Benchmark Studies of Machine Learning Inference using SOFIE

Lorenzo Moneta[1,*], *Sanjiban* Sengupta[1,2], *Ioanna-Maria* Panagou[1,3], *Neel* Shah[4], and *Paul* Wollenhaupt[1,5]

[1]European Organization for Nuclear Research (CERN), Geneva, Switzerland
[2]The University of Manchester, Manchester, United Kingdom
[3]University of Thessaly, Thessaly, Greece
[4]Veermata Jijabai Technological Institute, Mumbai, India
[5]Georg August University of Göttingen, Göttingen, Germany

Abstract.
SOFIE is a fast Machine Learning inference engine developed at CERN, capable of translating trained deep learning models—provided in ONNX, Keras, or PyTorch formats—into C++ code for efficient inference. The generated code has minimal dependencies, making it easily integrable into the data processing and analysis workflows of high-energy physics (HEP) experiments.
This study presents a comprehensive benchmark analysis of SOFIE against leading machine learning frameworks for model evaluation, including PyTorch, TensorFlow XLA, and ONNX Runtime. We focus on evaluating their performance in HEP applications, particularly for typical models such as Graph Neural Networks for jet tagging, and Variational Autoencoders and Generative Adversarial Networks for fast simulation. Our assessment considers key factors such as computational speed, memory usage, scalability, and ease of integration with existing HEP software ecosystems. Through this comparative study, we aim to provide insights that help the HEP community select the most suitable framework for their specific needs.

1 Introduction

Machine Learning inference is becoming increasingly critical across various domains, particularly in HEP, where efficient model evaluation is essential for production workflows. Integrating inference seamlessly into existing software systems, such as reconstruction, simulation, and analysis software, requires support for evaluating models directly within C++ code, beyond the typical Python-based ML environments. Furthermore, effective thread management is crucial for leveraging models in multi-threaded environments, ensuring optimal performance in large-scale data processing tasks. In many HEP applications, inference must be performed at the event level, often requiring single-batch processing while maintaining both computational speed and memory efficiency. Addressing these challenges is key to enabling fast and resource-efficient inference of ML models within complex scientific workflows.

*e-mail: lorenzo.moneta@cern.ch

2 Background

While TensorFlow[1] and PyTorch[2] provide robust inference capabilities, their use in C++ environments presents several challenges. These frameworks are primarily designed around their native model formats, limiting flexibility when integrating externally trained models. Using TensorFlow within a C++ environment is particularly challenging, as its C++ API is not trivial to use and introduces significant dependencies, making deployment more complex. Additionally, TensorFlow's thread management can be challenging to control, and its inference engine is often not optimized for specific use cases, such as single-event evaluation in HEP workflows.

PyTorch, on the other hand, offers the Torch C++ library (LibTorch), which provides a more convenient interface for C++ integration. It is generally easier to install and requires fewer dependencies compared to TensorFlow. However, full support for all PyTorch extensions is not always available, particularly for specialized libraries such as PyTorch Geometric or PyTorch Cluster, which are commonly used for Graph Neural Networks. Furthermore, certain issues arise when converting models from ONNX to the Torch format, limiting the flexibility of model deployment. These constraints highlight the need for a lightweight and efficient inference solution that seamlessly integrates into C++-based data processing pipelines while maintaining high performance and flexibility.

The Open Neural Network Exchange (ONNX)[3] provides a standardized format for describing and sharing deep learning models, facilitating interoperability across different frameworks. However, ONNX cannot fully represent all model architectures, particularly those used in Graph Neural Networks. To enable the efficient inference of ONNX models, Microsoft developed ONNX Runtime[4], an open-source inference engine that supports both C++ and Python environments. It offers flexibility by running on both CPUs and GPUs, with NVIDIA GPU acceleration via TensorRT[5] and AMD support through ROCm[6].

ONNX Runtime has already been successfully integrated into HEP software frameworks, including ATLAS and CMS, where its convenient C++ API and fine-grained thread control have proven valuable. As it is based on the ONNX format, trained models from TensorFlow and PyTorch can be converted for use with ONNX Runtime. However, not all models are fully compatible with ONNX, posing limitations when working with certain architectures. Despite these constraints, ONNX and ONNX Runtime provide a promising solution for deploying machine learning models efficiently within C++-based scientific computing workflows.

3 SOFIE

To address the challenges of efficient machine learning inference in C++ environments, we introduce SOFIE[7]—a tool within ROOT/TMVA[8] designed to generate optimized C++ code from trained ML models. SOFIE is capable of converting models in ONNX format to its own Intermediate Representation (Fig. 1). Additionally, it provides limited support for TensorFlow/Keras and PyTorch models, as well as message-passing Graph Neural Networks from DeepMind's Graph Nets library.

The key advantage of SOFIE is its ability to produce standalone C++ code that can be directly invoked within C++ applications with minimal dependencies—requiring only BLAS for numerical computations. This makes integration seamless for high-energy physics workflows and other computationally demanding applications. Moreover, the generated code can be compiled at runtime using ROOT[9] Cling Just-In-Time compilation, allowing for flexible execution, including within Python environments. By eliminating the need for heavyweight machine learning frameworks during inference, SOFIE offers a highly efficient and easily deployable solution for ML model evaluation.

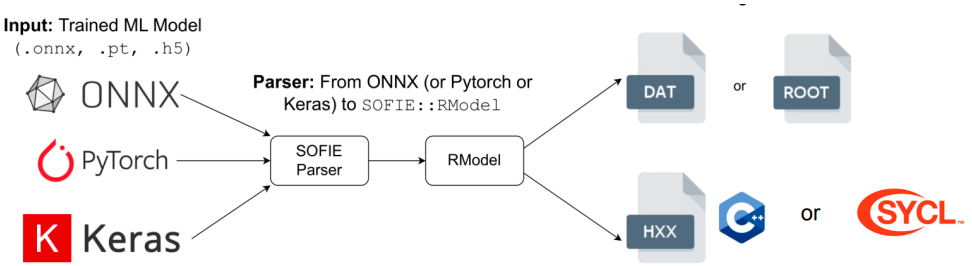


Figure 1: SOFIE Inference Workflow

3.1 GPU Support in SOFIE

To further enhance performance, SOFIE has been extended to generate GPU-accelerated code using SYCL[10], enabling efficient inference on heterogeneous computing architectures. A key design goal is to minimize the overhead of data transfers between the host and the device by optimizing memory management. Buffer allocations are handled efficiently by declaring them at the beginning of execution, reducing runtime overhead.

For GPU offloading, SOFIE leverages optimized linear algebra libraries, including GPU BLAS from Intel oneAPI[11] and PortBLAS[12] for broader GPU compatibility. Performance is further improved by operator fusion, combining multiple operations into a single GPU kernel to reduce memory accesses and computation time. Additionally, to avoid costly conditional branches in GPU execution, SOFIE replaces conditional checks with relational functions, ensuring more efficient instruction flow. These optimizations collectively enable SOFIE to provide high-performance inference on GPUs while maintaining its lightweight and flexible design.

4 Benchmarking Results

4.1 Experimental Setup

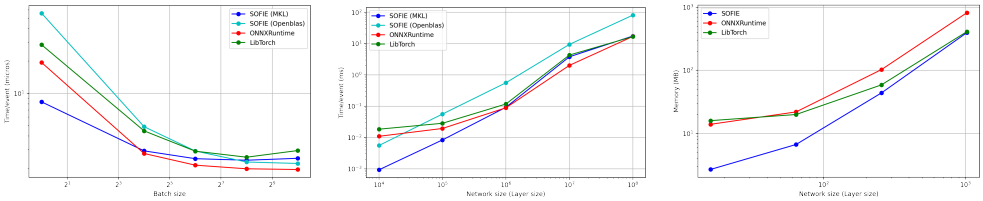
To assess the efficiency of ML model inference, we conducted a comprehensive benchmarking study evaluating both execution speed and memory consumption. Our benchmarks were performed on a standard Linux desktop equipped with an AMD Ryzen processor (24 threads, 4.4 GHz). For consistency, all tests were executed in single-thread mode to isolate the performance characteristics of each framework.

For CPU-based evaluation, SOFIE was tested using two different BLAS implementations: OpenBLAS[13] and Intel MKL (from Intel oneAPI). Comparisons were made against ONNXRuntime (CPU version 1.19.2) and LibTorch (CPU version 2.3.1), providing insight into the relative efficiency of each approach.

Additionally, we conducted GPU benchmarks using the SOFIE development branch with SYCL on an NVIDIA RTX 4090 desktop GPU, highlighting the impact of GPU acceleration on inference performance. Further details on GPU benchmarking methodology and results can be found in our IWOCL paper[10].

4.2 Linear Models

We benchmarked the inference of a linear model consisting of GEMM operators with ReLU and Sigmoid activations. GEMM[14], which stands for General Matrix-Matrix Multiplication, refers to a fundamental linear algebra operation that performs the product of two dense



(a) Time per event as a function of batch size.

(b) Time per event as a function of layer size.

(c) Memory usage as a function of layer size.

Figure 2: Benchmarking a linear model composed of GEMM and ReLU operators, followed by a final Sigmoid operator.

matrices, typically denoted as C = α AB + β C, with A and B as matrix inputs, α and β as scalar inputs, and C as a pre-existing matrix which is overwritten by the output. The test analyzed inference speed as a function of batch size, focusing particularly on single-event evaluation—a common use case in HEP applications. Our results as presented in Fig. 2a indicate that SOFIE, when linked with Intel MKL, outperforms other frameworks for single-event inference, demonstrating lower latency compared to ONNXRuntime and LibTorch. This advantage highlights SOFIE's efficiency in scenarios where real-time, event-level processing is required.

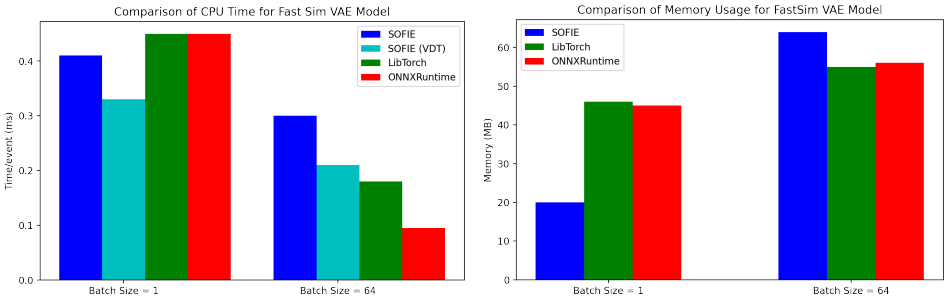
In addition to batch size scaling, we evaluated the inference performance of a simple linear model as a function of size, defined by the number of input and output neurons in the GEMM layer. This test provides insights into how different frameworks handle varying model complexities and on the performances of used Math libraries. Our benchmarks show that SOFIE outperforms other frameworks for smaller model sizes, as shown in Fig. 2b, benefiting from its lightweight design and efficient execution. Performance is further improved when using the Intel MKL BLAS implementation, highlighting the impact of optimized numerical libraries.

Regarding memory usage, SOFIE and LibTorch behave as expected (Fig. 2c), with SOFIE consuming less memory for small models. In contrast, ONNXRuntime exhibits significantly higher memory consumption, using nearly twice as much as SOFIE, making it less efficient for memory-constrained environments.

4.3 Fast Simulation VAE Model

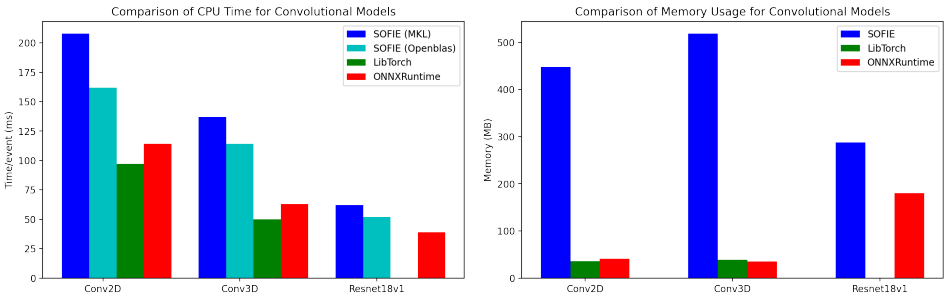
To assess SOFIE's performance on a more complex inference task, we benchmarked the decoder of a Variational Autoencoder, specifically the model used in the Par04 example in GEANT4[15] for fast simulation. The benchmark focused on evaluating CPU inference time and the memory usage, particularly for single-event processing, which is crucial for real-time applications in high-energy physics.

Our results show that SOFIE achieves faster inference for single-event evaluation compared to other frameworks (Fig. 3a). This performance gain is further enhanced by optimizing the Sigmoid function evaluation using vdt::exp, a high-performance mathematical library that provides efficient implementations of transcendental functions. These optimizations demonstrate SOFIE's ability to deliver both speed and efficiency in fast simulation tasks. For memory usage (Fig. 3b), SOFIE exhibits lower overhead for small models, but its memory consumption increases with larger model complexity.



(a) CPU time as a function of batch size (b) Memory usage as a function of batch size

Figure 3: Benchmarking the VAE model used in Par04 example in Geant4



(a) CPU time as a function of batch size (b) Memory usage as a function of batch size

Figure 4: Benchmarking Convolution models comprising of Conv2D with 14 layers, and Conv3D with 5 layers, and ResNet with 20 layers

4.4 Convolution Models

We evaluated the CPU inference performance of convolutional models, including 2D and 3D convolutional networks and a ResNet model (Fig. 4). Our benchmarks indicate that LibTorch and ONNXRuntime outperform SOFIE in these tasks, likely due to their highly optimized convolutional kernel implementations. This suggests that existing deep learning frameworks have a significant advantage in handling convolution-heavy models on CPUs.

We tested a ResNet[16] model with 20 convolutional layers and 224×224 input image sizes, varying the batch size to observe scalability. The results show that SOFIE exhibits extensive memory usage for convolutional networks, primarily due to the lack of intra-layer optimizations. These findings highlight areas where further improvements could enhance SOFIE's efficiency for convolution-based architectures.

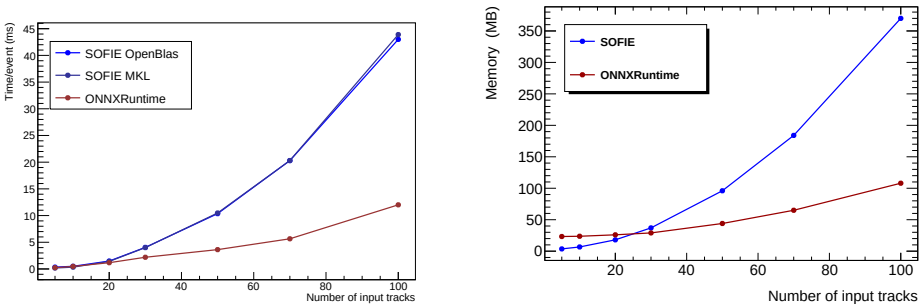
4.5 Graph Neural Networks

We assessed the inference performance of Graph Neural Networks (GNNs) used in high-energy physics experiments, specifically ATLAS, CMS, and a toy model based on Deep-

Mind's Graph Nets library. The benchmarks focused on evaluating scaling behavior, memory consumption, and inference time as a function of input size.

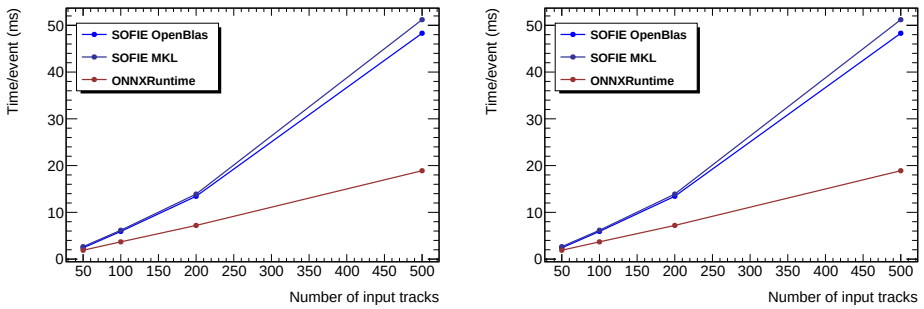
4.5.1 ATLAS GNN Performance

We tested GNN1, a jet-tagging model used in the ATLAS experiment, measuring time and memory consumption against the number of input tracks, with 14 features per track. The results show that SOFIE performs similarly to other frameworks for small input sizes, but ONNXRuntime scales better as the number of tracks increases.



(a) CPU time as a function of input tracks (b) Memory usage as a function of input tracks

Figure 5: Benchmarking ATLAS GNN with 14 input features per track

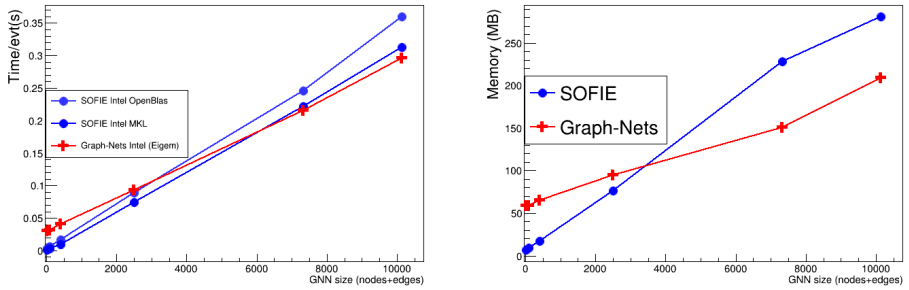


(a) CPU time as a function of input tracks (b) Memory usage as a function of input tracks

Figure 6: Benchmarking CMS ParticleNet with 20 input features per track

4.5.2 CMS GNN Performance

For CMS, we evaluated ParticleNet[17], a GNN model used for jet flavor tagging, measuring inference time (Fig. 6a) and memory usage (Fig. 6b) relative to the number of input tracks. The results show a similar trend to ATLAS GNN, with SOFIE having room for further optimization in memory usage and CPU performance.



(a) CPU time as a function of GNN size (Nodes+Edges)

(b) Memory usage as a function of GNN size (Nodes+Edges)

Figure 7: Benchmarking GNN Toy architecture from LHCb

4.5.3 Graph Nets Library Benchmark

We benchmarked a toy GNN architecture (Fig. 7) inspired by LHCb's usage of DeepMind's Graph Nets[18]. Performance was analyzed based on the scaling of nodes and edges in the graph. Our results indicate that SOFIE is 10 times faster for small GNN sizes, but performance converges for larger models, as inference becomes dominated by matrix operations. Currently, no dedicated memory optimizations have been implemented in SOFIE for GNN inference.

4.6 Summary of Benchmarking

The benchmarking results demonstrate that SOFIE provides faster inference for event-level evaluations and consumes less memory for smaller models. However, performance decreases for larger model sizes due to the lack of optimizations for multi-layer architectures. SOFIE does not yet implement memory usage optimizations or operator fusion from ONNX models, which limits efficiency in complex networks. For Graph Nets-based models, SOFIE scales better as it avoids the overhead introduced by splitting models with a large number of operators, a common issue in ONNX. Comparisons with ONNXRuntime and LibTorch show that both perform similarly in terms of CPU and memory usage, with LibTorch exhibiting slightly lower memory consumption. However, converting models from ONNX to LibTorch remains less flexible.

5 Conclusion

SOFIE is designed as an easy-to-use inference engine for deep learning models, supporting various ONNX operators, including complex GNNs used in high-energy physics experiments. It also extends support to models developed in Deepmind Graph Nets, which are not easily converted to ONNX. Integrated with ROOT[9] tools like RDataFrame, SOFIE enables seamless ML inference for end-user analysis. Its simplicity in both C++ and Python, combined with minimal dependencies, ensures that users have full control over the generated code. Additionally, SOFIE can store models and weights in ROOT format for improved compatibility. A prototype GPU implementation using SYCL has been developed, with future plans to implement memory optimizations, ONNX operator fusion, and extended GPU

support, including potential porting to CUDA or ALPAKA based on user interest. With these ongoing developments, SOFIE is positioned as a promising framework for efficient ML inference in high-energy physics applications.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen et al. 2016. TensorFlow: a system for large-scale machine learning. In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, USA, 265–283.https://dl.acm.org/doi/10.5555/3026877.3026899

[2] Adam Paszke, Sam Gross, Francisco Massa et al. 2019. PyTorch: an imperative style, high-performance deep learning library. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 721, 8026–8037. https://dl.acm.org/doi/10.5555/3454287.3455008

[3] The ONNX Community. ONNX: Open Neural Network Exchange - Open standard for machine learning interoperability. GitHub repository. https://github.com/onnx/onnx.

[4] Microsoft. ONNX Runtime: cross-platform, high performance ML inferencing and training accelerator. GitHub repository. https://github.com/microsoft/onnxruntime.

[5] NVIDIA. TensorRT: SDK for high-performance deep learning inference on NVIDIA GPUs. GitHub repository. https://github.com/NVIDIA/TensorRT.

[6] AMD. ROCm. GitHub repository. https://github.com/ROCm/ROCm.

[7] An S et al (2023) C++ code generation for fast inference of deep learning models in root/tmva. J Phys Conf Ser 2438:012013. https://doi.org/10.1088/1742-6596/2438/1/012013

[8] A. Hoecker, P. Speckmayer, J. Stelzer et al., TMVA - Toolkit for Multivariate Data Analysis (2007), physics/0703039. https://github.com/root-project/root/tree/master/tmva

[9] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. https://doi.org/10.5281/zenodo.3895860

[10] Ioanna-Maria Panagou, Nikolaos Bellas, Lorenzo Moneta et al. 2024. Accelerating Machine Learning Inference on GPUs with SYCL. In Proceedings of the 12th International Workshop on OpenCL and SYCL (IWOCL '24). Association for Computing Machinery, New York, NY, USA, Article 17, 1–2. https://doi.org/10.1145/3648115.3648123

[11] Intel, Intel oneAPI, https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html.

[12] Codeplay Software. portBLAS: Implementation of BLAS using the SYCL open standard. GitHub repository. https://github.com/codeplaysoftware/portBLAS.

[13] OpenMathLib. OpenBLAS: Optimized BLAS library based on GotoBLAS2 1.13 BSD version. GitHub repository. https://github.com/OpenMathLib/OpenBLAS.

[14] NVIDIA Corporation. (2022). Matrix Multiplication Background. Retrieved from https://docs.nvidia.com/deeplearning/performance/pdf/Matrix-Multiplication-Background-User-Guide.pdf

[15] S. Agostinelli and others. 2003. GEANT4 - A Simulation Toolkit. Nucl. Instrum. Meth. A 506, (2003), 250–303. https://doi.org/10.1016/S0168-9002(03)01368-8

[16] He, Kaiming, et al. 'Deep Residual Learning for Image Recognition'. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, https://doi.org/10.1109/CVPR.2016.90.

[17] Qu, Huilin, and Loukas Gouskos. 'Jet Tagging via Particle Clouds'. Physical Review D, vol. 101, no. 5, American Physical Society (APS), Mar. 2020, https://doi.org/10.1103/physrevd.101.056019.

[18] Google DeepMind. Graph Nets library: Build Graph Nets in Tensorflow. GitHub repository. https://github.com/google-deepmind/graph_nets.