C++ Code Generation for Fast Inference of Deep Learning Models in ROOT/TMVA

Sitong An1,2, Lorenzo Moneta1, Sanjiban Sengupta3, Ahmat Hamdan4, Federico Sossai5, Aaradhya Saxena6

1 CERN, Esplanade des Particules 1, 1211 Meyrin, Geneva, Switzerland
2 Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, U.S.
3 IIT Bhubaneswar, Gothapatana, Malipada, Bhubaneswar, Odisha 751029, India
4 ISSEA, Rue Pasteur, Yaoundé, Cameroon
5 University of Padova, Via VIII Febbraio, 2, 35122 Padova PD, Italy
6 IIT Roorkee, Roorkee - Haridwar Highway, Roorkee, Uttarakhand 247667, India

s.an@cern.ch

Abstract. We report the latest development in ROOT/TMVA, a new tool that takes trained ONNX deep learning models and emits C++ code that can be easily included and invoked for fast inference of the model, with minimal dependency. An introduction to SOFIE (System for Optimized Fast Inference code Emit) is presented, with examples of interface and generated code. We discuss the latest expanded support of a variety of neural network operators, including convolutional and recurrent layers, as well as the integration with RDataFrame. We demonstrate the latest performance of this framework with a set of benchmarks.

1. Introduction

Since 2005, as part of the ROOT Data Analysis Framework [1], the Toolkit for Multivariate Analysis (TMVA) [2] has provided an environment for the training and evaluation of a large variety of machine learning methods for data analysis in High Energy Physics and other scientific fields. For example, TMVA provides the training and inference of boosted decision trees (BDTs), which have been a popular algorithm for classification and regression among high energy physicists, contributing even to the Higgs discovery in 2012 [3-6].

The emergence of contemporary neural network design has revolutionized machine learning in recent years. With the emergence of deep learning, large technological companies' software solutions, such as TensorFlow, MXNet, and PyTorch [7-9], began to emerge and gradually dominated the landscape. Nowadays, numerous high-energy physics workflows, such as the CMSSW [10], have embraced and integrated these external technologies.

SOFIE [11], the latest development work in ROOT/TMVA, aims to provide a convenient solution for users to conduct inference of deep learning models trained by these frameworks in a C++ based production environment. It takes a trained ONNX [12] model as input and outputs snippets of C++ code that hard-code the inference function in a header file. This function can therefore be readily included and run from any C++ project, with the only dependency on linear algebra libraries such as BLAS (Basic Linear Algebra Subprograms). A dependency on Google Protocol Buffers (protobuf) is required for
parsing the ONNX model file, although it is not required for using the output C++ code. This allows for an easy integration of the generated code in other C++ based HEP software frameworks.

There have been plenty of effort in this area, including frameworks developed by tech companies such as ONNX Runtime [13] as well as solutions developed by the high energy physics community, such as Lightweight Trained Neural Network (lwtmn) [14]. Aside from software that provides inference on CPUs and GPUs, inference tools that specialize on other hardware, such as hls4ml [15], which emits FPGA implementations of machine learning algorithms, have been developed in recent years. Another train of thought is to provide inference as a service with integrated GPUs and FPGAs, such as SONIC [16]. Despite the many options available, many of the solutions still have a lot of dependencies or come with a runtime that is memory-heavy. SOFIE provides an alternative lightweight, plug-and-use solution in this front.

In particular, SOFIE now accepts ONNX, Keras, PyTorch and ROOT models. We have also expanded the supported operators to include Conv, Pool, RNN, GRU, LSTM, BatchNorm others. Note that SOFIE is designed to be modular, so that users can easily contribute to it by adding custom operators without having to understand the core of the framework. SOFIE is also designed to be thread-safe so that multi-threaded support could be vastly expanded in the future.

2. Examples of Interface and Generated Code

SOFIE is easy to use. To generate code, the user first needs to use a parser to parse the neural network format into an RModel object. Different parsers are being developed to support ONNX, ROOT, Keras and PyTorch formats to varying degrees. From the RModel object, we can easily generate code.

```cpp
using namespace TMVA::Experimental;
SOFIE::RModelParser_ONNX parser;
SOFIE::RModel model = parser.Parse("model.onnx");
model.Generate(); // generate output header and weights
model.OutputGenerated();
```

Fig. 1. Code snippet demonstrating the interface of SOFIE for generating code.

As this code is run a header file named “model.hxx” is generated. To infer with this generated code, simply include it and call the pre-defined function.

```cpp
#include "model.hxx"

TMVA_SOFIE_model::Session s;
std::vector<float> x = {...};
auto y = s.infer(x.data());
```

Fig. 2. Code snippet demonstrating the use of the generated inference code.

A closer look into the generated code reveals that SOFIE works by unrolling the logics of each operator to plain C++ code that allows maximal compiler optimization. For example, the new inference engine parses a general transpose operator as described by the ONNX Operator standard, with the hyperparameter permutation set to [3,2,1,0] (i.e., it permutes a tensor of shape [1,2,3,4] to a new shape of [4,3,2,1]), and emits the code snipper below.
3. Integration with Datagram

As part of modern analysis workflow in ROOT, users can easily carry out multi-threaded inference of their deep learning models on data in RDataFrame.

![Integration with Datagram](image)

Fig. 4. Code snippet demonstrating the integration of SOFIE with RDataFrame.

With the functor defined as:

```cpp
// Functor to wrap SOFIE session to RDF functor signature
template <typename Func>
struct SofieFunctor {
    std::vector<float> input;
    std::vector<std::shared_ptr<Func>> sessions;
    SofieFunctor(unsigned nslots) :
        input(7*nslots,0) {
        for (int i = 0; i < nslots; i++) {
            sessions.push_back(std::make_shared<Func>());
        }
    }
};
```

Fig. 5. Code snippet demonstrating the functor that wraps SOFIE session for RDF inference.

A preliminary benchmark for inference on RDataFrame has been conducted, with a model that consists of 7 inputs, 5 fully connected layers of 200 units with RELU activation functions. The model is evaluated in a single thread using RDataFrame and SOFIE, ONNX Runtime or LWTNN as inference engine.
4. Performance Benchmark

Performance benchmark of SOFIE on linear/dense/fully connected layers has been demonstrated before and presented at CHEP 2021 [17]. Here we reproduce the results performed on a fully connected network with 10 layers of 50 neurons wide each, coupled with the ReLu activation function.

Fig. 5. Benchmark results for conducting inference on RDataFrame data with SOFIE and other frameworks.

| SOFIE/µs | ONNXRuntime/µs | LWTNN/µs |
|----------|----------------|-----------|
| 3.2      | 8.0            | 8.1       |

Fig. 5. Benchmark results for dense layers.

Our latest development work on SOFIE included expanded support for convolutional neural network. Here, we performed inference on a convolutional network of filter size (5,5) and channel number varying from 1 to 128, on input data consisting of images of size (100,100). We have also performed benchmark on a full-fledge CNN architecture of resnet18.

Fig. 6. Benchmark results for conv layers.

| Model        | SOFIE/ms | ONNXRuntime/ms |
|--------------|----------|----------------|
| 1xConv, Batch=1 | 0.05     | 0.08           |
| 14xConv, Batch=1 | 126      | 100            |
| 14xConv, Batch=32 | 50       | 49             |
| Resnet18     | 44       | 34             |

Fig. 6. Benchmark results for conv layers.

Here, a similar pattern to that of linear layers emerge. For simpler models like a 1-layer conv-based network, we achieved better performance than ONNX Runtime. For more complex models with higher no. of layers and more complicated flow of data through the network operators, we achieve slightly worse performance comparable with ONNX Runtime. The likely reason for this pattern is that ONNX
Runtime maintains a running process during inference process that allows for greater optimization, but at a cost of greater overhead. For larger, more complex model, this is more likely to pay off.

5. Conclusion and Future Work

We reported the latest development work regarding the inference code generation engine SOFIE in ROOT/TMVA. Specifically, we reported on the expanded support for input data format (ROOT, Keras, PyTorch) and operators (Conv, RNN, LSTM, GRU, BatchNorm...) as well as integration with RDataFrame for convenient inference of data in modern ROOT data analysis workflow. The latest SOFIE has been included in ROOT 6.24 experimental.

As the next step in our plan of development, we aim to invest greater effort in further inference speed optimization, including deeper investigation of operator-level optimization and compiler optimization. We would also like to expand operator support according to our users’ feedback and demand. Finally, we intend to improve interoperability with other ROOT modern analysis facilities such as RDataFrame.

References

[1] R. Brun, F. Rademakers, Nucl.Instrum.Meth.A 389 (1997) 81-86
[2] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, M. Backes, T. Carli, O. Cohen, A. Christov et al., TMVA - Toolkit for Multivariate Data Analysis (2007), physics/0703039
[3] S. Chatrchyan et al. (CMS), Phys. Lett. B710, 403 (2012), 1202.1487
[4] The ATLAS collaboration, Evidence for Higgs Boson Decays to the \( \tau^+\tau^- \) Final State with the ATLAS Detector (2013)
[5] S. Chatrchyan, V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan et al., Physics Letters B 716, 30–61 (2012)
[6] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdelkhalek, A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins et al., Physics Letters B 716, 1–29 (2012)
[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., TensorFlow: A system for large-scale machine learning (2016)
[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., PyTorch: An Imperative Style, High-Performance Deep Learning Library (2019)
[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems (2015), 1512.01274
[10] CMSSW, GitHub repository, http://cms-sw.github.io
[11] ROOT/SOFIE, GitHub repository, https://github.com/root-project/root/tree/master/tmva/sofie
[12] B. Junjie, L. Fang, Z. Ke et al, ONNX: Open Neural Network Exchange (2019)
[13] ONNX Runtime Developers, ONNX Runtime (2021), https://onnxruntime.ai/
[14] D. Guest, J. Smith, M. Paganini, M. Kagan, M. Lanfermann, A. Krasznahorkay, D. Marley, A. Ghosh, B. Huth, Lightweight Trained Neural Network (2020), DOI 10.5281/zenodo.4310003. GitHub repository, https://github.com/lwtnn/lwtnn.
[15] J. Duarte et al., JINST 13 P07027 (2018), Fast inference of deep neural networks in FPGAs for particle physics
[16] K. Pedro, SonicCMS. GitHub repository, https://github.com/cms-sw/cmssw/tree/master/HeterogeneousCore/SonicCore.
[17] S. An, L. Moneta, C++ Code Generation for Fast Inference of Deep Learning Models in ROOT/TMVA, EPJ Web Conf. 251 (2021), CHEP 2021