SOFIE: C++ Code Generation for Fast Deep Learning Inference

IML Meeting 5/7/2022

Sitong An, Lorenzo Moneta, Sanjiban Sengupta, Ahmat Hamdan, Federico Sossai, Aaradhya Saxena, Neel Shah

ROOT
Data Analysis Framework
https://root.cern

Motivation

- ML ecosystem focus mainly on training the models
- Deployment of models (inference) is often neglected
- Tensorflow/PyTorch have functionality for inference
  - can run only for their own models
  - usage in C++ environment is cumbersome
  - requires heavy dependence
- A new standard exists for describing deep learning models
  - **ONNX** ("*Open Neural Network Exchange*")
- ONNXRuntime: a new efficient inference engine based by Microsoft
  - 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

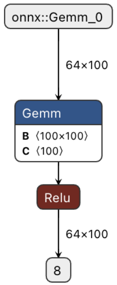
- **SOFIE : S**ystem for **O**ptimised **F**ast **I**nference code **E**mit

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");
```

ONNX



- Parser exists also for (with more limited support)
  - Native PyTorch files (*model.pt* files)

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

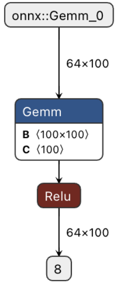
  - Native Keras files (*model.h5* files)

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

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();
```

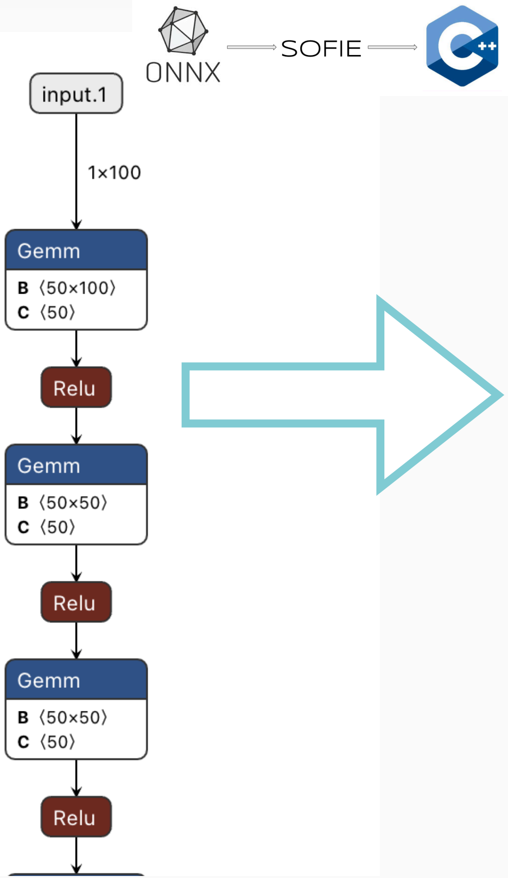


- Generated code has minimal dependency
  - only linear algebra library (BLAS)
  - no dependency on ROOT libraries
  - can be easily integrated in whatever software code

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){
```

Code Generation




```
namespace TMVA_SOFIE_Model {

struct Session {
 Session(std::string filename ="Model.dat") {
   // read weight data file
   std::ifstream f;
   f.open(filename);
   ………………………
 }
 std::vector<float> infer(float* tensor_input1){
   …………………….
   //——— Gemm
   BLAS::sgemm_(……..);  // from tensor_input -> tensor_21

   //------ RELU
   for (int i = 0; i < 50 ; i++)
      tensor_22[i] = ((tensor_21[i] > 0 )? tensor_21[i] : 0);
   ……………………..
   BLAS::sgemm_(…..);
   ………………
   // return output tensor
   std::vector<float> ret (tensor_39, tensor_39 + 10);
   return ret;
}
};
}
```

See tutorial TMVA_SOFIE_ONNX.C

Using the Generated code

- 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 *
   auto result = s.infer(input);
}
```

- 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 Libraries

- Separation between parser and code generation (RModel class)
  - ONNX Parser is in a separate library
    `libROOTTMVASofieParser.so`
    - Dependency on Google Protocol Buffers (a.k.a. Protobuf) for parsing the ONNX file
  - RModel class and code generation is in another library:
    `libROOTTMVASofie.so`
    - Minimal dependency on ROOT (for I/O)
    - Model can be serialised and stored in a ROOT file !
- Emitted C++ code requires only a linear algebra library (BLAS)
  - optionally vdt for efficient Math functions

Minimal dependencies !

ONNX Supported Operators

Gemm	Implemented and integrated (ROOT 6.26)
Activations: Relu, Seul, Sigmoid, Softmax, LeakyRelu	Implemented and integrated
Convolution (1D, 2D and 3D)	Implemented and integrated
Recurrent: RNN, GRU, LSTM	Implemented and integrated
BatchNormalization	Implemented and integrated
Pooling: MaxPool, AveragePool, GlobalAverage	Implemented and integrated
Layer operations: Add, Sum, Mul, Div, Reshape, Flatten, Transpose, Squeeze, Unsqueeze, Slice, Concat, Identity	Implemented and integrated
InstanceNorm	Implemented but to be integrated (PR #8885)
Deconvolution, Reduce operators (for generic layer normalisation), Gather (for embedding)	Planned for next release
???	Depending on user needs

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

RDF Integration

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

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

- RDF Interface requires a functor with this signature:

```
T 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");
```

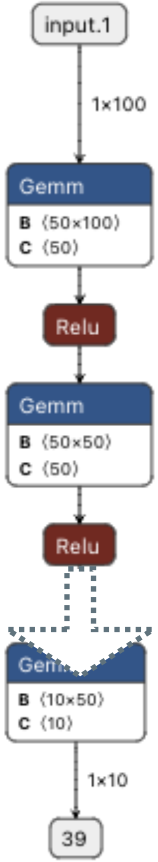
See full Example tutorial code in C++ or Python

SOFIE Functor for RDF

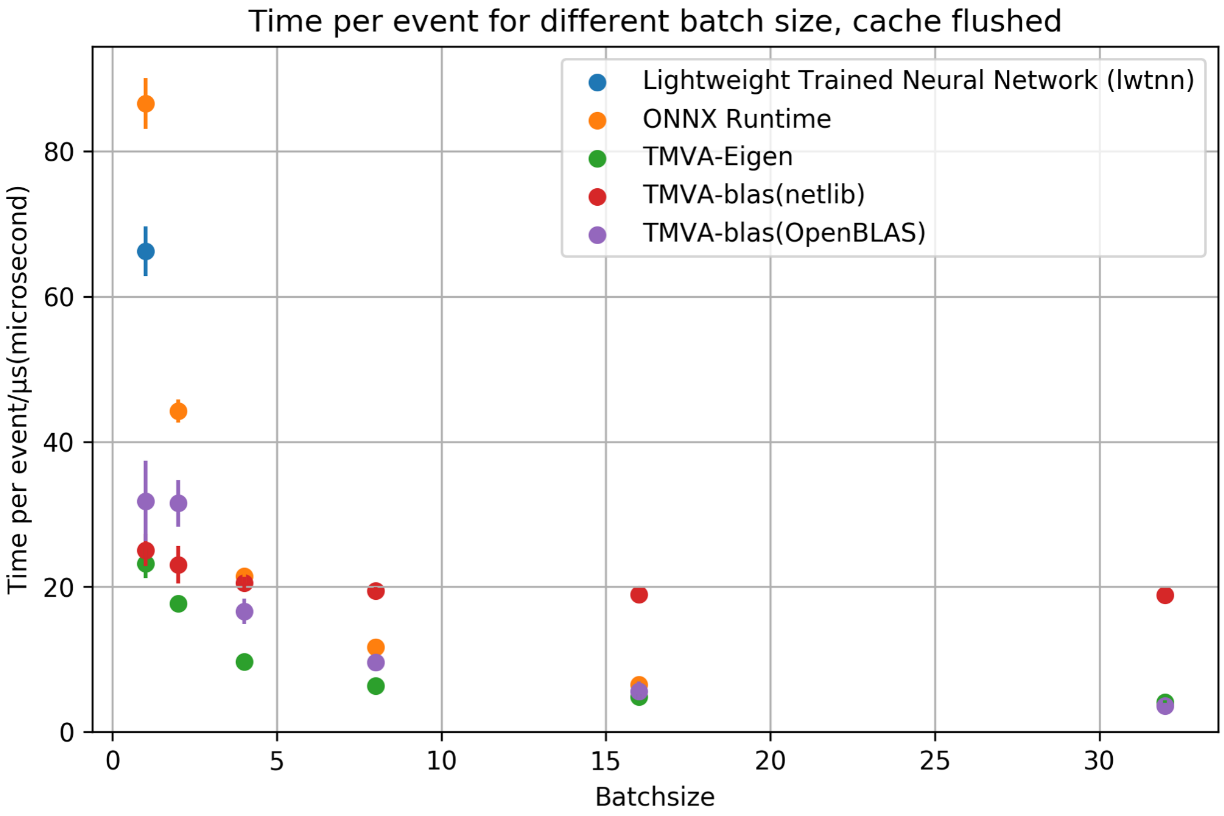
```
template <std::size_t... N,  typename S, typename T>
struct SofieFunctorHelper<std::index_sequence<N...>, S, T> {
   // use index_sequence to define an operator () with N fixed parameter arguments
  …….. 
  // Constructor: create vector of Sessions
SofieFunctorHelper(int nslots, const std::string & filename = “") {
  …….
   for (unsigned int i = 0; i < nslots; i++)
     fSessions.emplace_back(filename);
   }


double operator()(unsigned slot, AlwaysT<N>... args) {
      fInput[slot] = {args...};
      auto y =  fSessions[slot]->infer(fInput[slot].data());
      return y[0];
   }
};


template <std::size_t N, typename F>
auto SofieFunctor(int nslot)->SofieFunctorHelper<std::make_index_sequence<N>, F, float>
{
   return SofieFunctorHelper<std::make_index_sequence<N>, F, float>(nslot);
}
```

Full code available here.

Benchmark: Dense Model

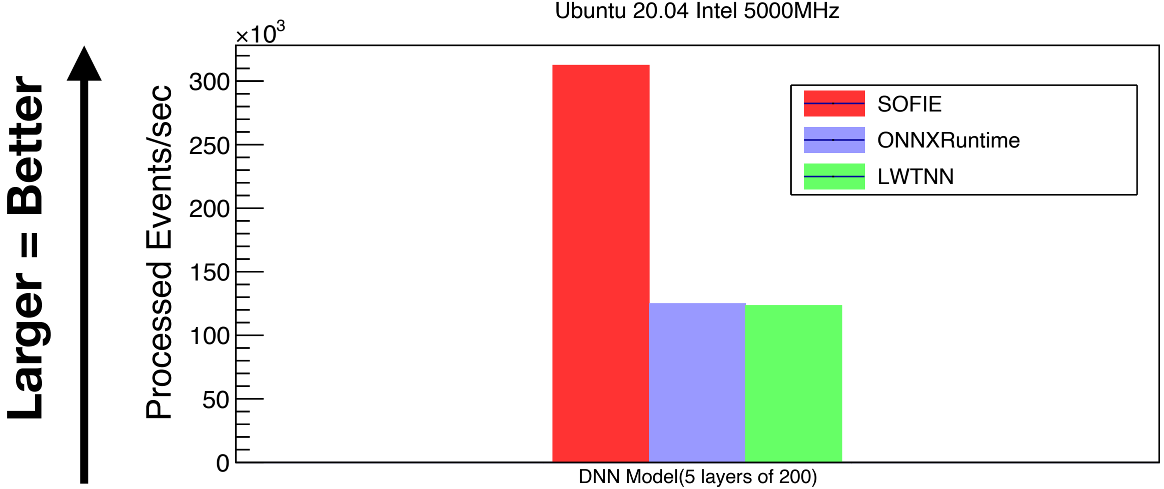


10 Dense layers



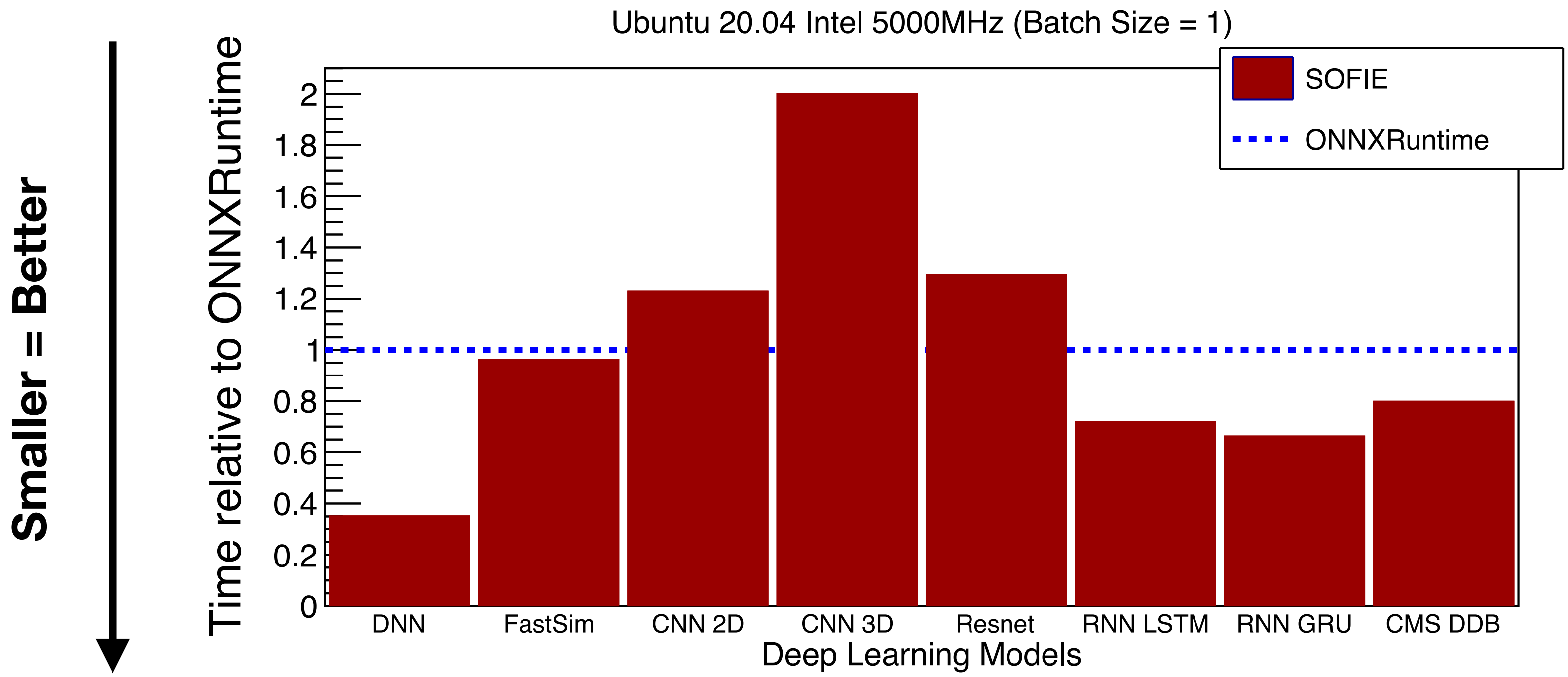
Benchmark with RDF

- 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 Multi-Threads



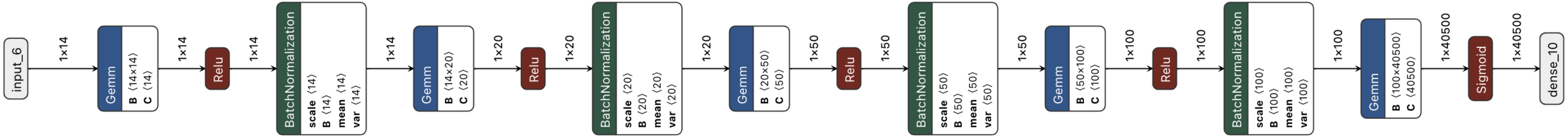
Benchmark: All Models (on Linux PC)

- Test event performance of SOFIE vs ONNXRuntime (BS=1)



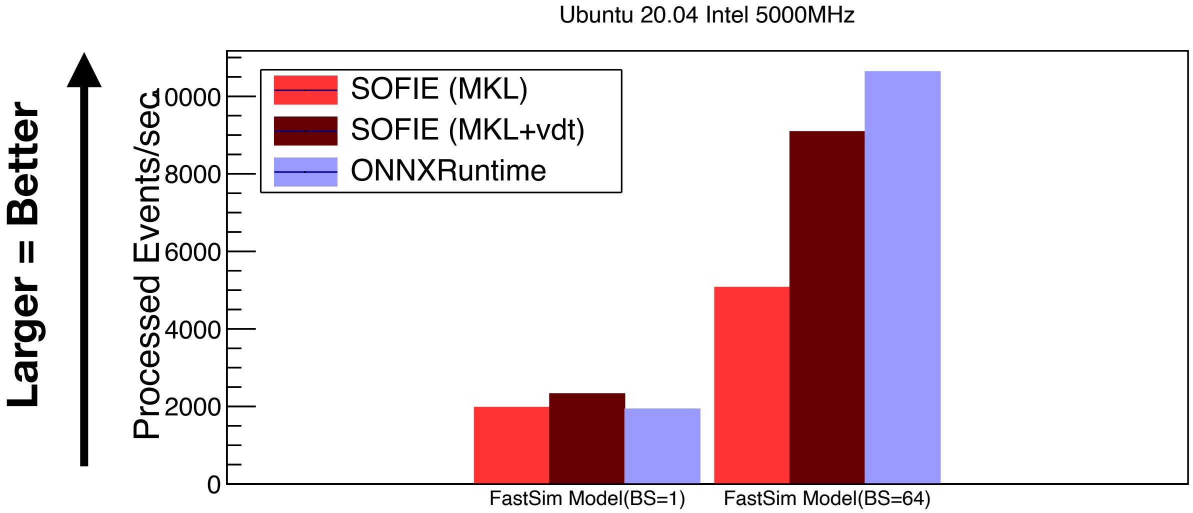
Benchmark using a FastSim Model

- Using ONNX model from a G4 example (Par04)
  - dense layer with Relu and Batch norm. layers (a decoder model)



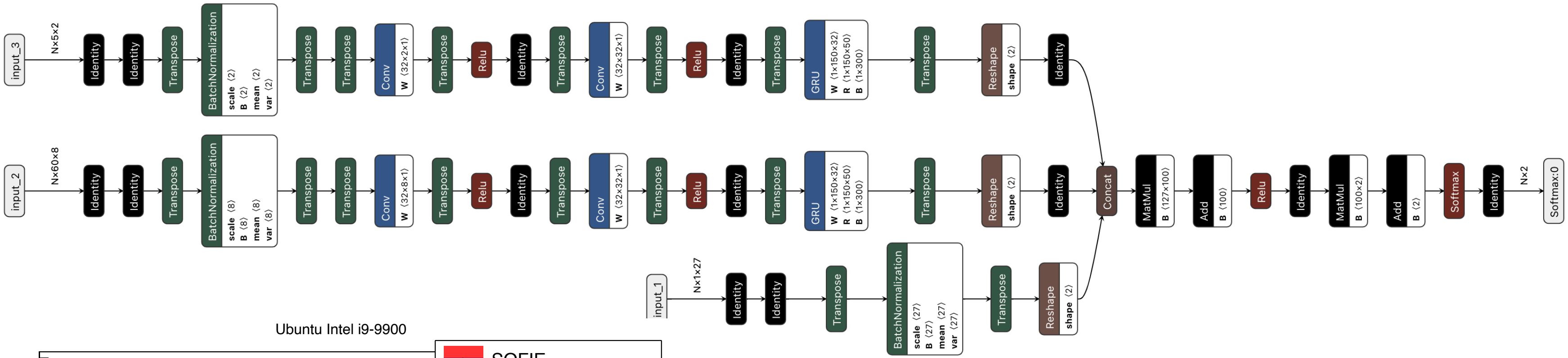
100 x 40500 weight matrix

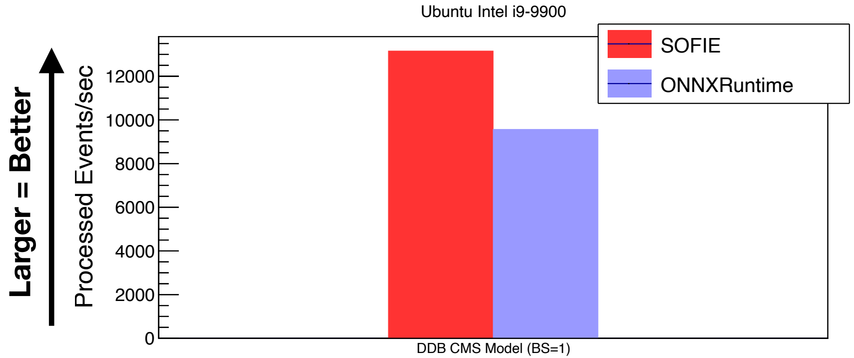
output sigmoid BS x 40500



Benchmark using a CMS Model

- Results using CMS Deep model for jet tagging (DDB)
  - 3 inputs with 1d Convolutions (32x32x1) + GRU (32x150 and 50x150)





one of the model presented here

Future Plans

- Implement some missing operators:
  - Deconvolution, etc..
  - User defined operators
  - more depending on user needs and feedback
- Improve parser for Keras models
  - adding Batch normalisation
- Integration with new RReader class for TMVA Inference
  - Have a user interface starting from an input model and performs internally JIT-ing of code
  - Example : RBDT interface for fast BDT

```
TMVA::RBDT bdt("myBDT", "model.root");
auto x = TMVA::RTensor<float>(data, shape);
auto y2 = bdt.Compute(x);
```

Other Possible Developments

- Implement further optimisations:
  - layer fusions, quantisations,….
  - we are in contact with *hls4ml* project for collaborating
- Generate code for different architectures (e.g GPU)
- Investigate extensions to parse and generate code for graph models (GNN)
  - not supported by ONNX , will parse directly saved models
- Store model weights in a binary file (e.g. ROOT format)
  - currently using a simple text file or including weights directly in header file declaration
- Have SOFIE as an independent package
  - give possibility of frequent version updates
  - ROOT dependency for model serialisation could be optional

if requested, it can be done

Summary

- First release of SOFIE, fast and easy to use inference engine for ML models, is available in ROOT 6.26
- Good performance compared to existing package (ONNXRuntime) and LWTNN
  - further optimisations are still possible
- Integrated with other ROOT tools to evaluate models in user analysis (*RDataFrame)*
- Planning to use with existing TMVA RReader class
- Future developments will be done according to user needs and the received feedback!

Example Notebooks

- Some example notebooks on using SOFIE:
  - https://github.com/lmoneta/tmva-tutorial/tree/master/sofie
- Some tutorials are also available in the tutorial/tmva directory

Conclusion

- Link to SOFIE in current ROOT master
- Link to TMVA/SOFIE tutorials
- Link to SOFIE notebooks
- Link to benchmark in rootbench (PR #239)
- Link to previous benchmark sample code

The presenter gratefully acknowledges the support of the Marie Skłodowska-Curie Innovative Training Network Fellowship of the European Commission Horizon 2020 Programme, under contract number 765710 INSIGHTS.