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

Ahmat Hamdan, Lorenzo Moneta, Sanjiban Sengupta

CERN

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
  - requires large dependency
  - 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

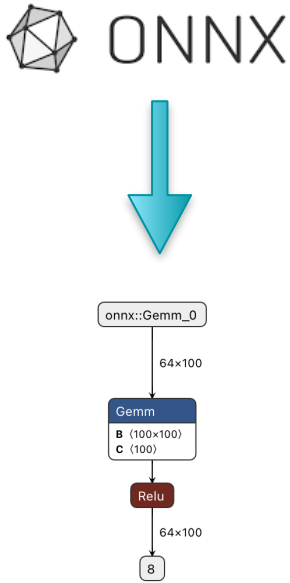
- An inference engine that…
  - **Input**: trained ONNX model file
    - Common standard for ML models
    - Supported by PyTorch natively
    - Converters available for Tensorflow and Keras
  - **Output**: Generated C++ code that hard-codes the inference function
    - Easily invokable directly from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

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

Parsing input models

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

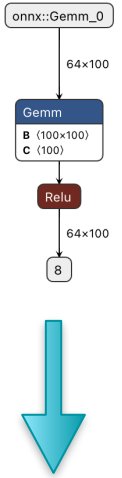
```
using namespace TMVA::Experimental::SOFIE;
RModelParser_ONNX parser;
RModel model = parser.Parse("Model.onnx");
```



Code Generation

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

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



C++ code

```
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){
```

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

Using the Generated code: in C++

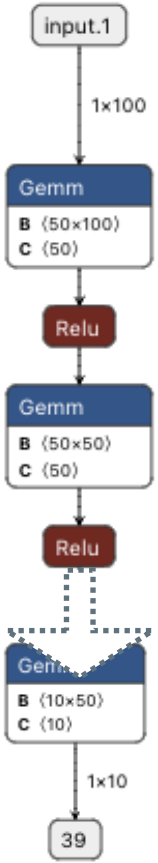
- SOFIE generated code can be easily used in compiled C++ code

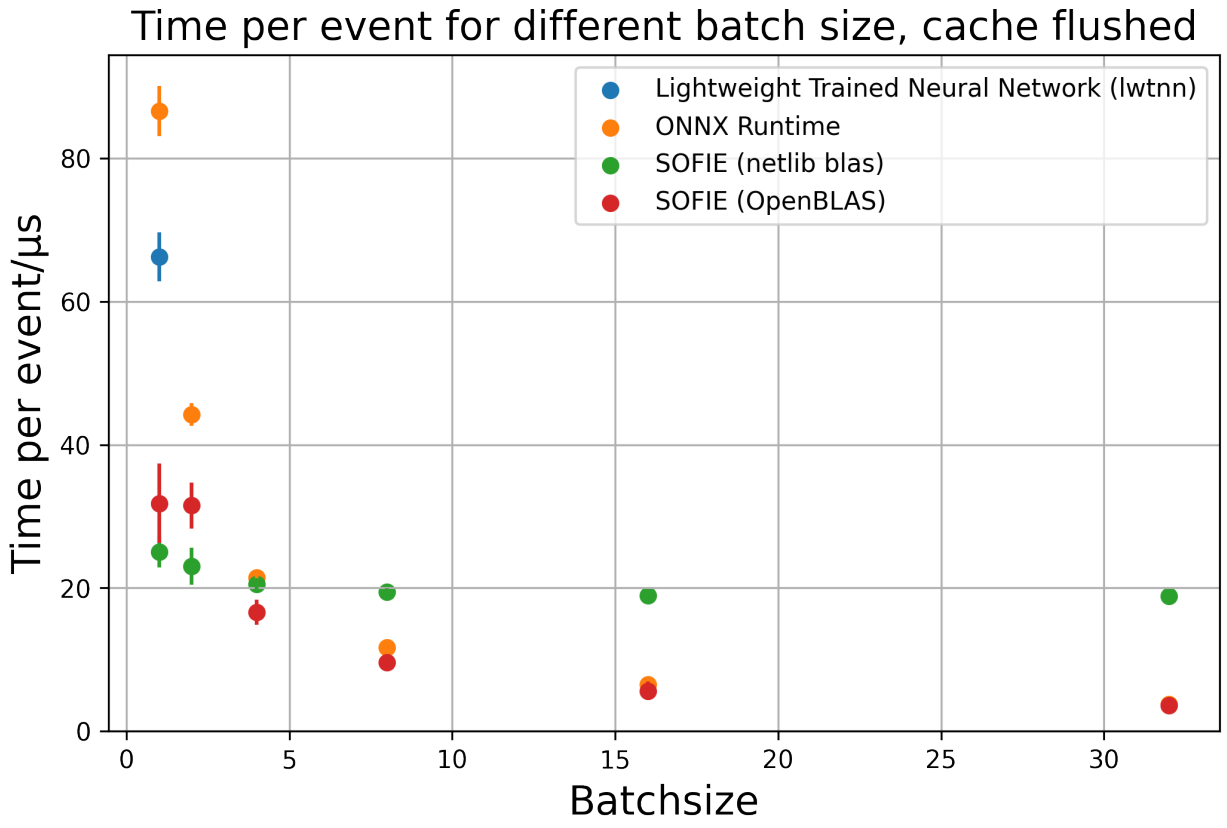
```
#include "Model.hxx"
// create session class
TMVA_SOFIE_Model::Session s();
//-- event loop
…….
{
   // evaluate model: input is an array of type float *
   float * input = …..
   std::vector<float> result = s.infer(input);
}
```

See full Example tutorial code

Benchmark: Dense Model

10 Dense layers





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()
#-- event loop
.....
  # evaluate the model , input can be a numpy array of type float32
  result = s.infer(input)
```

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,….);
```

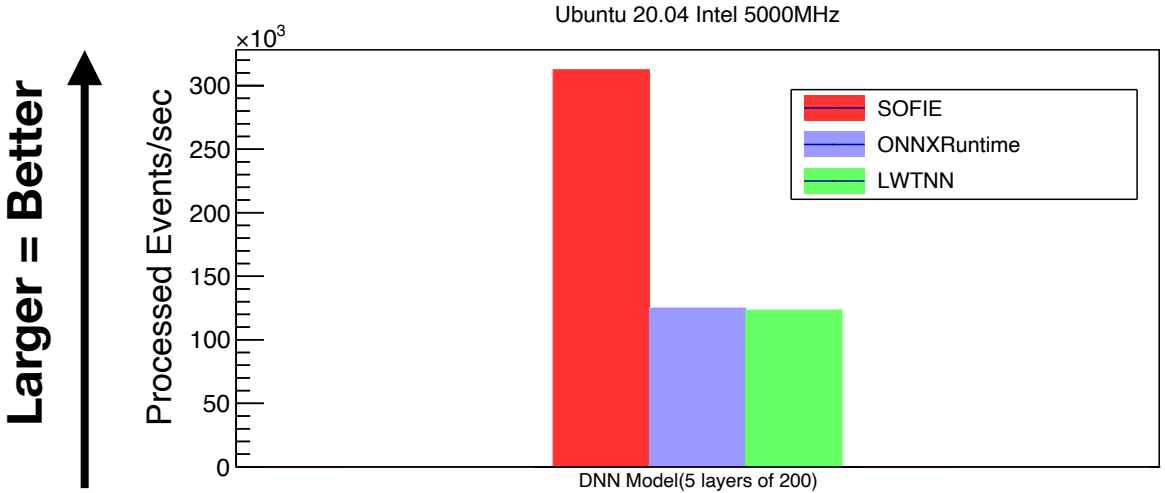
- We have developed a generic functor adapting SOFIE signature to the RDF one
  - Support for multi-thread evaluation, using RDF slots

```
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");
```

See full Example tutorial code in C++ or Python

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 Multi-Threads



Other SOFIE Parsers

- Parser exists 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

ONNX Supported Operators

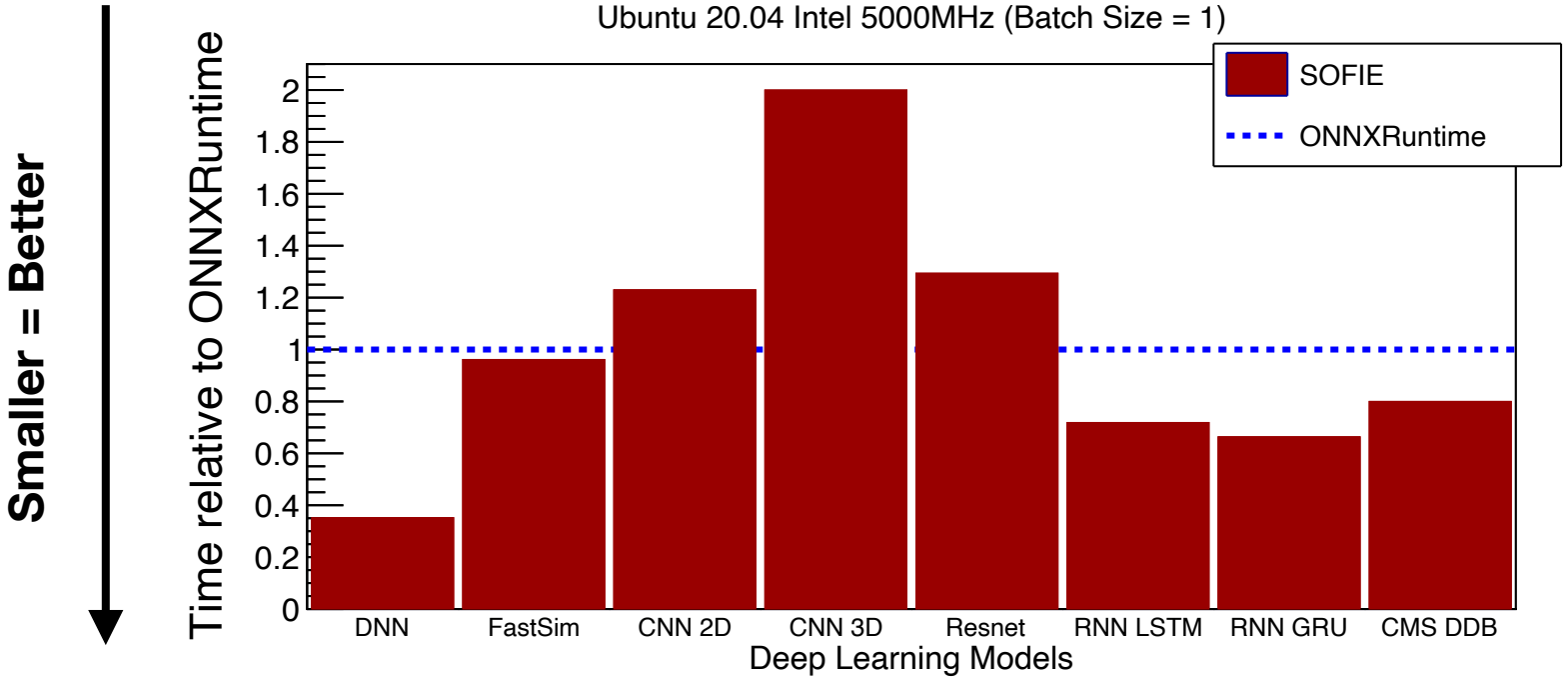
Implemented and integrated (all in ROOT 6.28)
Perceptron: Gemm
Activations: Relu, Selu, Sigmoid, Softmax, Tanh, LeakyRelu
Convolution (1D, 2D and 3D)
Recurrent: RNN, GRU, LSTM
Pooling: MaxPool, AveragePool, GlobalAverage
Deconvolution (1D,2D,3D)
Layer Unary operators: Neg, Exp, Sqrt, Reciprocal, Identity
Layer Binary operators: Add, Sum, Mul, Div
Reshape, Flatten, Transpose, Squeeze, Unsqueeze, Slice, Concat, Reduce
Custom operator
BatchNormalization, LayerNormalization
Gather (for embedding)

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

Benchmark Different Model Architectures

- Test event performance of SOFIE vs ONNXRuntime

(using batch size = 1)

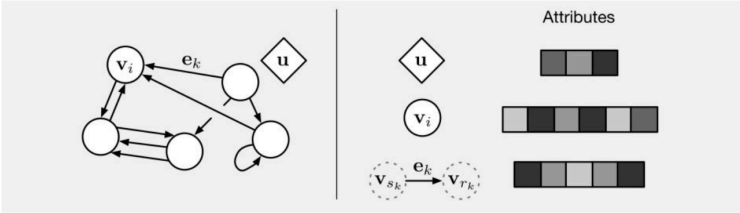


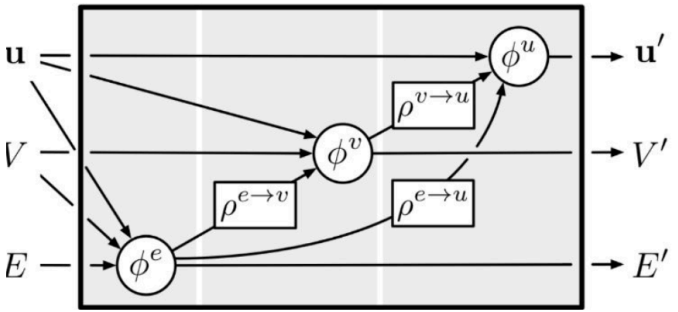
Graph Nets Inference

- Initiated developments to support GNN inference
- Focus on a type of network developed within LHCb:
  - Message Passing GNN from the DeepMind GraphNet library
    - model to be used in LHCb for End-End event reconstruction and filtering
    - important to have efficient implementation with minimal dependencies
  - Initial prototype available as a PR in ROOT
    - see https://github.com/root-project/root/pull/11208

GNN Support

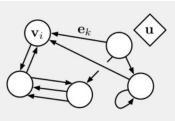
- Following GraphNet architecture
  - Model described by
    - number of nodes and edges
    - sender/receiver list of edges
    - number of features (for node,edge and global)
  - Updating functions on node, edge and global features
    - MLP (Multi-Layer Perceptron)
      - including activation functions and layer normalisation
    - Aggregation functions
      - Mean, Sum,…

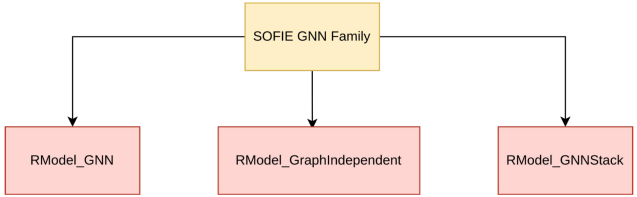


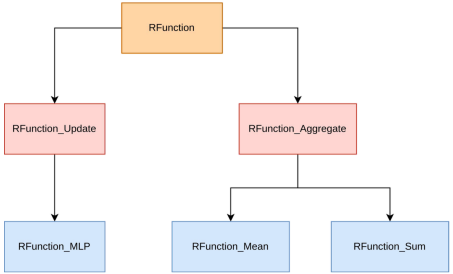


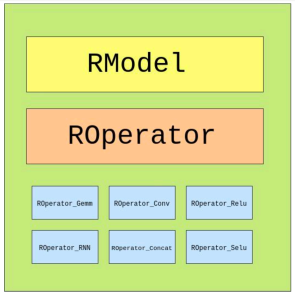
SOFIE GNN Support

- C++ classes for describing GNN structure and its intermediate representation.
  - Based on SOFIE `RModel` and `ROperator` classes developed for supporting ONNX.
- Python code (based on PyROOT) for initialising SOFIE C++ objects from external GNN models (e.g. GraphNet)
- SOFIE classes provide the functionality to generate C++ inference code



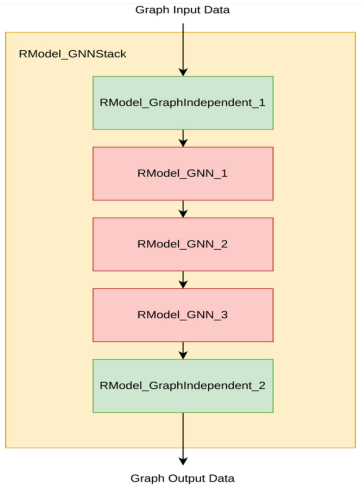






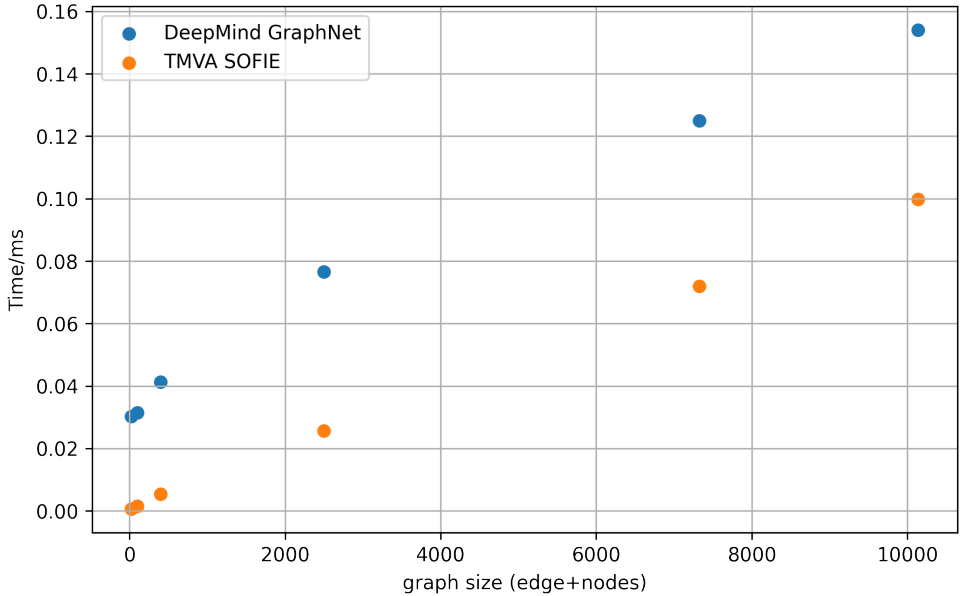
GNN Inference

- Full model can be composed by several blocks chained together
  - SOFIE can generate C++ code for each single block
  - Users can integrate them according to the desired model architecture in the final inference function for the model



GNN Performance

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



- C++ SOFIE implementation faster than GraphNet !
  - > 10 faster for smaller models
  - 50 % for large ones
- Expect for large models similar performance since evaluation will be dominated by matrix operations (BLAS)

Future Work

- Implement missing ONNX operators depending on user requests
- Extend support for Keras/Tensorflow direct parser
- Implement same model optimisations:
  - layer fusions, quantisations,….
- Extend GNN support for different types of GNN
  - e.g. point-cloud GNN used by ParticleNet (CMS)
  - support for GNN from the PyTorch geometric library
- Generate code for different architectures (e.g GPU)
  - In contact with hls4ml project for collaborating and have inter-operability between the tools
- Support for other model architectures can be done depending on user needs

Summary

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

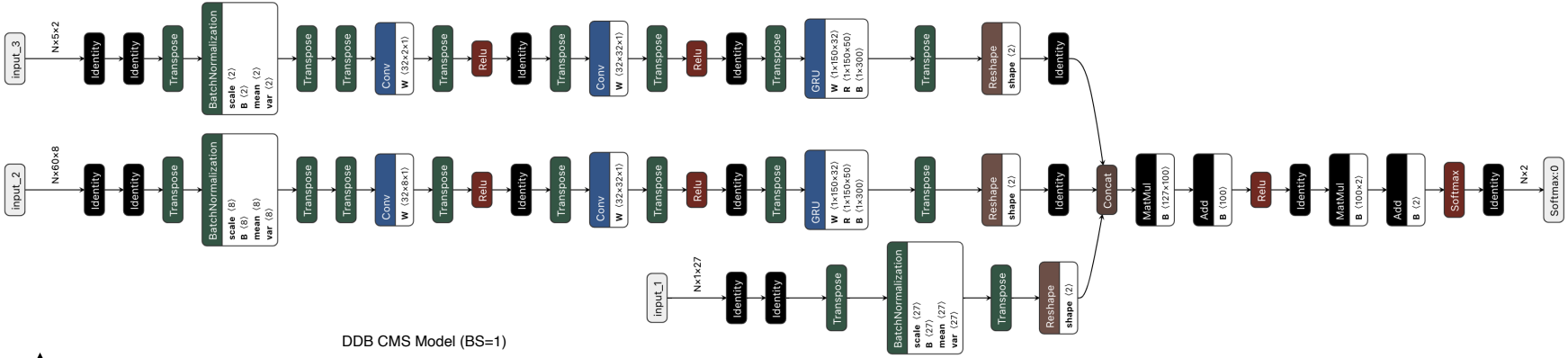
Example Notebooks and Tutorials

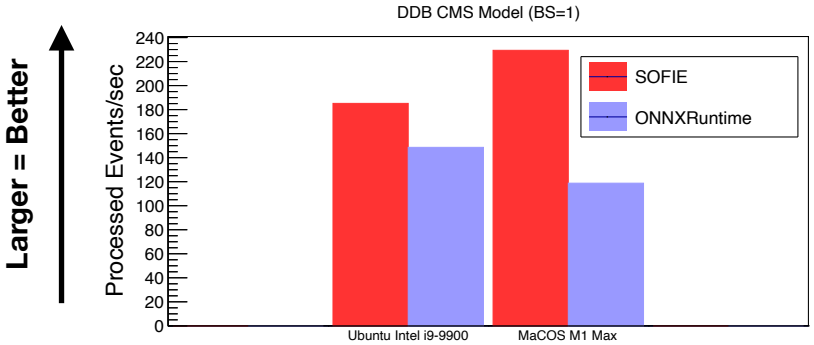
- Example notebooks on using SOFIE:
  - https://github.com/lmoneta/tmva-tutorial/tree/master/sofie
- Tutorials are also available in the tutorial/tmva directory
- Link to SOFIE code in current ROOT master in GitHub
- Link to benchmarks in *rootbench*

Backup Slides

Benchmark using a CMS Model

- Preliminary results using CMS Deep Double model (DDB.onnx)
  - 3 inputs with 1d Conv + GRU





Benchmark Conclusions

- 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) but further optimisations still possible