Inference Code Generation for Deep Learning models

Mentors: Lorenzo Moneta, Sanjiban Sengupta

GSOC lightning talk

- By Neel Shah

Reach out to me at:- *neelshah29042002@gmail.com*

ROOT

Data Analysis Framework

TMVA

ROOT:-

- ROOT is a framework for data processing, born at CERN, at the heart of the research on high-energy physics.
- Physicists use ROOT applications to analyze their data(save, access and mine data), publishing results, run interactively or build a new application or to perform simulations.

TMVA:-

- Toolkit for Multivariate Analysis
- Provides a Machine Learning environment for training, testing and evaluation of multivariate methods.

BORN OF FAST INFERENCE ENGINE!

Focus is put on a fast machine learning inference system, which will enable analysts to deploy their machine learning models rapidly on large scale datasets.

SOFIE

What does “SOFIE” stand for?

SOFIE

System for **O**ptimized **F**ast **I**nference code **E**mit

inference code, fast to operate, with least dependencies

Motivation

- ML ecosystem mostly focuses on model training.
- Machine Learning Inference & deployment is often neglected
- Inference in Tensorflow & PyTorch
  - supports only their own model
  - usage of C++ environment is difficult
  - heavy dependency

- Inference in ONNX (Open Neural Network Exchange)
  - can use ONNXRuntime by Microsoft
  - large dependency
  - difficult to integrate in HEP applications
    - control of libraries, threads
    - not optimized for single event evaluation

What is SOFIE?

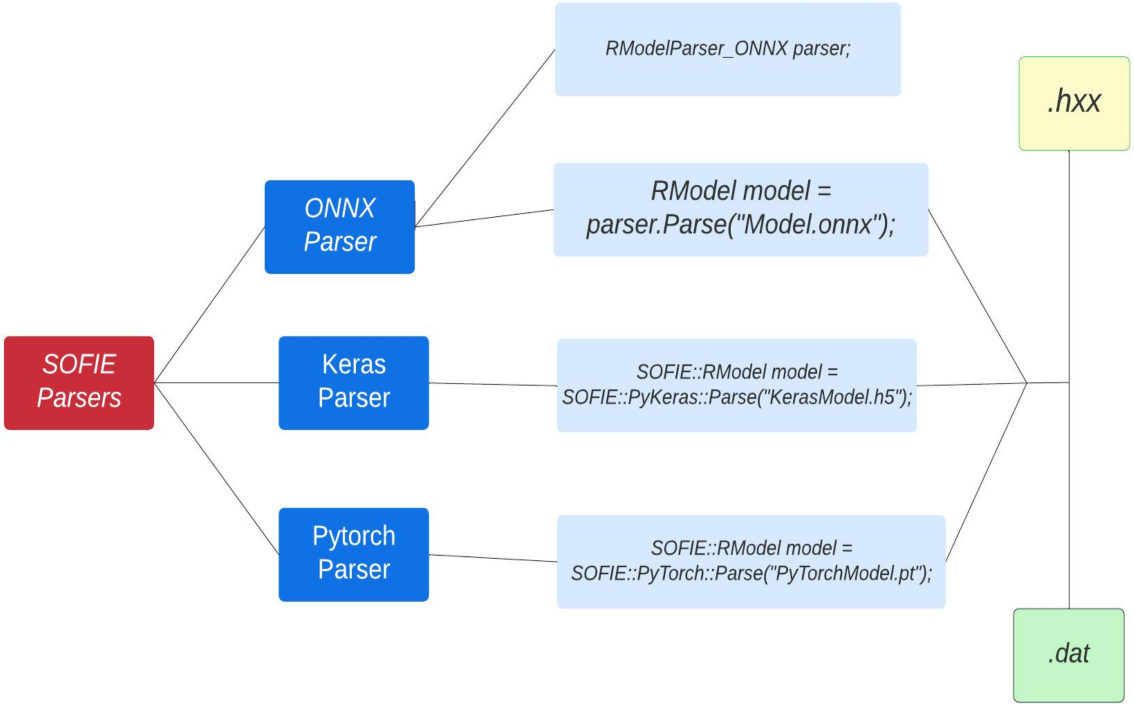
System for **O**ptimized **F**ast **I**nference code **E**mit

SOFIE(System for Optimized Fast Inference code Emit) is a deep learning inference engine that

- Takes ONNX files as input
- Produces a C++ script as output

TMVA SOFIE (“System for Optimized Fast Inference code Emit”) generates C++ functions easily invokable for the fast inference of trained neural network models. It takes ONNX model files as inputs and produces C++ header files that can be included and utilized in a “plug-and-go” style. This is a new development in TMVA and is currently in early experimental stage.

- Intermediate representation following ONNX standards.
- Inference code generation with least latency and minimal dependency



Description of Project

- This project will focus on development of some missing deep learning operations which will allow to build more complex networks within TMVA for parsing the **Transformer based models** and **Graph Net Models** in SOFIE.

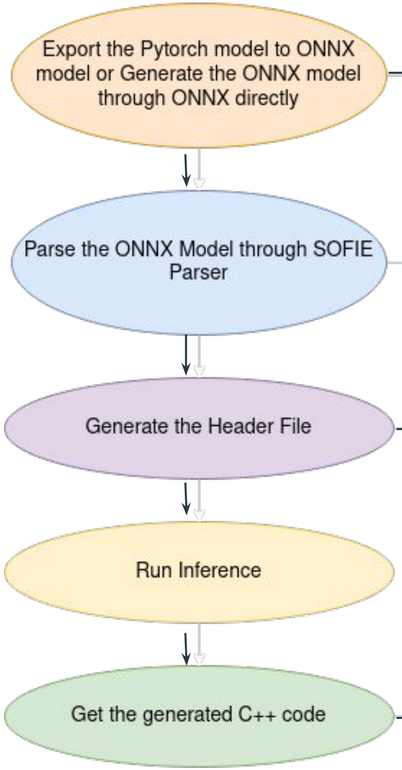
- The expected result is a working implementation of modular operators classes that implement the operators as defined by the ONNX standards in the code generation format. The project requires also to write the corresponding unit tests need to validate the written code.

Generated Code Dependencies

Generated code has minimal dependency

- only linear algebra library (BLAS)
- no dependency on ROOT libraries
- can be easily integrated in whatever software code

Code Generation in SOFIE




```python
torch.onnx.export(
        m,
        input,
        "LeakyRelu" + ".onnx",
        export_params=True
)
```

```
root [0] using namespace TMVA::Experimental;
root [1] SOFIE::RModelParser_ONNX parser;
root [2] SOFIE::RModel model = parser.Parse("./LeakyRelu.onnx");
root [3] model.Generate();
root [4] model.OutputGenerated("./LeakyRelu.hxx");
```

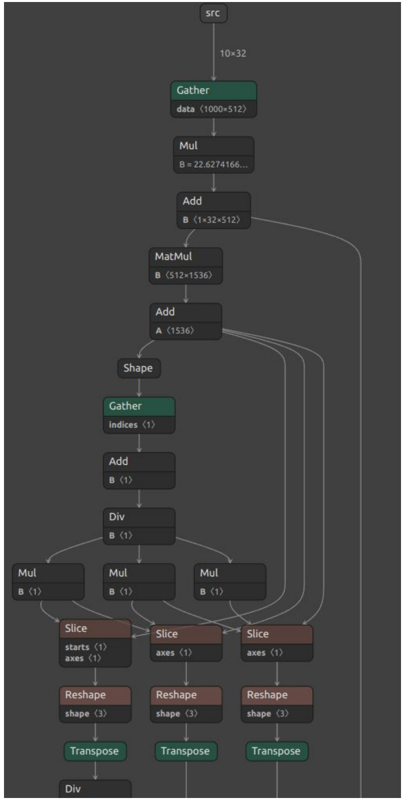
```cpp
//Code generated automatically by TMVA for Inference of Model file [LeakyRelu.onnx] at [Sat Sep 2

#ifndef TMVA_SOFIE_LEAKYRELU
#define TMVA_SOFIE_LEAKYRELU

#include<vector>
#include "TMVA/SOFIE_common.hxx"
#include <fstream>

namespace TMVA_SOFIE_LeakyRelu{
struct Session {
std::vector<float> fTensor_1 = std::vector<float>(24);
float * tensor_1 = fTensor_1.data();


Session(std::string filename ="") {
   if (filename.empty()) filename = "LeakyRelu.dat";
}

std::vector<float> infer(float* tensor_input){
   float op_0_alpha = 0.100000001;

//------ LEAKY RELU
   for (int id = 0; id < 24 ; id++){
      tensor_1[id] = ((tensor_input[id] > 0 )? tensor_input[id] : op_0_alpha * tensor_input[id]);
   }
   std::vector<float> ret (tensor_1, tensor_1 + 24);
   return ret;
}
};
} //TMVA_SOFIE_LeakyRelu

#endif  // TMVA_SOFIE_LEAKYRELU
```

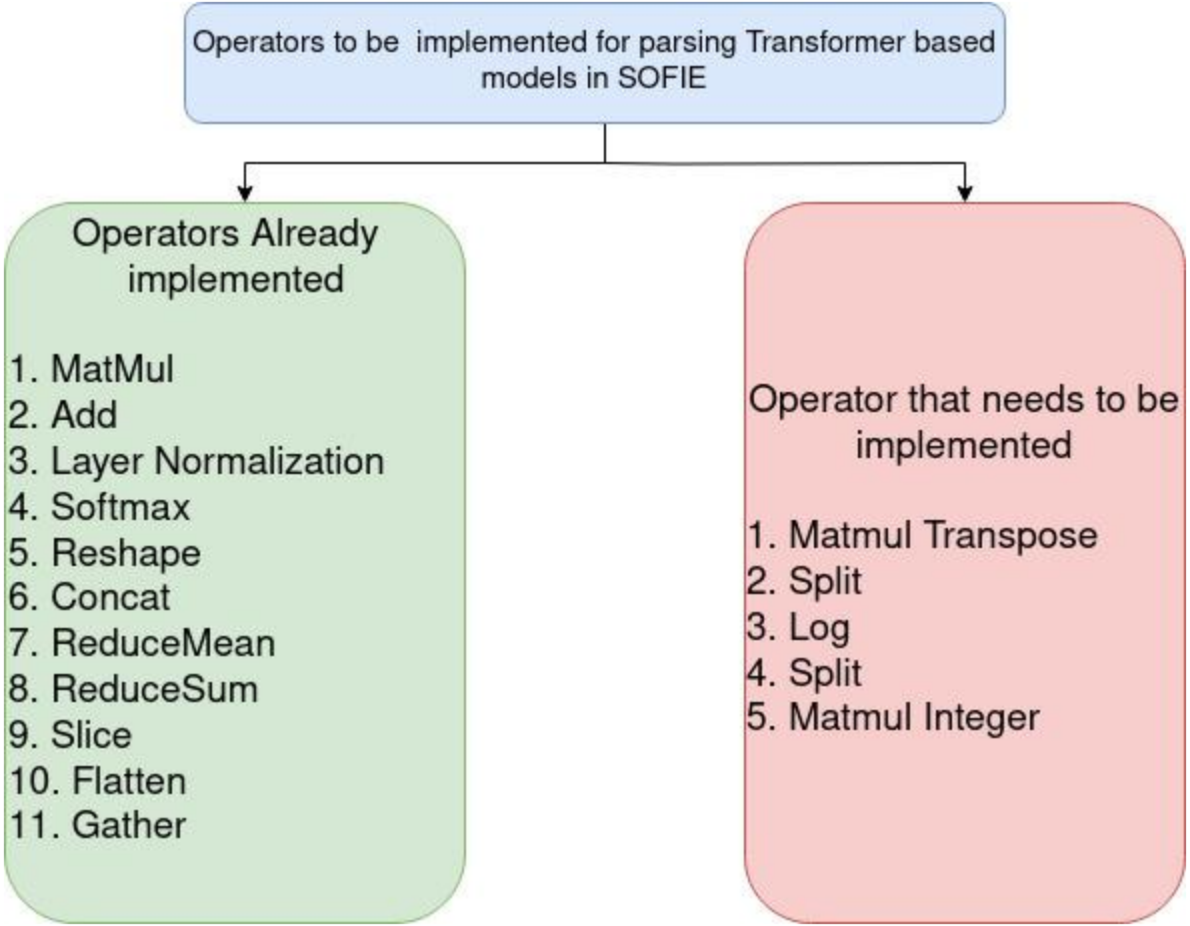
Entire output can be found [here]

Transformer Based models

```python
import torch
import torch.nn as nn
import torch.onnx as onnx
import math

class TransformerModel(nn.Module):
        def __init__(self, vocab_size, d_model, nhead, num_layers, dim_feedforward, dropout=0.1):
    super(TransformerModel, self).__init__()
    self.model_type = 'Transformer'
    self.embed = nn.Embedding(vocab_size, d_model)
    self.pos_enc = nn.Embedding(1000, d_model)
    self.transformer = nn.Transformer(d_model=d_model, nhead=nhead, num_encoder_layers=num_layers,
                                    num_decoder_layers=num_layers, dim_feedforward=dim_feedforward,
                                    dropout=dropout)
    self.fc = nn.Linear(d_model, vocab_size)
    self.init_weights()

def init_weights(self):
    initrange = 0.1
    self.embed.weight.data.uniform_(-initrange, initrange)
    self.fc.bias.data.zero_()
    self.fc.weight.data.uniform_(-initrange, initrange)

def forward(self, src, tgt):
    src = self.embed(src) * math.sqrt(self.embed.embedding_dim)
    tgt = self.embed(tgt) * math.sqrt(self.embed.embedding_dim)
    src_pos_enc = self.pos_enc(torch.arange(0, src.size(1), dtype=torch.long, device=src.device))
    tgt_pos_enc = self.pos_enc(torch.arange(0, tgt.size(1), dtype=torch.long, device=tgt.device))
    src = src + src_pos_enc.unsqueeze(0)
    tgt = tgt + tgt_pos_enc.unsqueeze(0)
    memory = self.transformer.encoder(src)
    output = self.transformer.decoder(tgt, memory)
    output = self.fc(output)
    return output

model = TransformerModel(vocab_size=1000, d_model=512, nhead=8, num_layers=6, dim_feedforward=2048, dropout=0.1)
src = torch.randint(0, 1000, (10, 32), dtype=torch.long)
tgt = torch.randint(0, 1000, (20, 32), dtype=torch.long)
output = model(src, tgt)

# Export the model to ONNX format
input_names = ["src", "tgt"]
output_names = ["output"]
onnx_path = "transformer_model.onnx"
torch.onnx.export(model, (src, tgt), onnx_path, input_names=input_names, output_names=output_names)
```

Generates



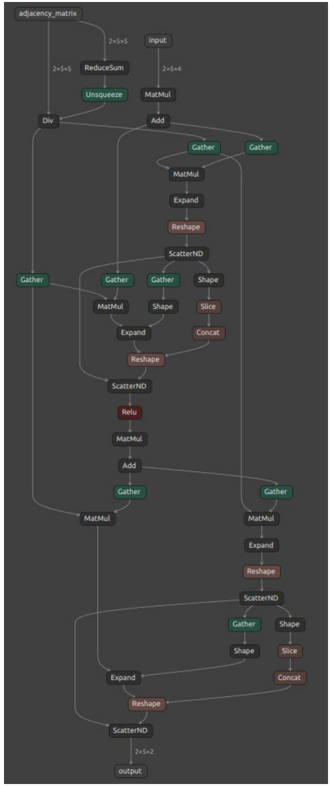

Operators to be implemented for parsing Transformer based models in SOFIE
Operators Already implemented
1. MatMul
2. Add
3. Layer Normalization
4. Softmax
5. Reshape
6. Concat
7. ReduceMean
8. ReduceSum
9. Slice
10. Flatten
11. Gather
Operator that needs to be implemented
1. Matmul Transpose
2. Split
3. Log
4. Split
5. Matmul Integer

Graph Net Based models

Entire output can be found [here]

```python
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
from torch_geometric.data import Data
from torch_geometric.utils import to_undirected
from torch.onnx import export

# Define the GNN model
class GNNModel(torch.nn.Module):
        def __init__(self, input_dim, hidden_dim, output_dim):
        super(GNNModel, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, output_dim)

        def forward(self, x, edge_index):
        edge_index = to_undirected(edge_index) # convert to undirected graph
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        return x

# Define input data
x = torch.randn(5, 3) # feature matrix with 5 nodes and 3 features per node
edge_index = torch.tensor([[0, 1, 1, 2, 3, 4], [1, 0, 2, 1, 4, 3]]) # edge index tensor

# Create PyTorch Geometric Data object
data = Data(x=x, edge_index=edge_index)

# Create GNN model instance
model = GNNModel(input_dim=3, hidden_dim=16, output_dim=2)
# Export model to ONNX format
input_names = ["input_x", "input_edge_index"]
output_names = ["output"]
torch.onnx.export(model, (data.x, data.edge_index), "gnn_model.onnx", input_names=input_names, output_names=output_names)
```

Generates



Operators to be implemented for parsing Graph Net based models in SOFIE

Operators Already implemented

1. Constant
2. Gather
3. ReduceMean
4. Squeeze
5. Unsqueeze
6. Concat
7. Reshape
8. Matmul
9. Add
10. Sub
11. Mul
12. Div
13. Exp
14. Transpose

Operator that needs to be implemented

1. Scatter
2. ScatterND
3. GatherND
4. Split
5. Log
6. Where
7. If
8. TopK
9. Greater
10. ConstantofShape
11. Range
12. Non-zero
13. Tile
14. Erf

Work done till now.....

Description	Pull Request
Added support for standalone MatMul operator to be accepted by Gemm Operator	Merged
Swish Activation function implemented in the Keras Parser	Merged
Implemented the Range ONNX Operator with unit tests	Under Review
Implemented the TopK ONNX Operator with unit tests	Under Review
Implemented the Log ONNX Operator with unit tests	Approved

Work done till now.....

Description	Pull request
Implemented the Erf ONNX Operator with unit tests	Approved
Implemented the Where ONNX Operator with unit tests	Under Review
Feature: Add an option of saving both .dat and .root files	Under Review
Implemented the Equal ONNX Operator with unit tests	Under Review
Implemented the ConstantOfShape ONNX Operator with unit tests	Under Review
Implemented the Elu ONNX operator	Under Review

Important Links:

1. Python Tutorials for various C files of Tutorials/TMVA
2. Documentation on RModelParser_ONNX.cxx
3. All about Community Bonding Period
4. Implementing the Operators in Sofie
5. GSOC 2022 Report
6. Final Project Presentation - GSOC 2022
7. GSOC 2023 Project Page

Thankyou for providing this opportunity!!

Any Questions?