

Day 3

1st NextGen Hackathon



NextGen
Next Generation Triggers

```
template <template <class> class F>  
    struct TheBestTeam {
```

```
        F<Person> Jolly Chen, Leonardo Beltrame, Oliver Rietmann,  
                Simone Balducci, Simone Rossi Tisbeni,  
                Luca Ferragina, Aurora Perego, Davide Gadioli;
```

```
};
```



NextGen
Next Generation Triggers

*it doesn't compile btw

What happened today

- Looked at Oliver's SoA implementation based on C++17 template metaprogramming
- Discussed ideas for future work
 - Support for 1 scalar for the whole SoA
 - Fewer template declarations for passing wrappers to kernels
- Brainstormed shared benchmarks for the SoAs:
<https://codimd.web.cern.ch/s/7U6ihzBCc>
- Setup a repo for the benchmarks:
<https://github.com/cern-nextgen/wp1.7-soa-benchmark>

Plan for tomorrow

- See how the 3 implementations can be combined/unified
- Continue implementing the shared benchmarks

Fast ML Inference

Alessandro Crespi, [Jamie Gooding](#), Enrico Lupi,
Lorenzo Moneta, Anastasiia Petrovych,
Sanjiban Sengupta, Jonas Rembser



NexTGen
Next Generation Triggers

What happened today

- [SOFIE standalone](#) and [Docker container](#) now released
- Introduced support for Smartpixels models and extended SOFIE's operator support for:
 - `CastLike` and `Round` operators
 - Dynamic broadcast of tensors
- Integration of Alpaka in SOFIE in progress
- Full integration of SOFIE into RooFit+LHCb simulation-based inference workflow
- Refactoring SOFIE to emit differentiable code
- Work progressing on derivative of wrapper to `sgemm_`
 - Wrote validator for the derivative under test cases

Plan for tomorrow

- Progress with Alpaka integration
- Test more models from users to extend SOFIE operator coverage
- Brainstorm on SOFIE memory optimisation
- Further refactoring of SOFIE to express NN with functions wrapping BLAS calls that only use separate input/output parameters
- Develop unit test in ROOT for differentiating wrappers of BLAS calls
- Work on RooFit tutorial combining SOFIE and simulation-based inference

Fast ML Inference with hls4ml

Dimitrios Danopoulos, Roope Niemi, Vladimir Loncar, Francesco Vaselli

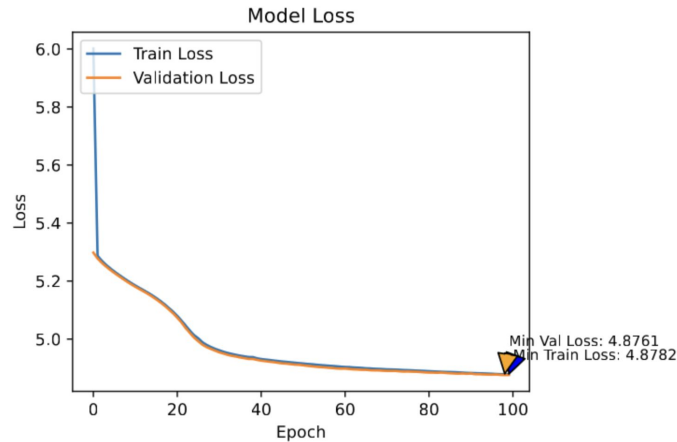


NextGen
Next Generation Triggers

What happened today

NGT WP2.2: Implementation of classification model for L0 MDT Trigger

- PyTorch model compiled in HLS4ML and now synthesizing
 - Fully partitioned implementation
 - Uniform quantization: integer: 6 bits, decimals: 14 bits
 - 0 mismatches with tolerance 0.01



Model: "sequential"

Layer (type)	Output Shape	Param #
C1D 1 (Conv1D)	(None, 216, 12)	300
C1D 2 (Conv1D)	(None, 216, 12)	444
C1D 3 (Conv1D)	(None, 216, 1)	13

Total params: 757 (2.96 KB)
Trainable params: 757 (2.96 KB)
Non-trainable params: 0 (0.00 Byte)

What happened today

fAD: flow anomaly detection for triggers

HLS4ML model parsed fine

Regular densely-connected NN model

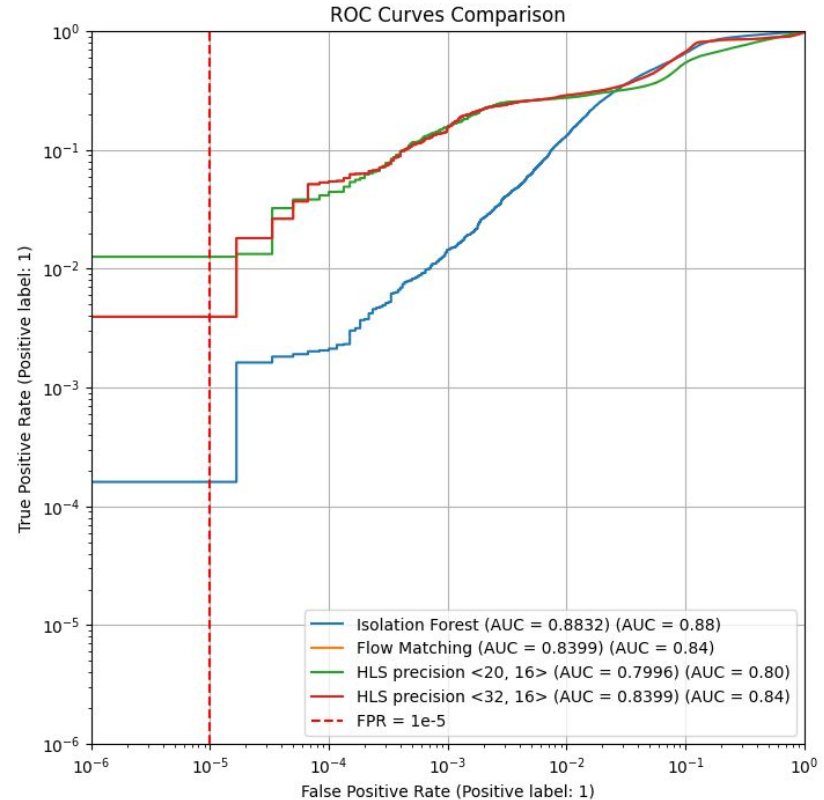
Start synthesizing on the server tomorrow

Weights
weight <32×77>
bias <32>

Weights
weight <32×32>
bias <32>

Weights
weight <32×32>
bias <32>

Weights
weight <76×32>
bias <76>



Plan for tomorrow

NGT WP2.2 model → reduce number of bits, profile layer by layer, apply layer-wise quantization

fAD model → start synthesizing, experiment with PQuant to reduce No. of parameters

Try to parse other models with hls4ml

PQuant: finalize documentation about pruning methods, try to make models trained with PQuant work with hls4ml

Torch inference in heterogenous CMSSW software

Leonardo Beltrame, Lukasz Michalski, [Davide Valsecchi](#), Christine Zeh



NexTGen
Next Generation Triggers

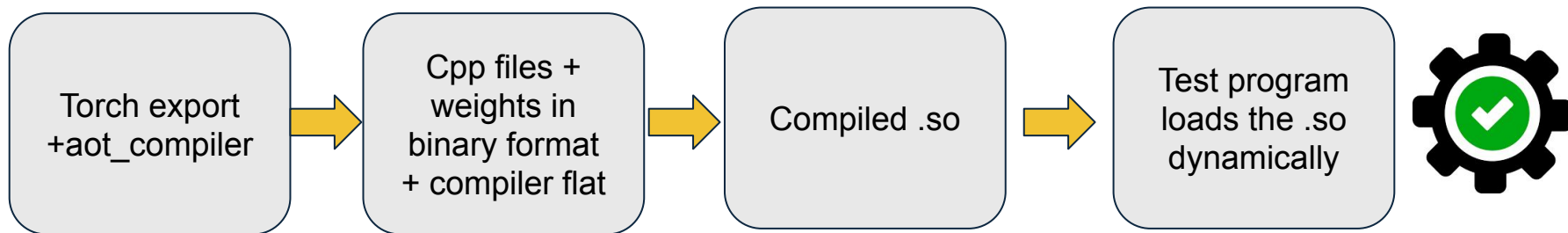
Plan for the week

- **Integrate Torch-Alpaka Inference** in cmssw, open PR to collect feedback
- **Improve interface for SoA to torch::Tensor** improvements possible with merged PR: <https://github.com/cms-sw/cmssw/pull/47306>
- **Profiling & Optimization:**
 - Ensure compatibility with CMSSW's execution model.
 - Ensure we are able to lock Torch internal optimizations (multithreading, Alpaka GPU queues).
- **Explore Ahead-of-Time (AOT) Compilation:** Investigate how AOT can improve model execution instead of relying on just-in-time (JIT) compilation.

TODAY

What was achieved today

Exploring **torch Ahead-Of-Time compiler (AOTinductor)** in the context of the CMS Software.



Found a flag to export the sources instead of compiling directly!



Compile it directly in the CMSSW environment (where libtorch is available)



How it looks

Compile linking to CMSSW headers and libraries

```
import os
import torch

class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = torch.nn.Linear(10, 16)
        self.relu = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(16, 1)
        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

with torch.no_grad():
    device = "cpu"
    model = Model().to(device=device)
    example_inputs=(torch.randn(8, 10, device=device),)
    batch_dim = torch.export.Dim("batch", min=1, max=1024)
    # [Optional] Specify the first dimension of the input x as dynamic.
    exported = torch.export.export(model, example_inputs, dynamic_shapes={"x": {0: batch_dim}})
    # [Note] In this example we directly feed the exported module to aoti_compile_and_package

# Depending on your use case, e.g. if your training platform and inference platform
# are different, you may choose to save the exported model using torch.export.save and
# then load it back using torch.export.load on your inference platform to run AOT compilation.
output_path = torch._inductor.aoti_compile_and_package(
    exported,
    # [Optional] Specify the generated shared library path. If not specified,
    # the generated artifact is stored in your system temp directory.
    package_path=os.path.join(os.getcwd(), "model.pt2"),
    inductor_configs={"aot_inductor.package_cpp_only": True}
)
```

Torch export

```
#!/bin/sh

g++ model.cpp -DTORCH_INDUCTOR_CPP_WRAPPER -DSTANDALONE_TORCH_HEADER -DC10_USING_CUSTOM_GENERATED_MACROS -DCPU_CAPABILITY_AVX512 -I/cvmfs/cms.cern.ch/e19_amd64_gcc12/external/python3/3.9.14-2cc3edc74f38772096f1e8978a9716da/include/python3.9 -I/cvmfs/cms.cern.ch/e19_amd64_gcc12/external/pytorch/2.6.0-b283a5816a48ce89ff595247dda5c1b7/include/THC -I/cvmfs/cms.cern.ch/e19_amd64_gcc12/external/pytorch/2.6.0-b283a5816a48ce89ff595247dda5c1b7/include/torch/csrc/api/include -I/cvmfs/cms.cern.ch/e19_amd64_gcc12/external/pytorch/2.6.0-b283a5816a48ce89ff595247dda5c1b7/include/TH -fPIC -O1 -g3 -fno-trapping-math -funsafe-math-optimizations -ffinite-math-only -fno-signed-zeros -fno-math-errno -fexcess-precision=fast -fno-finite-math-only -fno-unsafe-math-optimizations -ffp-contract=off -fno-tree-loop-vectorize -march=native -Wall -std=c++20 -Wno-unused-variable -Wno-unknown-pragmas -fopenmp -c -o model.o
```

Load it in the inference code

```
1 #include <iostream>
2 #include <vector>
3
4 #include <torch/torch.h>
5 #include <torch/csrc/inductor/aoti_package/model_package_loader.h>
6
7 int main() {
8     c10::InferenceMode mode;
9
10    torch::inductor::AOTIModelPackageLoader loader("model_cpp_only_export_v1.pt2");
11    torch::inductor::AOTIModelContainerRunner* runner = loader.get_runner();
12    // Assume running on CUDA
13    std::vector<torch::Tensor> inputs = {torch::randn({8, 10}, at::kCPU)};
14    std::vector<torch::Tensor> outputs = runner->run(inputs);
15    std::cout << "Result from the first inference:" << std::endl;
16    std::cout << outputs[0] << std::endl;
17
18    // The second inference uses a different batch size and it works because we
19    // specified that dimension as dynamic when compiling model.pt2.
20    std::cout << "Result from the second inference:" << std::endl;
21    // Assume running on CUDA
22    std::cout << runner->run({torch::randn({1, 10}, at::kCPU)})[0] << std::endl;
23
24    return 0;
25 }
```

Plan for tomorrow

- Understand how to streamline AOT model compilation and inference with cmsdist to do every necessary steps automatically
- Test more complicated models
- Test CUDA support

Opens from yesterday:

- Investigate cudaMalloc operations during inference in cmssw.
 - In principle we are reusing SoA mem buffers, torch is doing internal allocations for intermediate stuff - I had already seen such behavior in ONNX runtime, need check if we can get rid of that completely
- Update our conversion scheme with latest changes in SoA in cmssw,
 - Simplify interface by using SoA metadata rather than relying on user understanding of interface.

Running event generators on GPU

Daniele Massaro, Davide Di Croce, Johannes Junggeburth, Mateusz Jakub Fila, Eric Cano, Axel Naumann, Attila Krasznahorkay, Simone Balducci



NextGen
Next Generation Triggers

What happened yesterday and today

- We studied how to profile MadGraph:
 - Recompile the code with new flags (debug options + frame pointers).
 - Understand how MadGraph works under the hood.
 - Generate flamegraphs with Linux perf.
- Maksymilian Graczyk from IT-FTI-PSE came to present Adaptyst profiler, discussing its features:
 - Use Adaptyst to profile MadGraph and produce flamegraphs.

Plan for tomorrow

- Further tests with Adaptyst and perf.
- Understand the impact of PDF evaluation, and study how we can improve the situation by using LHAPDF library.

C++ Coroutines

Mateusz Jakub Fila, Eric Cano, Axel
Naumann, Attila Krasznahorkay



NexTGen
Next Generation Triggers

What happened today

- Started adding some simple coroutine code to [Gaudi](#)
 - https://gitlab.cern.ch/akraszna/GaudiGaudi/-/tree/CoroutineAsyncAlgorithm-master-20250409?ref_type=heads
 - To be used for testing the overhead of using coroutines on “real jobs” that don’t actually use coroutines
- Investigated call stack of nested coroutines with ping-pong example

CALL STACK

```
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
std::__n4861::coroutine_handle<CoroutineTests::Player
CoroutineTests::Player::start(CoroutineTests::Player
main()
```

CALL STACK

```
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
pong(_Z4pongv.Frame *)(_Z4pongv.Frame * frame_ptr)
ping(_Z4pingv.Frame *)(_Z4pingv.Frame * frame_ptr)
std::__n4861::coroutine_handle<CoroutineTests::Player
CoroutineTests::Player::start(CoroutineTests::Player
main()
```

```
Pong 63964
Ping 63965
Pong 63965
Ping 63966
Pong 63966
Ping 63967
Pong 63967
Segmentation fault (core dumped)
```

Plan for tomorrow

- Continue to investigate mechanisms for nesting coroutines
- Explore executing coroutines on a thread-pool to achieve asynchronicity
- Benchmark examples:
 - Compare with thread-based solutions
 - Measure overheads of suspension and resume
- Write example utility functions:
 - Aggregating await to resume when all operations are done(`co_await when_all(task1, task2)`)
 - Waiting synchronously for async coroutine (`task1.run(thread_pool).wait()`)
- Run some actual tests with Gaudi