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

*Sitong* An[1,2] , and *Lorenzo* Moneta[1]

[1]CERN, Esplanade des Particules 1, 1211 Meyrin, Geneva, Switzerland
[2]Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, U.S.

**Abstract.** We report the latest development in ROOT/TMVA, a new system 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. We present an overview of the current solutions for conducting inference in C++ production environment, discuss the technical details and examples of the generated code, and demonstrates its development status with a preliminary benchmark against popular tools.

## 1 Introduction

Since 2005, the Toolkit for Multivariate Analysis (TMVA) [1] has been part of the ROOT Data Analysis Framework [2]. It provides 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, before the now-popular machine learning tools and platforms developed by the wider community, such as Scikit-learn, are available. 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 physicist, contributing even to the Higgs discovery in 2012 [3-6].

In recent years, the rise of modern neural network architecture has revolutionised the field of machine learning. With the rise the deep learning, software solutions supported by large technology companies started to emerge and gradually dominated the landscape, such as TensorFlow, MXNet and PyTorch [7-9]. Nowadays, these external tools have been adopted and integrated into many high energy physics workflow, such as in the CMSSW[10].

While there is a plethora of software support for the training and development of deep neural network, solutions for doing inference of the trained model on real data in a production environment have been lacking for some time. While one can always use the original framework with which the model is trained to carry out the inference, in a C++ based production environment this is usually cumbersome to set up and awkward to use. This is especially so in the context of high energy physics, where the emphasis is usually on the event loop. Recently, there has been a lot of effort in making up this missing link. This paper provides a brief overview of the recent developments in this field, and provides a new development in TMVA to provides an alternative solution. We will discuss the most salient

features of this new c++ code generation system for fast inference of deep learning models in TMVA and demonstrate its preliminary benchmark.

## 2 Overview of Deep Learning Inference Solutions

With the rise of competing deep learning frameworks such as MXNet, TensorFlow and PyTorch, the difficulty in interchanging trained models in different, mutually incompatible formats arises. ONNX [11], which stands for Open Neural Network Exchange, is an open standard for defining deep learning models with the aim of improving interoperability. It is supported natively in PyTorch, and converters from TensorFlow and Keras models to ONNX models have also been provided. Currently, it supports the most popular deep learning operators and can be used to describe the vast majority of modern deep learning architecture.

Together with ONNX, an open source project aiming to accelerate deep learning inference across different frameworks, operating systems and hardware platforms has been developed with the support of Microsoft. This project is the ONNX Runtime [12]. Before carrying out the inference, ONNX Runtime also optimises the model for best inference speed. With the provision of C++ interface by ONNX Runtime, there has been some efforts in integrating ONNX Runtime into analysis frameworks in high energy physics. As ONNX Runtime provides its inference functionality via a session, any production workflow that wishes to make use of ONNX Runtime will have to manage properly this dependency on ONNX Runtime and the dependencies thereof.

Recently, major deep learning frameworks such as PyTorch and TensorFlow have proposed more compilation-based solutions for inference. For example, TorchScript is a recent development in PyTorch that allows the easy creation of serializable and optimizable models that can be invoked in a standalone C++ program in a production environment. Nevertheless, to use the generated TorchScript model, the C++ program will still necessarily have dependency on the PyTorch C++ core library (libtorch). Similarly, TensorFlow has developed XLA (Accelerated Linear Algebra) that aims to compile TensorFlow models into computation kernels written in machine codes highly optimised for the target platform. The generated code has dependencies on the kernels used in the computation.

Besides solutions provided by technology companies in the wider community, there are also excellent options developed closer to home by the high energy physics community. For example, Lightweight Trained Neural Network (lwtnn) [13] provides fast inference on some of the popular deep learning architectures trained with Keras framework. It has only minimal dependency on Eigen and boost PropertyTree. It has been a popular choice for high energy physicist who need to carry out inference in a lightweight manner in the C++ production environment.

Aside from software that provide inference on CPUs and GPUs, the recent years have also seen development of inference tools that specialise on other hardware, such as hls4ml [14], which emits FPGA implementations of machine learning algorithms.

## 3 Code Generation for Fast Inference

Despite the many choices available, it is noted that many of the solutions have still many dependencies or comes with a runtime that has a heavy memory footprint. It is for this reason that we would like to introduce a recent development in TMVA on C++ code generation for fast inference of ONNX-based deep learning models [15]. Specifically, this new TMVA inference engine takes a trained ONNX model as input and emits snippets of C++ code that hard-codes the inference function in a header file. This function can then be easily included and invoked from any C++ project, with the only dependency on BLAS (Basic Linear Algebra Subprograms) for the linear algebra operations. The parsing of ONNX model file would necessitate a dependency on protobuf, but this dependency is not required for using the emitted C++ code.

As an example, a general transpose operator as defined by the ONNX Operator standard, with the hyperparameter permutation set to [3,2,1,0] (i.e., it permutes a tensor with shape [1,2,3,4] to a new shape of [4,3,2,1]), is parsed by the new inference engine and the following code snipper is emitted:

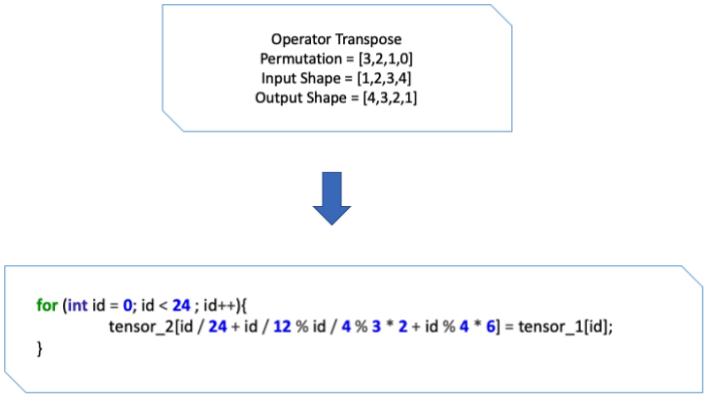


**Fig. 1.** Code snippet demonstrating the use of inference code generation on a transpose operator, with permutation set to [3,2,1,0]. For this particular operator, it has an input size of [1,2,3,4] and an output size of [4,3,2,1].

Note that the evaluations are made explicit for better compiler optimization, such as loop unrolling. Activation functions, such as ReLU, can be similarly implemented:

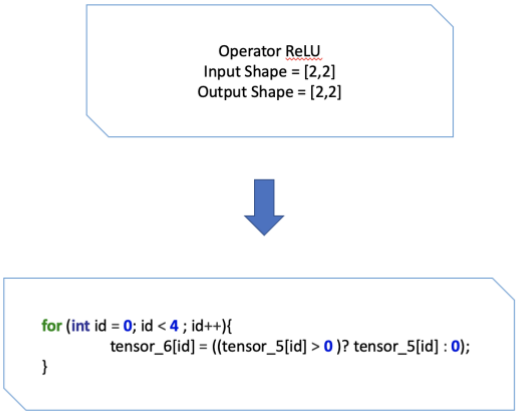


**Fig. 2.** Code snippet demonstrating the use of inference code generation on a ReLU operator. Activation functions are defined in a similar way and exposed to the user's C++ compiler of choice. Support for more activation functions are upcoming. Users can also define their own activation operators with custom evaluations.

More heavy duty linear algebra operators, such as Gemm (matrix-matrix multiplications) are implemented with calls to BLAS. The $(A*B)^T = A^T * B^T$ trick has been applied to call column-major Fortran BLAS functions on row-major C++ arrays.

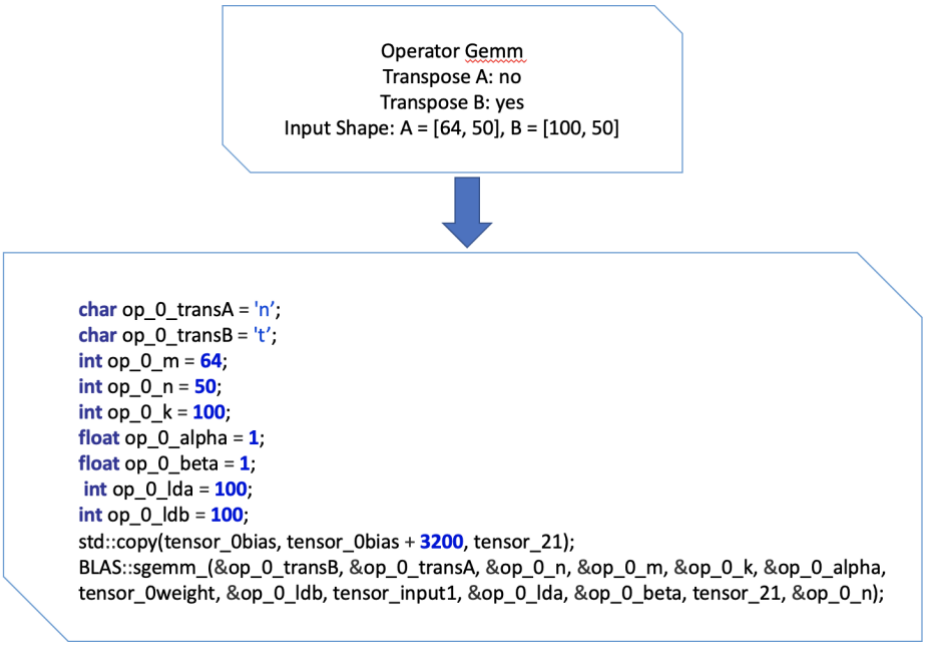


**Fig. 3.** Code snippet demonstrating the use of inference code generation on a Gemm operator with a call to Fortran BLAS.

At the same time, the new inference system also proposes a user-friendly interface, as demonstrated in Fig. 4. The implementation of operators are designed to be modular, so as

to allow for easy support of new neural network architecture in the future. The parsing of the ONNX model is decoupled from the TMVA intermediate representation of deep learning model RModel, so that direct support for more deep learning model support can be developed separately from the core of the new system.

```
RModelParser_ONNX parser;
RModel model = parser.Parse("./Linear.onnx");
model.Generate();
model.OutputGenerated("Linear");
```

**Fig. 4.** Code snippet demonstrating the user interface of the new inference code generation system in TMVA. After running these codes, a header file named "Linear.hxx" is generated in the current directory.

## 4 Preliminary Performance Benchmark

A preliminary benchmark has been performed to compare the performance of lwtnn, ONNX Runtime and TMVA Inference Code Generation. The platform is a CERN Openlab Machine running CentOS 8, with an Intel Xeon CPU E5-2683 v2 at 2.00 GHz and 25 Mb cache size. The benchmarked neural network is a feedforward network with 10 dense layers each with width 50. The number of input features is 100, and 10 output is produced. The neural network is trained with randomly generated training set. Here, TMVA inference code generation is built and linked against standalone netlib Fortran BLAS, as well as OpenBLAS and Eigen. The emitted code is compiled with GCC 10.1.0 at optimization level O3. lwtnn version 2.11.1 is built according to instructions on the GitHub repository, with Boost and Eigen downloaded and built as part of the package. ONNX Runtime 1.6.0 is linked from lcg release LCG 100.

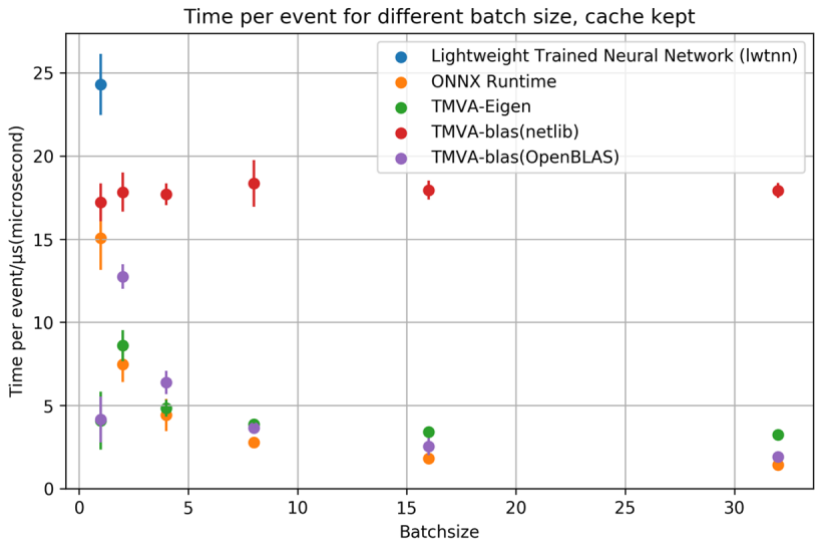


**Fig. 5.** Preliminary benchmarking results for TMVA inference code generation, as compared against lwtnn and ONNX Runtime

All inferences are instructed to be single-threaded only, and for each batch size, 1000 inference runs are done to measure the mean and variance in runtime. For each inference run, random data is generated as input for the network. The time necessary to convert the data into the necessary format for ONNX Runtime and lwtnn is not accounted. Only the time taken to run the inference call is measured.

As seen in Fig. 5., TMVA inference code generation performs at least comparatively to ONNX Runtime at all batch sizes when using Eigen or OpenBLAS as the backend. It outperforms ONNX Runtime by a factor of 3 and lwtnn by a factor of almost 5 at batch size equals to 1 (single event inference). At high batch sizes, the difference in time per event between TMVA inference code generation and ONNX Runtime is very small.

During benchmarking, we noticed that caching seems to play a significant role in determining the temporal cost of inference. We envision that the availability of cache in real life use cases of inference depends on the size of the workflow, and might vary vastly from use case to use case. To better examine to effect of caching on inference, we repeated the benchmarking process with the cache flushed at the end of each inference run, by writing a large vector of random numbers onto the stack.

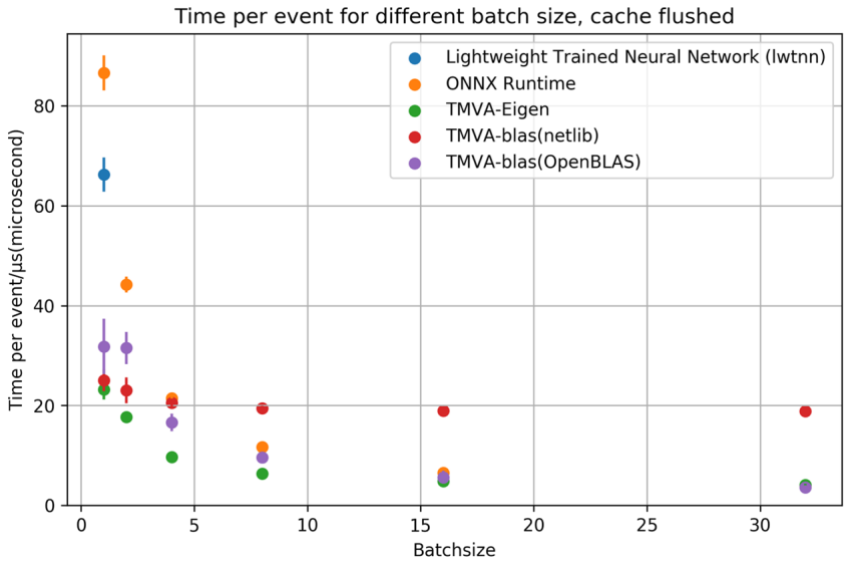


**Fig. 6** Preliminary benchmarking results for TMVA inference code generation, as compared against lwtnn and ONNX Runtime

As seen in Fig.6, with cache flushed, TMVA inference code generation performs at least comparatively to ONNX Runtime at all batch sizes when using Eigen or OpenBLAS as the backend. It outperforms ONNX Runtime and lwtnn significantly at low batch sizes.

## 5 Conclusion and future work

Deep learning model inference in an event loop is probably the most important usage scenario of these different inference solutions for high energy physics. Here, we have demonstrated for this benchmarked neural network architecture that TMVA inference code generation can achieve at least comparable performance to ONNX Runtime at high batch sizes, and significantly outperforms it at low batch sizes.

Currently, the system supports operator Gemm, Relu and Transpose. Operator Conv is already implemented and under testing. In the next few months, we aim to provide support for more operators in this new inference code generation system. Specifically, support for operators RNN, GRU, LSTM, BatchNormalization and InstanceNormalization are foreseen by the end of the summer 2021. We also plan to make further studies into better compiler optimization of the generated code, as well as the memory footprint thereof. We will also update the benchmark for more complex neural network architectures once further operators are supported.

This new system will also be part of the proposed new TMVA machine learning interface[16]. Integration with ROOT for interactive execution by making use of the Cling just-in-time compilation is also foreseen.

As of the submission of this paper, the initial pull request to include this development into ROOT is under review[17][18]. A development prototype can be found in [19] and example benchmarking code can be found in [20].

Sitong An 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.

## References

1. 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
2. R. Brun, F. Rademakers, Nucl.Instrum.Meth.A **389** (1997) 81-86
3. S. Chatrchyan et al. (CMS), Phys. Lett. **B710**, 403 (2012), 1202.1487
4. The ATLAS collaboration, *Evidence for Higgs Boson Decays to the τ+τ− 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. Abdel Khalek, 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. B. Junjie, L. Fang, Z. Ke et al, *ONNX: Opern Neural Network Exchange* (2019)
12. ONNX Runtime Developers, ONNX Runtime (2021), https://onnxruntime.ai/
13. 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.
14. J. Duarte et al., JINST 13 P07027 (2018), *Fast inference of deep neural networks in FPGAs for particle physics*
15. K. Albertsson, S. An, L. Moneta, S. Wunsch, L. Zampieri, EPJ Web of Conferences **245**, 06008 (2020)
16. K. Albertsson, S. An, S. Gleyzer, L. Moneta, J. Niermann, S. Wunsch, L. Zampieri, O. Mesa, EPJ Web of Conferences **245**, 06019 (2020)
17. TMVA SOFIE - Fast Inference Code Generation Initial Commit, GitHub pull request, https://github.com/root-project/root/pull/7544
18. TMVA SOFIE, GitHub repository, https://github.com/sitongan/root/tree/tmva-sofie/tmva/sofie
19. TMVAFastInferencePrototype, GitHub repository, https://github.com/sitongan/TMVAFastInferencePrototype
20. sofie_benchmarking, GitHub repository, https://github.com/sitongan/sofie_benchmarking