
# Inference of ML Models on Intel GPUs with SYCL and Intel OneAPI using SOFIE

*SFT Group*
*Supervisor: Lorenzo Moneta*

Ioanna-Maria Panagou

16/09/2022

# Introduction

*Prerequisites*

- ROOT is an open-source framework for high-scale **data processing** and **analysis** in HEP
- ROOT offers support for **supervised learning techniques** for data analysis through the **TMVA** (**T**oolkit of **M**ulti-**v**aried **A**nalysis) module
- **TMVA SOFIE** (**S**ystem for **O**ptimised **F**ast **I**nference code **E**mit) is a fast machine-learning inference engine

# Introduction

*How SOFIE works*

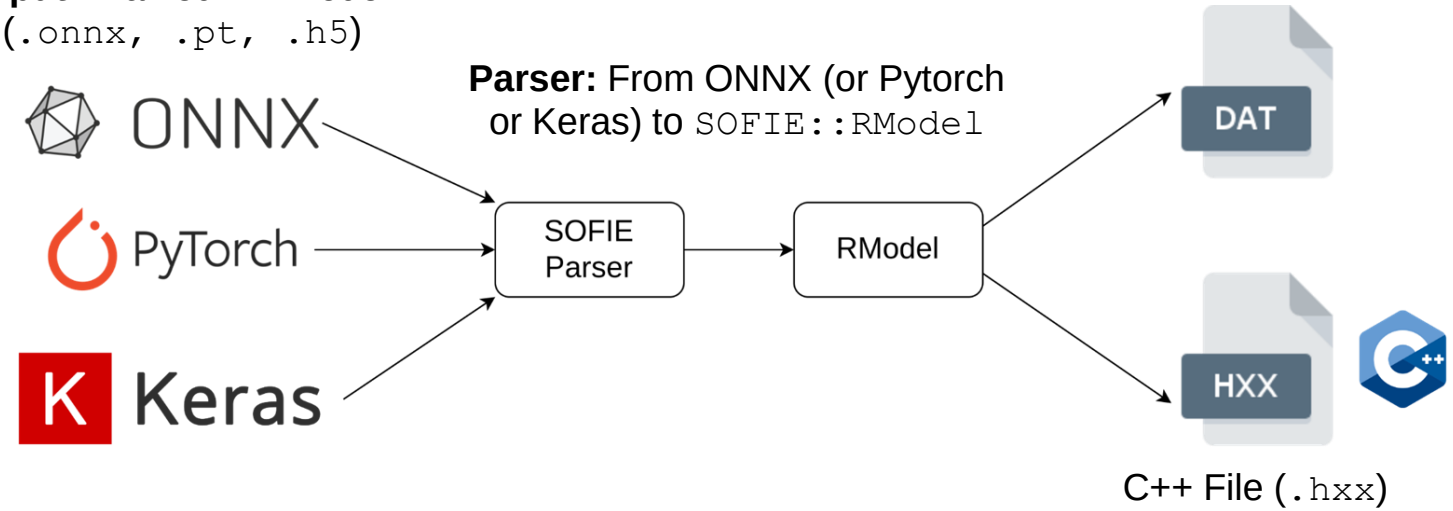
**Input:** Trained ML Model (`.onnx, .pt, .h5`)

**Parser:** From ONNX (or Pytorch or Keras) to `SOFIE::RModel`

**Outputs**

Weight File (`.dat`)

C++ File (`.hxx`)




- C++ file hardcodes the inference function
- Directly invokable from other C++ projects
- Minimal dependencies (on BLAS only)

# Introduction

## *How SOFIE works*

```
// Parser
using namespace TMVA::Experimental::SOFIE;
RModelParser_ONNX parser;
RModel model = parser.Parse("model.onnx");

// Generate Code Internally
model.Generate();

// Write Output Header File and Data Weight File
model.OutputGenerated();
```

model.hxx

```
namespace TMVA_SOFIE_model {

struct Session {

      Session(std::string = "") {
            . . .
      }
      // hardcoded inference function
      std::vector<float> infer(float *input) {
            . . .
      }
};
}
```

infer.cpp

```
#include "model.hxx"
// Create Session Class
TMVA_SOFIE_model::Session s();

// Event Loop
{
            auto result =
s.infer(input);
```

Inference code: needs to be linked against the BLAS Libraries

# Project Description

*Extend SOFIE functionality to produce SYCL code*

```
// Parser
using namespace TMVA::Experimental::SOFIE;
RModelParser_ONNX parser;
RModel model = parser.Parse("model.onnx");

// Generate SYCL Code Internally
model.GenerateGPU();

// Write Output Header File and Data Weight File
model.OutputGeneratedGPU();
```

model.hxx

```
namespace TMVA_SOFIE_model {

struct Session {

      Session(std::string = "") {
            . . .
      }
      // hardcoded inference function
      std::vector<float> infer(std::vector
      *input)           {
            . . .
      }
};
}
```

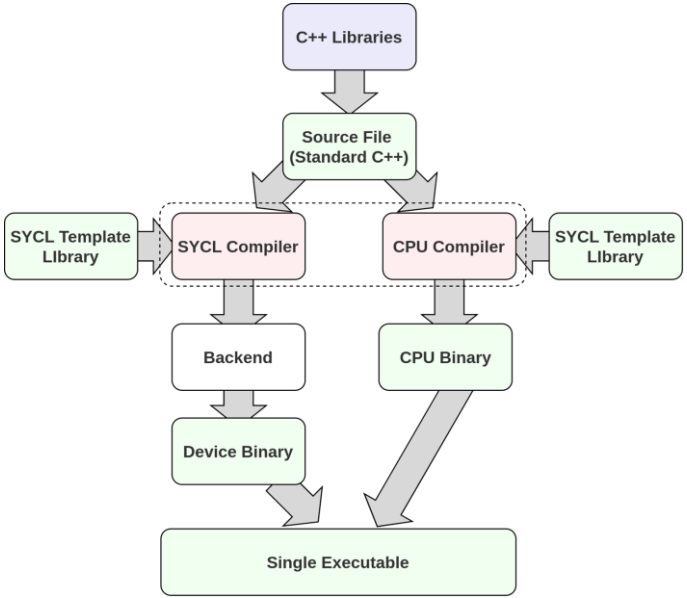
infer.cpp

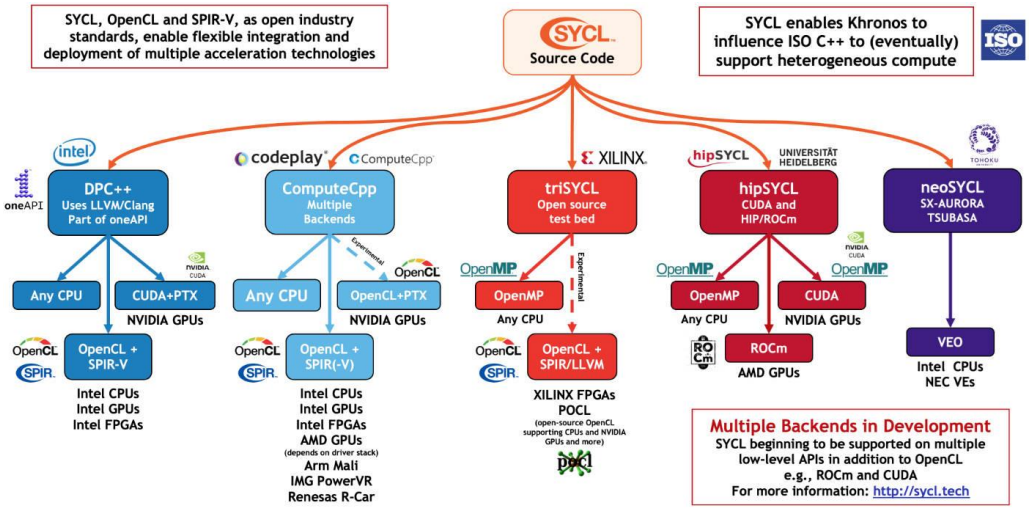
```
#include "model.hxx"
// Create Session Class
TMVA_SOFIE_model::Session s();

// Event Loop
{
            auto result =
s.infer(input);
```

Inference code: needs to be linked against the **oneAPI MKL Libraries (oneapi::mkl::blas)** and compiled using a **SYCL compiler**

# What is SYCL?

- SYCL is a single-source, high-level, standard C++ programming model, that can target a wide range of heterogeneous platforms (CPUs, GPUs, FPGAs)



# SYCL Application Structure

```
#include <CL/sycl.hpp> // header file
namespace sycl = cl::sycl;

int main() {
  // host storage (a = b * 2)
  std::vector<float> a = {1.0, 2.0, 3.0, 4.0};
  std::vector<float> b = {0.0, 0.0, 0.0, 0.0};
  auto length = a.size();
  sycl::default_selector device_selector; // device selector
  sycl::queue queue(device_selector); // queue

  { // begin scope
    // device storage
    sycl::buffer<float, 1> a_buf(a.data(), sycl::range<1>(length));
    sycl::buffer<float, 1> b_buf(b.data(), sycl::range<1>(length));

    // kernel execution
    queue.submit([&] (sycl::handler& cgh) {
      auto a_acc = a_buf.get_access<sycl::access::mode::discard_write>(cgh);
      auto b_acc = b_buf.get_access<sycl::access::mode::read>(cgh);
      cgh.parallel_for<class op>(sycl::range<1>(length), [=] (sycl::id<1> id) {
        a_acc[id] = b_acc[id] * 2;
      });
    });
  } // end scope
  return (0);
}
```

Include SYCL header

Setup Host Storage

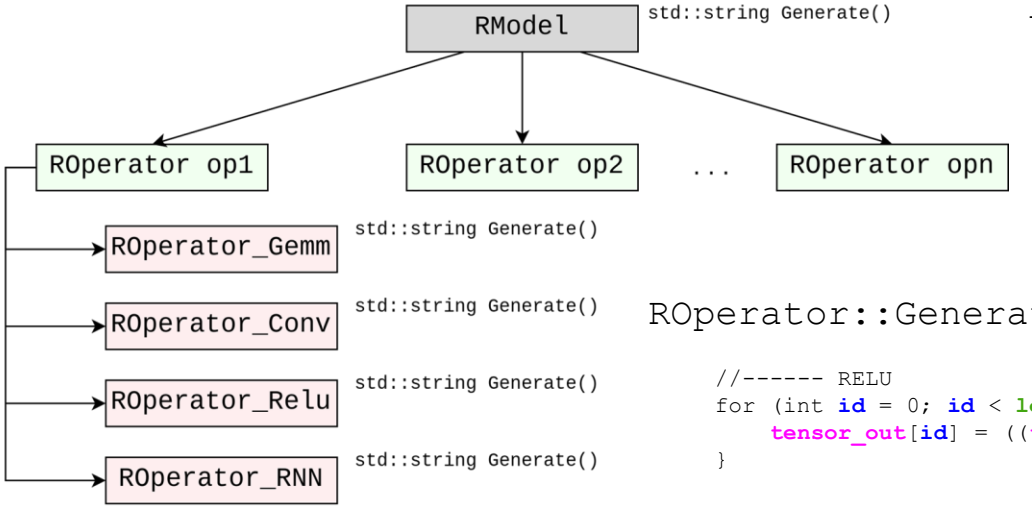
Initialize Device Selector

Initialize Device Queue

Setup Device Storage

Execute Kernel

# C++ to SYCL




`RModel::GenerateGPU` generates code for:

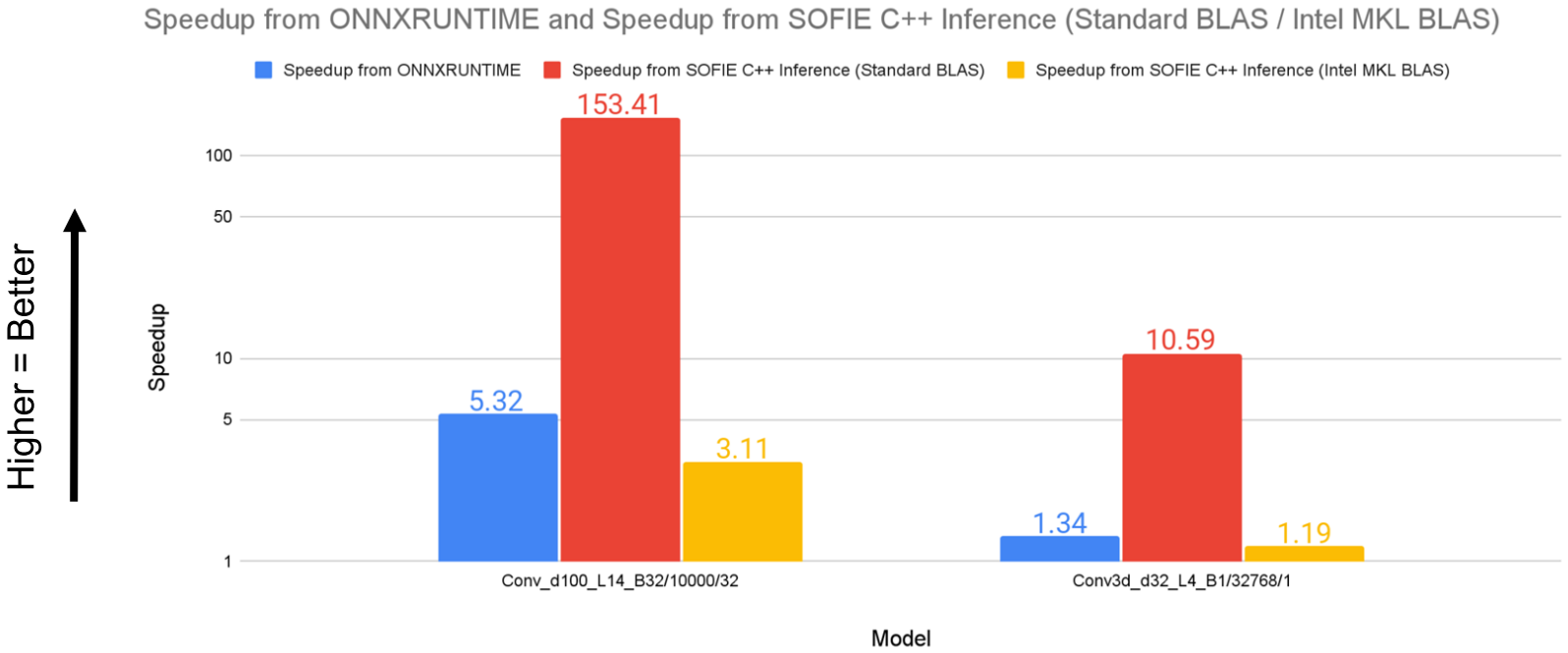
- Device Selection
- Queue Initialization
- Host & Device storage

`ROperator::GenerateGPU` generates code for the kernel execution

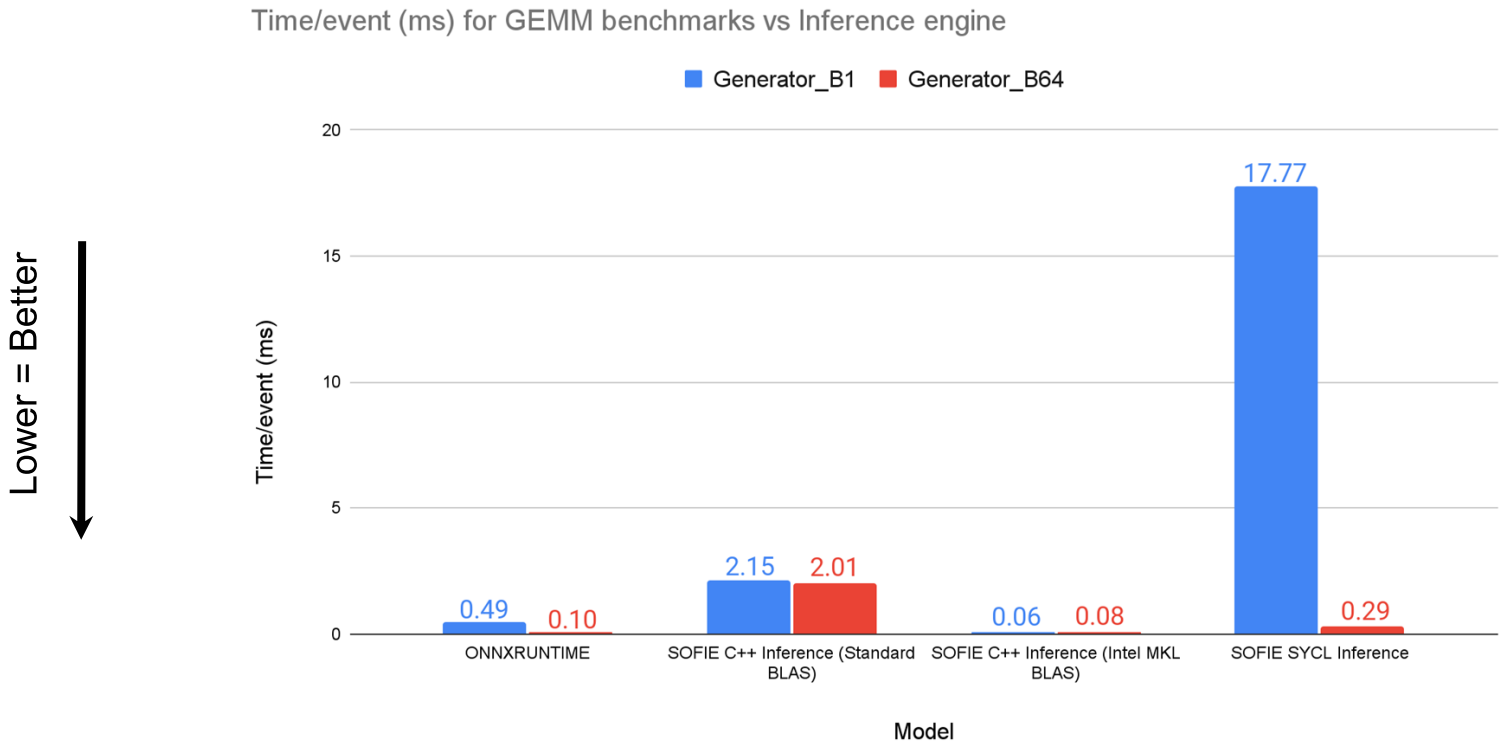
```
//------ RELU
for (int id = 0; id < length ; id++){
    tensor_out[id] = ((tensor_in[id] > 0 )? tensor_in[id] : 0);
}
```

```
//------ RELU
q.submit([&](cl::sycl::handler &cgh){
    auto acc_tensor_in = cl::sycl::accessor{buf_tensor_in, cgh, cl::sycl::read_only};
    auto acc_tensor_out = cl::sycl::accessor{buf_tensor_out, cgh, cl::sycl::write_only,
    cl::sycl::no_init};
    cgh.parallel_for<class op_17>(cl::sycl::range<1>(length), [=](cl::sycl::id<1> id){
        acc_tensor_out[id] = cl::sycl::max(acc_tensor_in[id], 0.0f);
    });
});
```

# Benchmarking



# Benchmarking



# Thank you!

*Questions?*