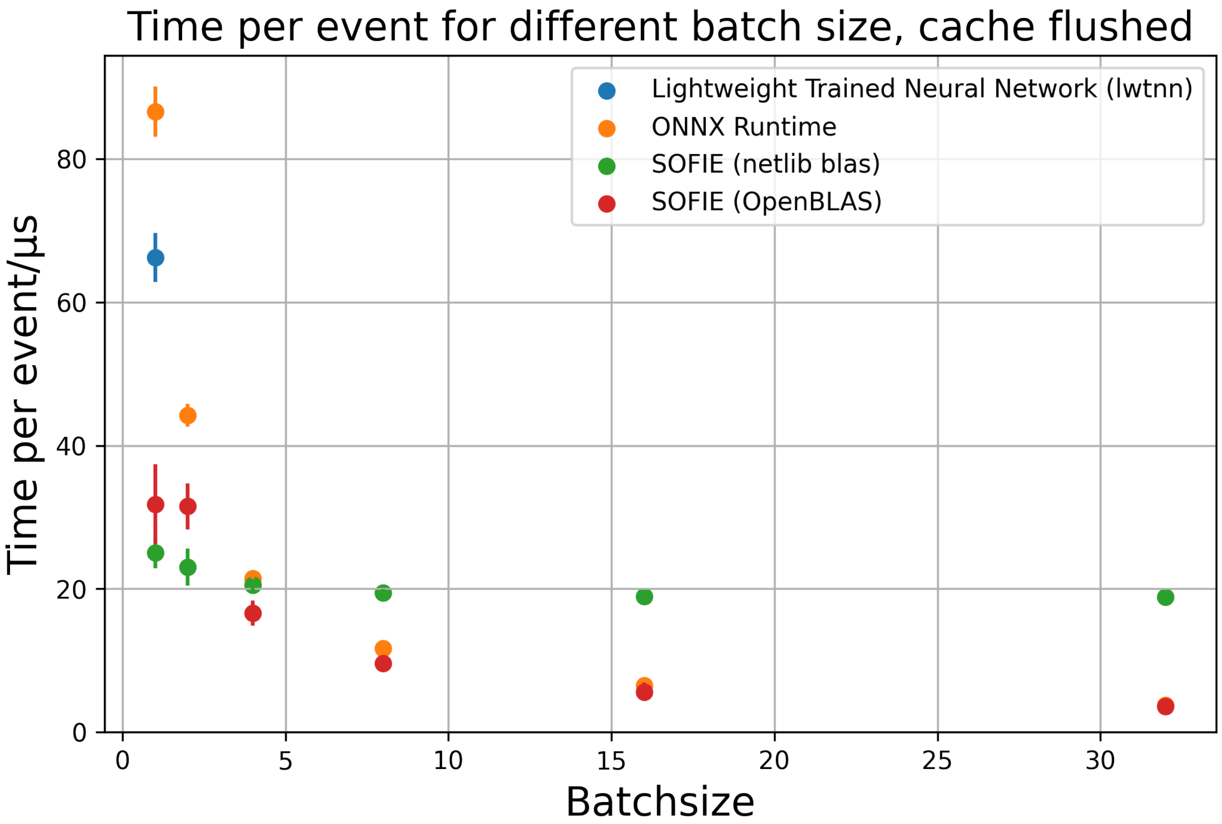
```
ROOT::RDataFrame df("tree", “inputDataFile.root”);
auto h1 = df.DefineSlot("DNN_Value",
SofieFunctor<7,TMVA_SOFIE_higgs_model_dense::Session>(nslots),
{"m_jj", "m_jjj", “m_lv", “m_jlv","m_bb","m_wbb","m_wwbb"}).
Histo1D(“DNN_Value”);
h1->Draw();
```

See full Example tutorial code in C++ or Python

# Benchmark: Dense Model

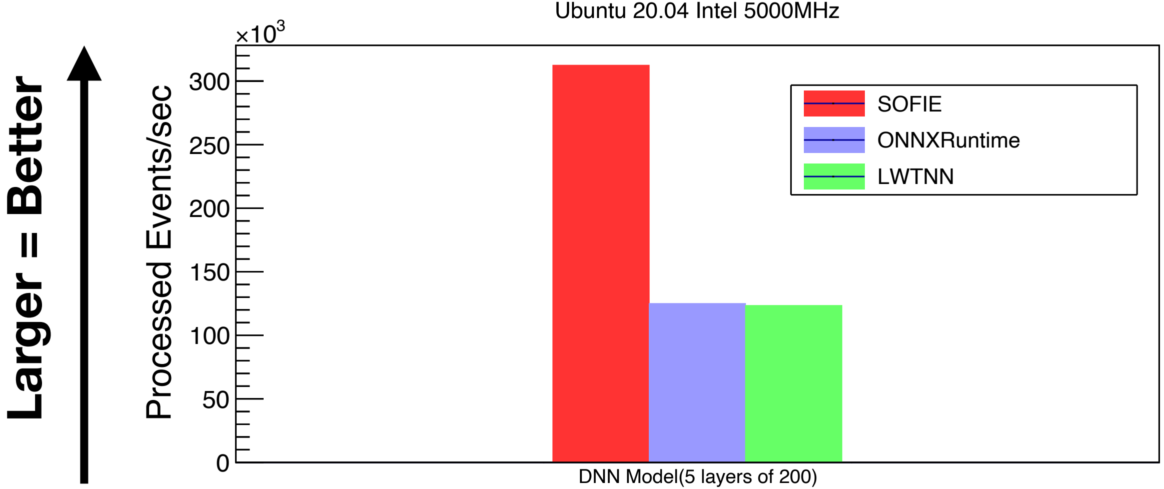



10 Dense layers



# Benchmark with RDF

- Test on a Deep Neural Network (from [TMVA_Higgs_Classification.C]() tutorial)
  5 fully connected layers of 200 units
- Run on dataset of  5M events:
  - Single Thread, but can run also on Multi-Threads



# ONNX Supported Operators

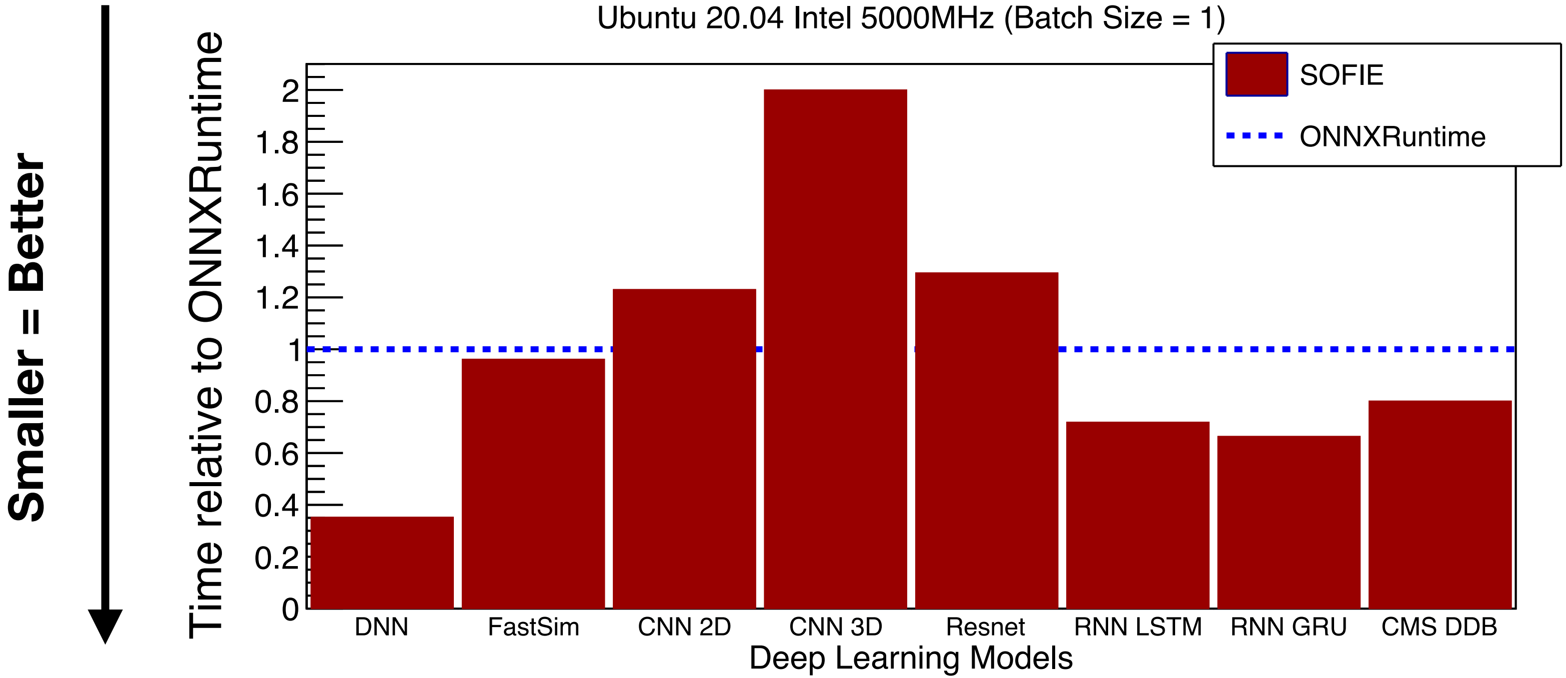
| Implemented and integrated (all in ROOT 6.28) |
|---|
| Perceptron: Gemm |
| Activations: Relu, Selu, Sigmoid, Softmax, Tanh, LeakyRelu |
| Convolution (1D, 2D and 3D) |
| Recurrent: RNN, GRU, LSTM |
| Pooling: MaxPool, AveragePool, GlobalAverage |
| Deconvolution (1D,2D,3D) |
| Layer Unary operators: 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 |

- **Implemented but to be integrated (PR #11208):**
  - GNN (Message Passing GNN based on DeepMind GraphNet
- Next to support:
  - e.g. GNN from PyTorch geometric?
  - Depending on user needs

# Benchmark Different Model Architectures

- Test event performance of SOFIE vs ONNXRuntime

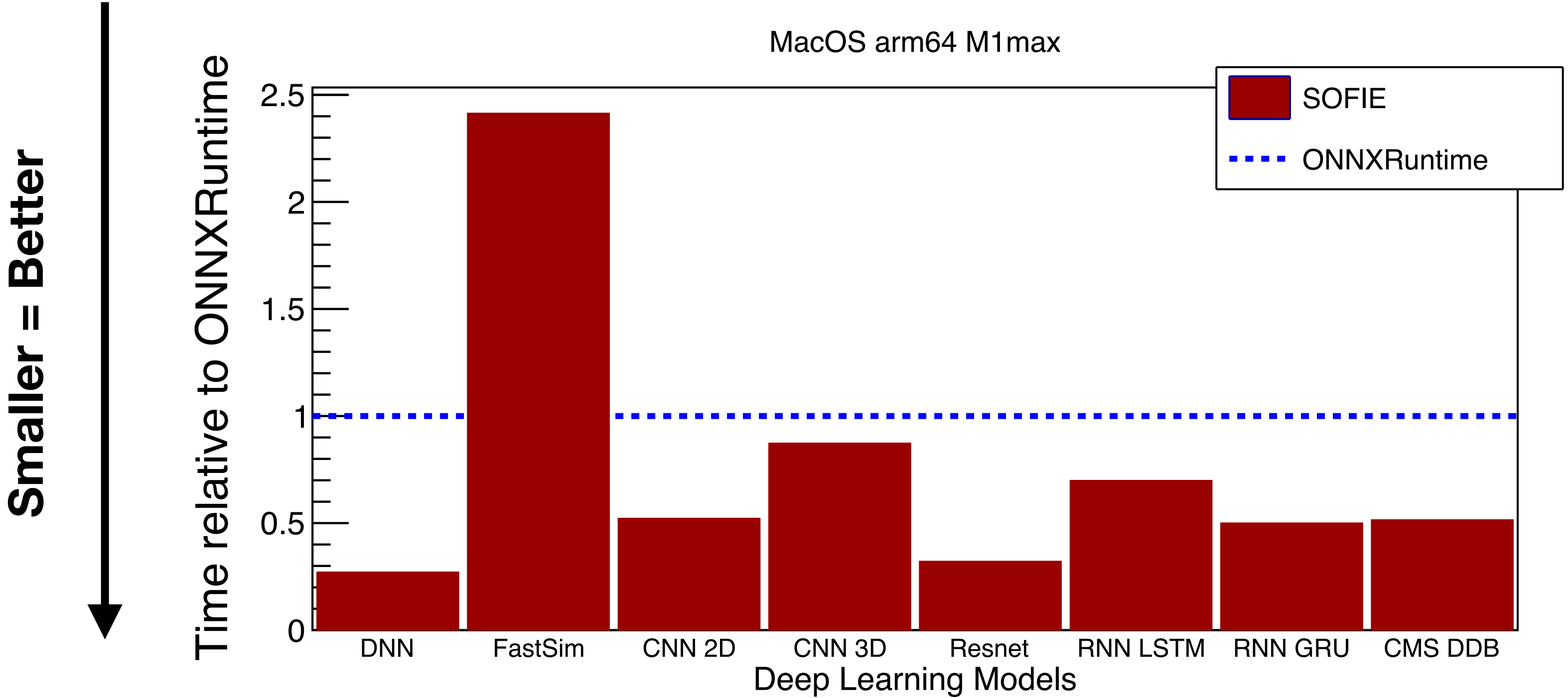
  (using batch size = 1)



# Benchmark Different Model Architectures

- Test event performance of SOFIE vs ONNXRuntime

  (using batch size = 1 and MacOS M1)

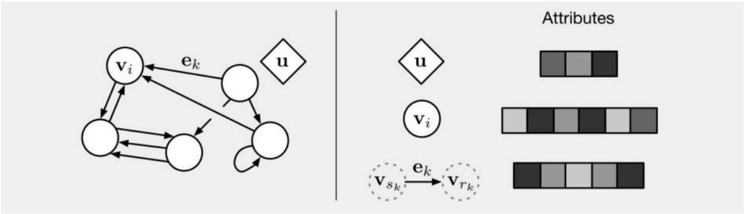


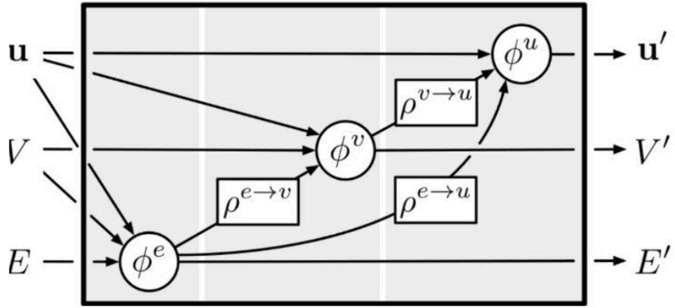
# SOFIE for Graph Networks

- First developments to support GNN models
- Started 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 CHEP-2023 contribution #459* )
    - important to have efficient implementation and with minimal dependencies
  - The initial prototype for SOFIE has been developed
    - available as ROOT PR #11208

# GNN Support

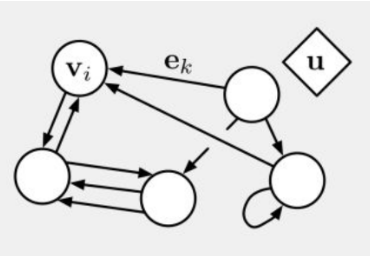
- Follow **Graph Nets** architecture
  - A model is described by
    - number of nodes and edges
    - sender/receiver list of edges
    - number of features (for node, edge and global)
  - Updating functions on node, edge and global features
    - MLP (Multi-Layer Perceptron)
      - including activation functions and layer normalisation
    - Aggregation functions
      - Mean, Sum,…



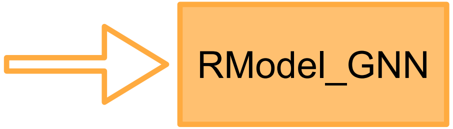



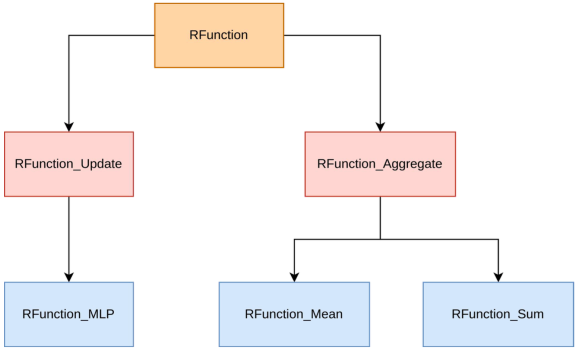
# SOFIE GNN Support

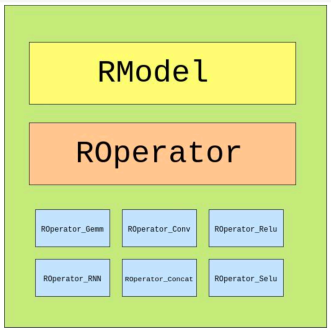
- Developed **C++ classes** for representing **GNN structure**.
  - based on SOFIE **`RModel`** and the **`ROperator`** classes developed for supporting ONNX.
  - SOFIE classes provide the functionality to generate C++ inference code
- **Python code** (based on PyROOT) for initialising SOFIE classes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




Graph Nets GNN

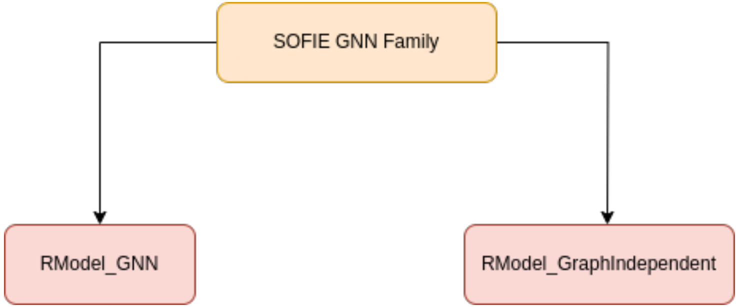


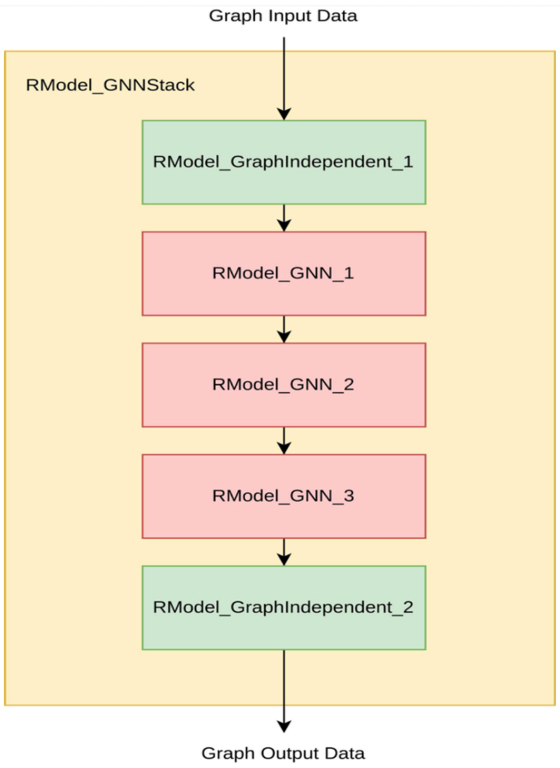



# GNN Inference

- Final model is composed by several blocks chained together
  - SOFIE can generate C++ code for each single GNN block
  - a C++ `struct` of `RTensor`'s represents the GNN data flowing trough the model
  - Users can stuck the GNN blocks according to the desired architecture in the inference function for the full model

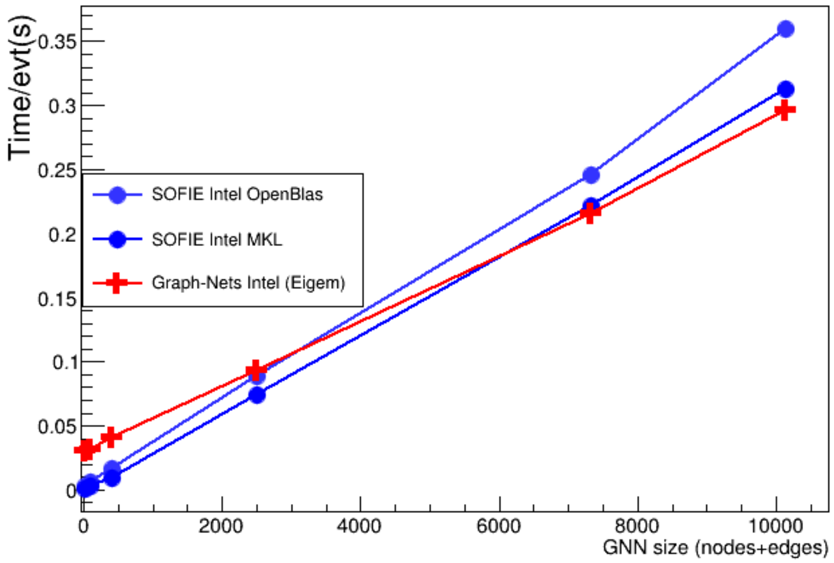


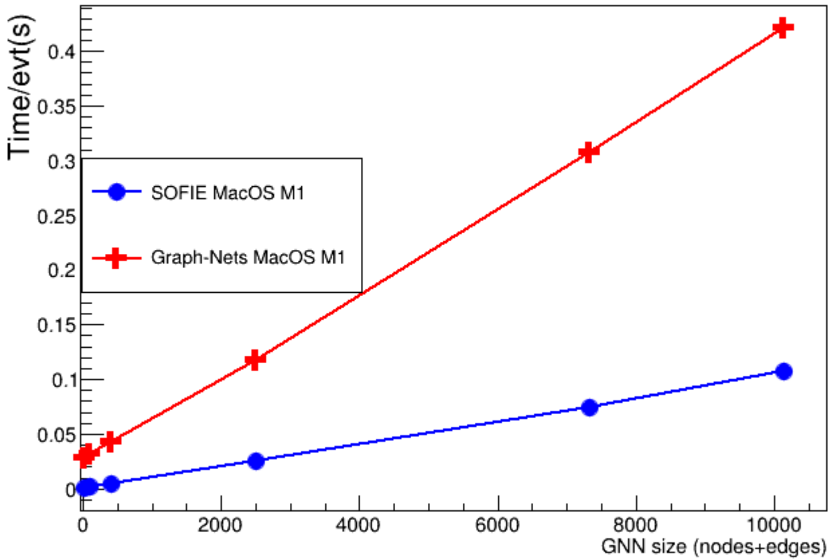
# Benchmark of SOFIE GNN

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

Intel Linux Desktop



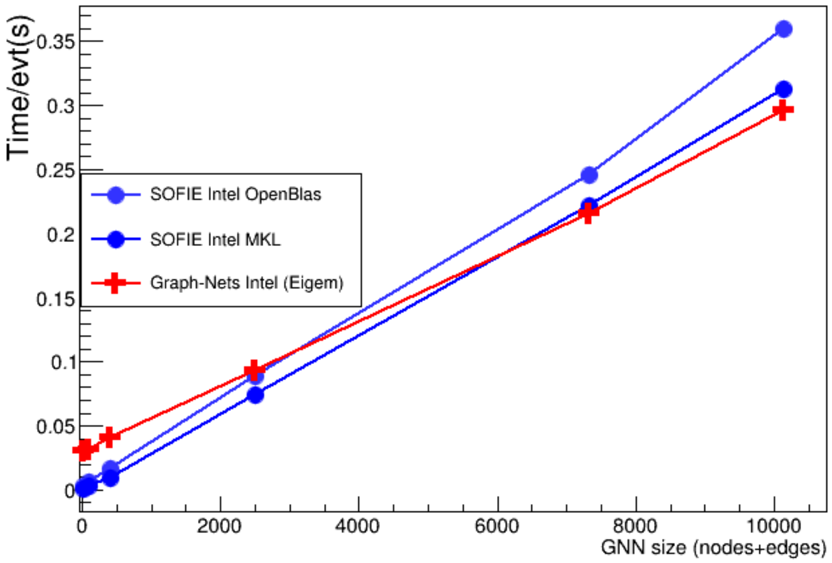

MacOS M1



# SOFIE GNN Performance

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

Intel Linux Desktop




- Benchmark results for SOFIE
  - 5-10 faster for small GNN size
  - comparable for large GNN (but much faster on MacOS)
- For large model, evaluation will be dominated by matrix operations (BLAS)
- Memory usage is similar, but no optimisation for memory has been done so far in SOFIE.

# Future Work for SOFIE

- Implement missing ONNX operators depending on user requests
- Extend support for Keras/Tensorflow direct parser
- Extend GNN support for different types of GNN
  - support some GNN types from the PyTorch geometric library
  - e.g. point-cloud GNN used by ParticleNet (CMS)
- Implement some optimisations:
  - optimisation of memory usage
  - layer fusions
- Investigate to generate code for different architectures (e.g GPU)
- Collaborate with hls4ml project to have inter-operability between the tools
- Support for other type of architectures can be done depending on user needs

# Summary

- **SOFIE**, fast and easy-to-use inference engine for Deep Learning models, is available in ROOT (version 6.28)
  - Integrated with other ROOT tools (*RDataFrame* ) for ML inference in end-user analysis
- **Good performance** compared to existing packages (e.g. ONNXRuntime)
- **SOFIE can now support Graph Networks**
- Future developments are done according to user needs and the received feedback!

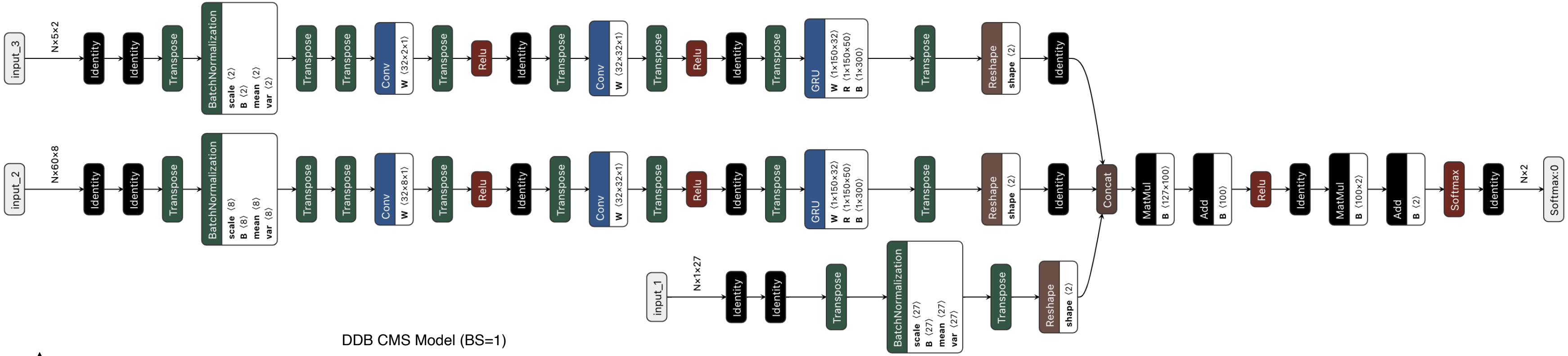
# Example Notebooks and Tutorials

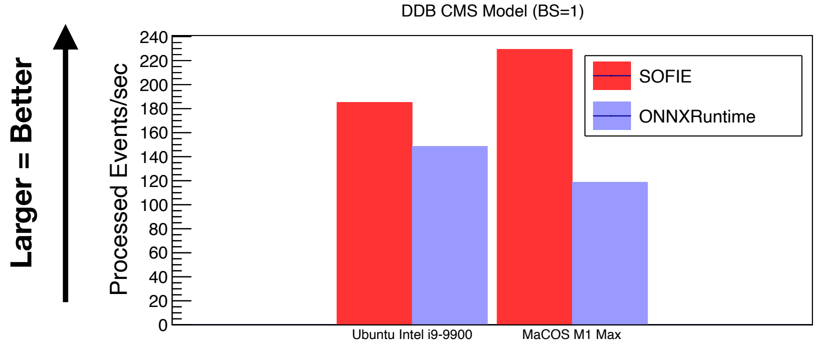
- Example notebooks on using SOFIE:
  - https://github.com/lmoneta/tmva-tutorial/tree/master/sofie
- Tutorials are also available in the tutorial/tmva directory
- Link to SOFIE code in current ROOT master in GitHub
- Link to benchmarks in *rootbench*

# Backup Slides

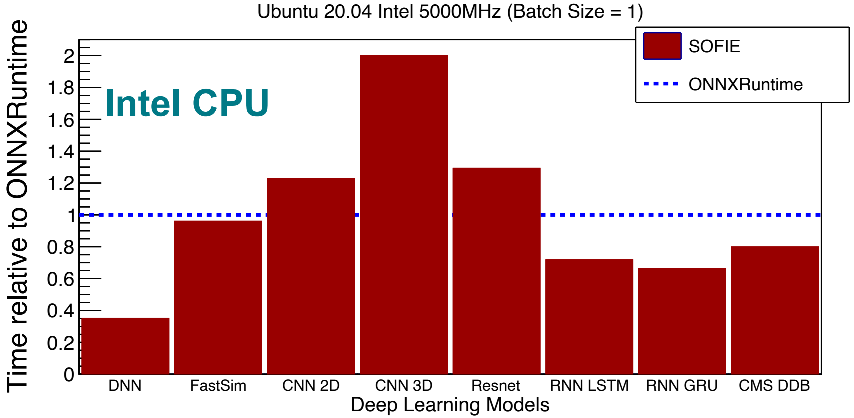
# Benchmark using a CMS Model

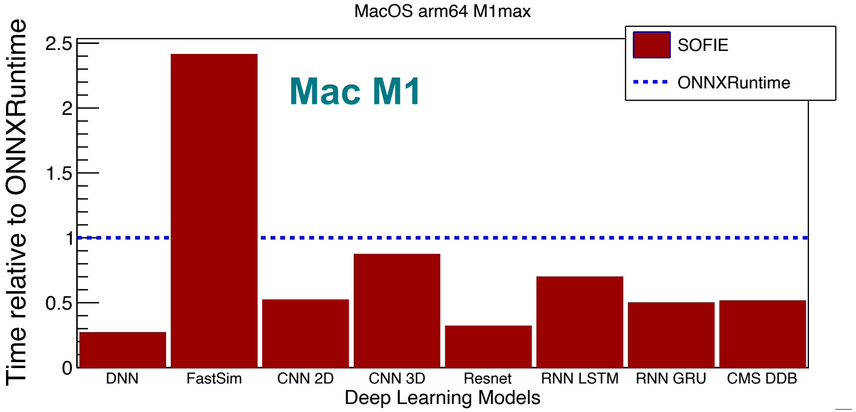
- SOFIE can parse some complex models: CMS Deep Double model (DDB.onnx)
  - 3 inputs with 1d Conv + GRU



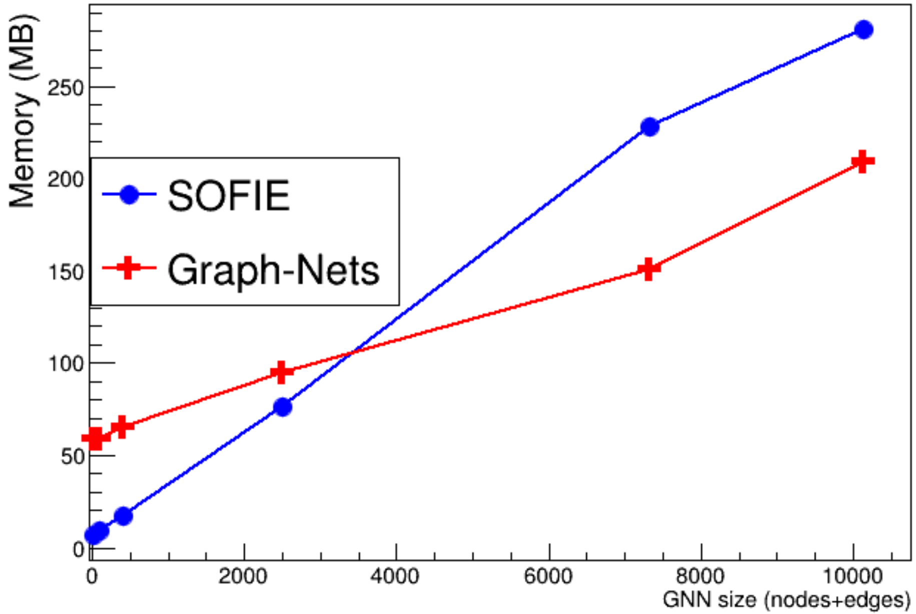
- Comparison of SOFIE inference with ONNXRuntime (from Microsoft) and LWTNN (ATLAS)
  - 2-3 faster than ONNXRuntime for DNN with batch size=1
    - e.g. using RDF interface for a DNN with 5 layers of 200x200 nodes:
      - SOFIE: 310K evts/s, ONNXRuntime: 120K evt/s, LWTNN: 120K evts/s
  - 20% faster for RNN operators
  - slightly slower for CNN ( 20% for 2D ) on Linux but not on MacOS M1 (difference probably due to different BLAS implementation used)
  - Further optimisations are still possible



# GNN Memory Usage

- Measure memory usage in both SOFIE and Graph-Nets




- no optimization done for SOFIE
- possibility to reduce memory usage by a significant factor