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

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

presented by Enrico Guiraud

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

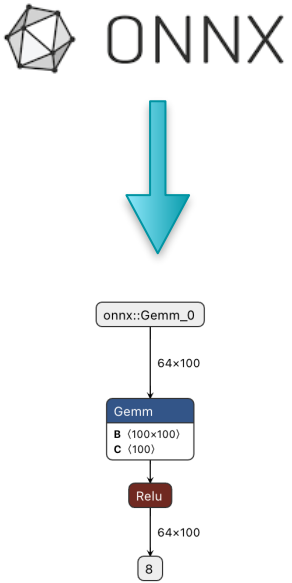
  - **Output**: Generated C++ code that hard-codes the inference function
    - Easily invokable directly from other C++ project (plug-and-use)
    - Minimal dependency (on BLAS only)
    - Can be compiled on the fly using Cling JIT

- **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");
```



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

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

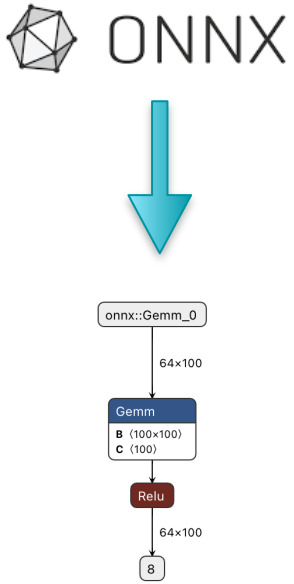
  - Native Keras files (*model.h5* files)

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

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");
```



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

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

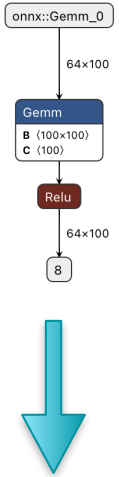
  - Native Keras files (*model.h5* files)

```
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();
```



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

- 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 *
   std::vector<float> 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

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");
```

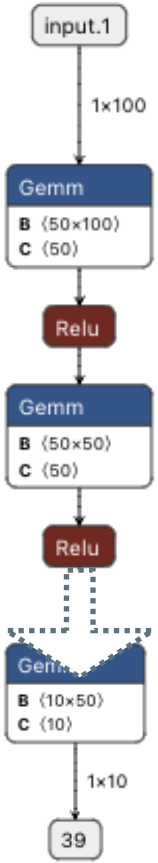
See full Example tutorial code in C++ or Python

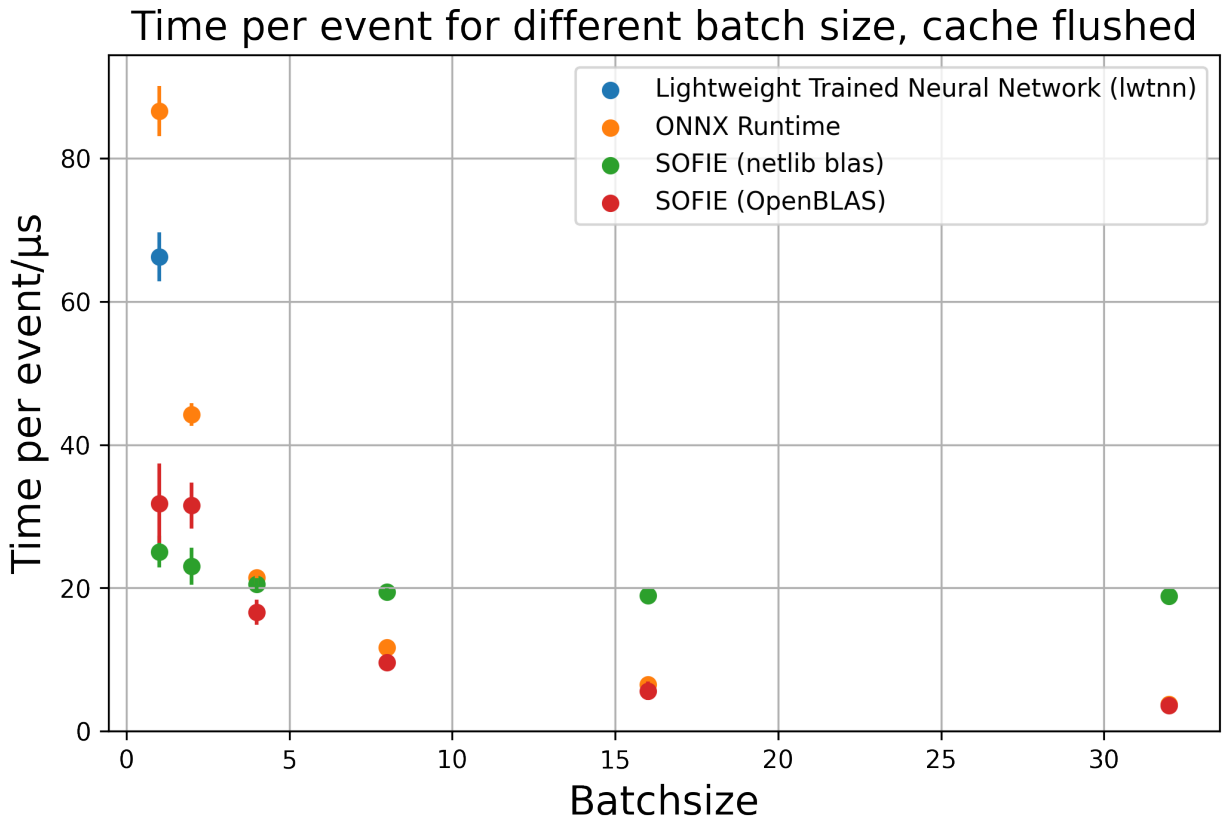
ONNX Supported Operators

Perceptron: 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

Benchmark: Dense Model

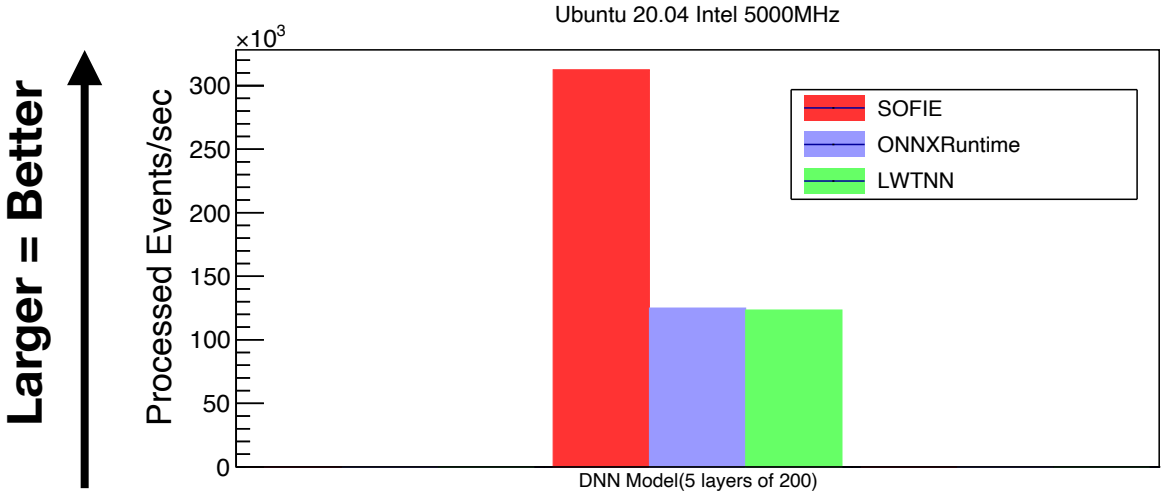
10 Dense layers





Benchmark with RDF

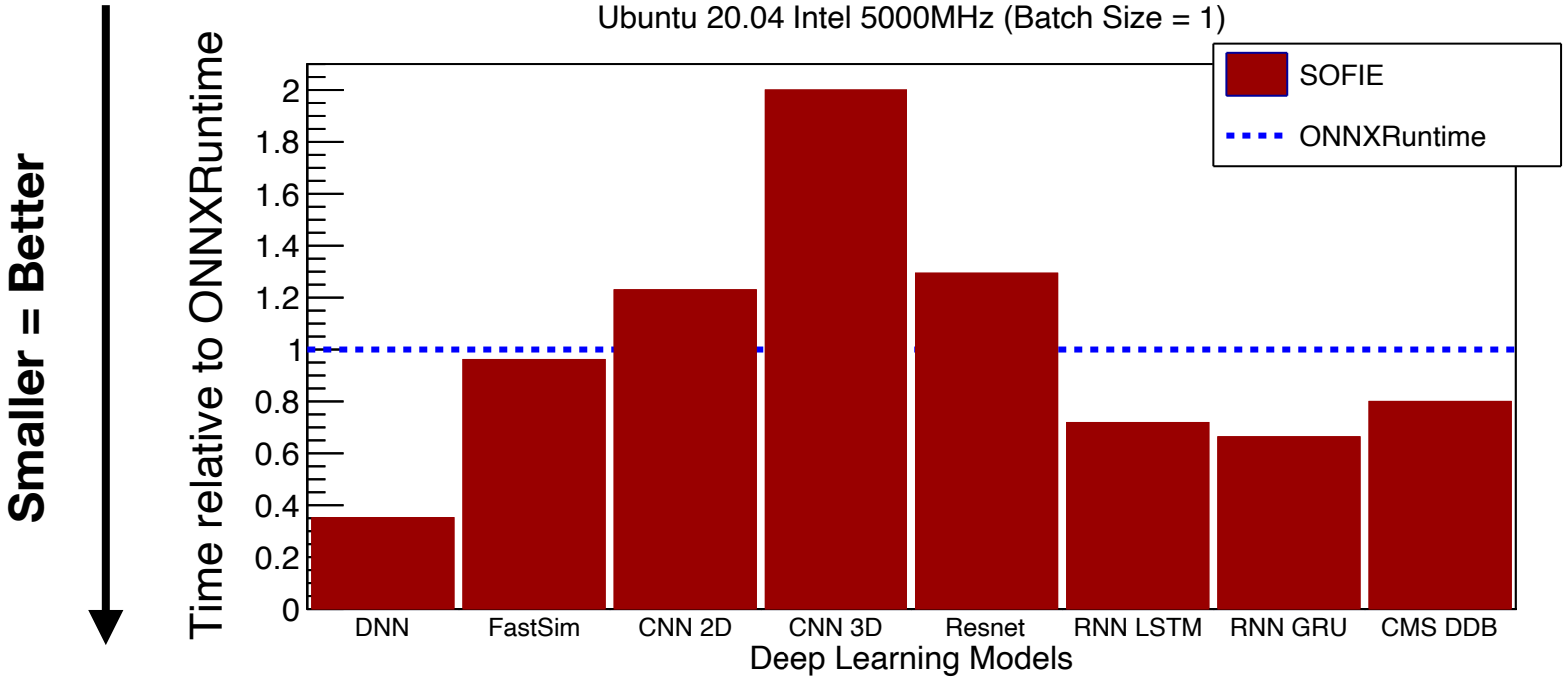
- Test on a Deep Neural Network (from TMVA_Higgs_Classification.C tutorial)
  5 fully connected layers of 200 units
- Run on dataset of 5M events:
  - Single Thread, but can run Multi-Threads



Benchmark: All Models (on Linux PC)

- Test event performance of SOFIE vs ONNXRuntime

(using batch size = 1)



Future Plans

- Implement some missing operators:
  - Deconvolution, etc..
  - more depending on user needs and feedback
- Implement same model 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

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*
- Future developments will be 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*

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.

Conclusion

- Link to SOFIE in current ROOT master
- 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.