Deploying ML algorithms in ATLAS with ONNX

Dhanush Hangal (he/him)

US ATLAS Machine Learning Training 2023

July 28, 2023

hangal1@llnl.gov

Why ONNX Runtime?

"ONNX Runtime is a cross-platform machine-learning model accelerator, with a flexible interface to integrate hardware-specific libraries. ONNX Runtime can be used with models from PyTorch, Tensorflow/Keras, TFLite, scikit-learn, and other frameworks."

onnxruntime.ai

Optimize Inferencing | **Optimize Training**

Platform	Windows	Linux	Mac	Android	iOS	Web Browser (Preview)	
API	Python	C++	C#	C	Java	JS	Obj-C	WinRT
Architecture	X64	X86	ARM64	ARM32	IBM Power		
Hardware Acceleration	Default CPU	CoreML	CUDA	DirectML	oneDNN		
	OpenVINO	TensorRT	NNAPI	ACL (Preview)	ArmNN (Preview)		
	MIGraphX (Preview)	Rockchip NPU (Preview)	SNPE	TVM (Preview)	Vitis AI (Preview)		
Installation Instructions	Please select a combination of resources						

ONNX Runtime for inferencing

ONNX Runtime Inference powers machine learning models in key Microsoft products and services across Office, Azure, Bing, as well as dozens of community projects.

Examples use cases for ONNX Runtime Inferencing include:

- Improve inference performance for a wide variety of ML models
- Run on different hardware and operating systems
- Train in Python but deploy into a C#/C++/Java app
- Train and perform inference with models created in different frameworks

How it works

- **Get a model.** This can be trained from any framework that supports export/conversion to ONNX format. See the tutorials for some of the popular frameworks/libraries.
- **Load and run the model with ONNX Runtime.** See the basic tutorials for running models in different languages.

ONNX Runtime basics in Python

Let's try some basic steps for ORT:

1. Train a simple model in Keras
2. Converting it into ONNXRunTime format and
3. Compare prediction results from the two models

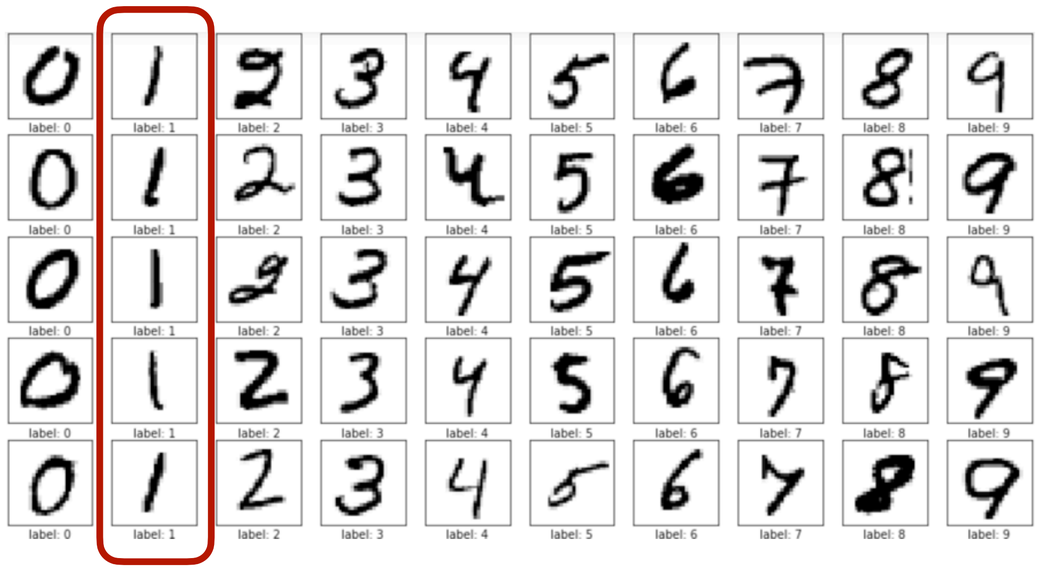
Using tutorial created by Aishik and Elham here

1. Generate data with some input features and a target.
2. Train a model on the data and convert it into onnx format using tf2onnx
3. Load ORT model for inference and compare MSE for predictions using Keras vs. ORT model

Workflow in Athena

1. Perform training externally and save the model, no support for training in Athena currently
2. Convert model to `.onnx` format using tf2onnx or onnxmltools
3. (Optional) Highly encourage checking the behavior of converted onnx model before using it in C++
4. Validation example : MNIST handwritten digit classification using Onnxruntime inference in Athena

Sample label = 1

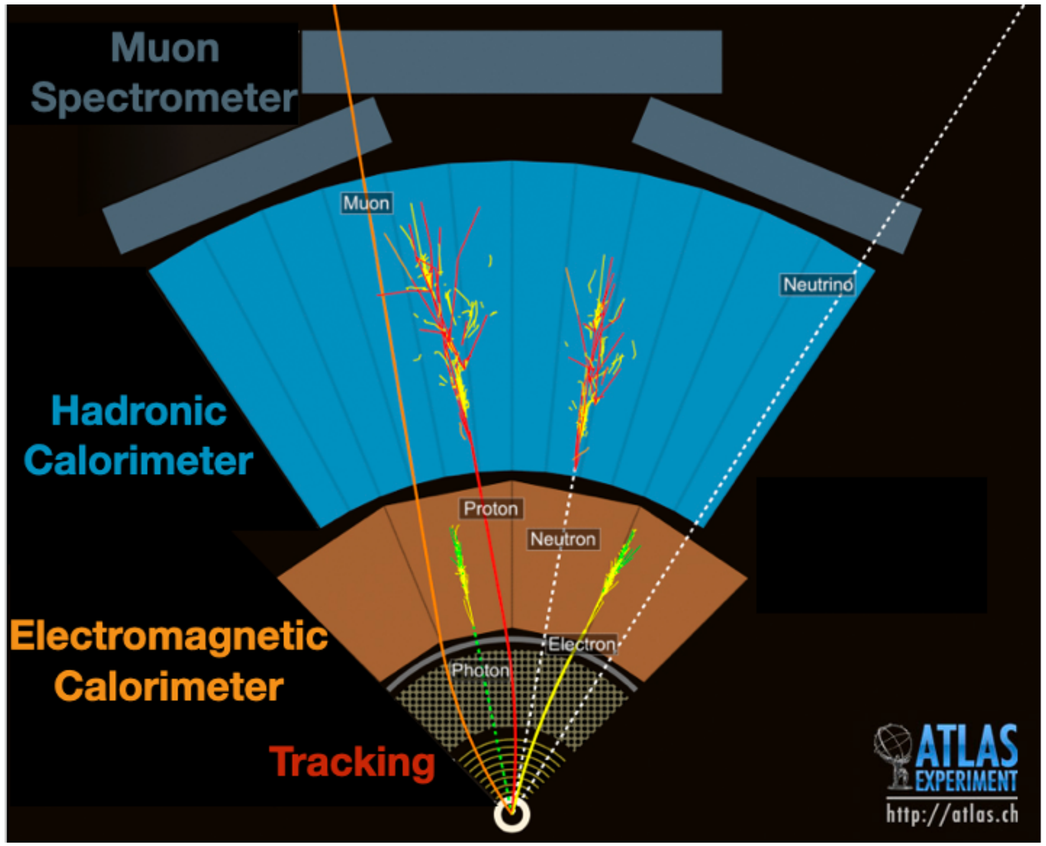


https://m-alcu.github.io/blog/2018/01/13/nmist-dataset/

Neural Network Magic

```
INFO Label for the input test data  = 1
AthONNX              DEBUG Score for class 0 = 1.07293e-07
AthONNX              DEBUG Score for class 1 = 0.999818
AthONNX              DEBUG Score for class 2 = 1.18024e-05
AthONNX              DEBUG Score for class 3 = 2.53529e-05
AthONNX              DEBUG Score for class 4 = 4.19157e-06
AthONNX              DEBUG Score for class 5 = 1.66088e-06
AthONNX              DEBUG Score for class 6 = 7.7723e-06
AthONNX              DEBUG Score for class 7 = 6.33801e-05
AthONNX              DEBUG Score for class 8 = 5.83467e-05
AthONNX              DEBUG Score for class 9 = 9.74693e-06
AthONNX               INFO Class: 1 has the highest score: 0.999818
```

Topo-cluster calibration



Target tasks :

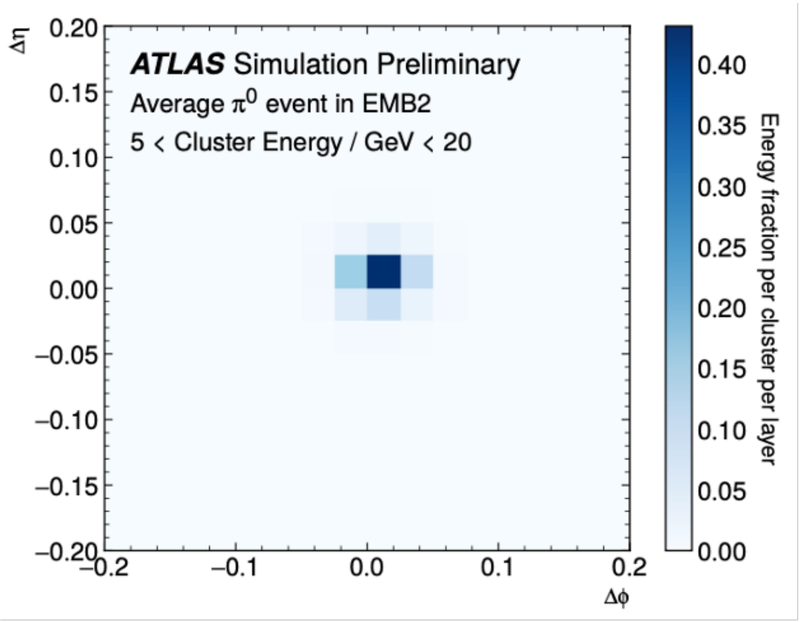
- **Classification** : Neutral vs. charged pions
- **Regression** : Predict pion energy

ATL-PHYS-PUB-2020-018

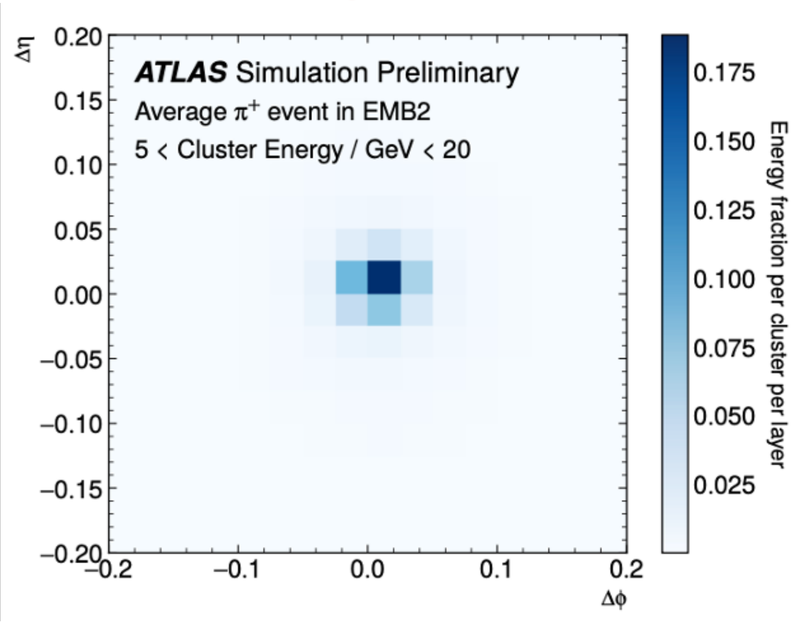
Topo-cluster calibration

- Represent cluster energy distribution in each layer of the calorimeter as images to perform pion classification
- Single-pion MC energy depositions reconstructed in the barrel calorimeter layers for neutral and charged pions
  - Difference between the average shower energy distributions for π^0 and π^+ shown below
  - EM showers of $\pi^0 \rightarrow \gamma\gamma$ are significantly narrower compared to the hadronic π^+ showers

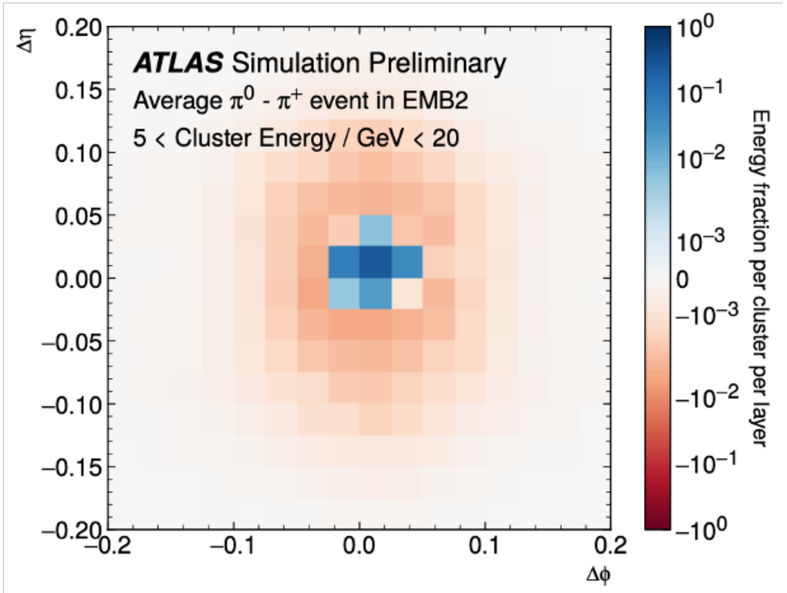
Neutral pions



Charged pions



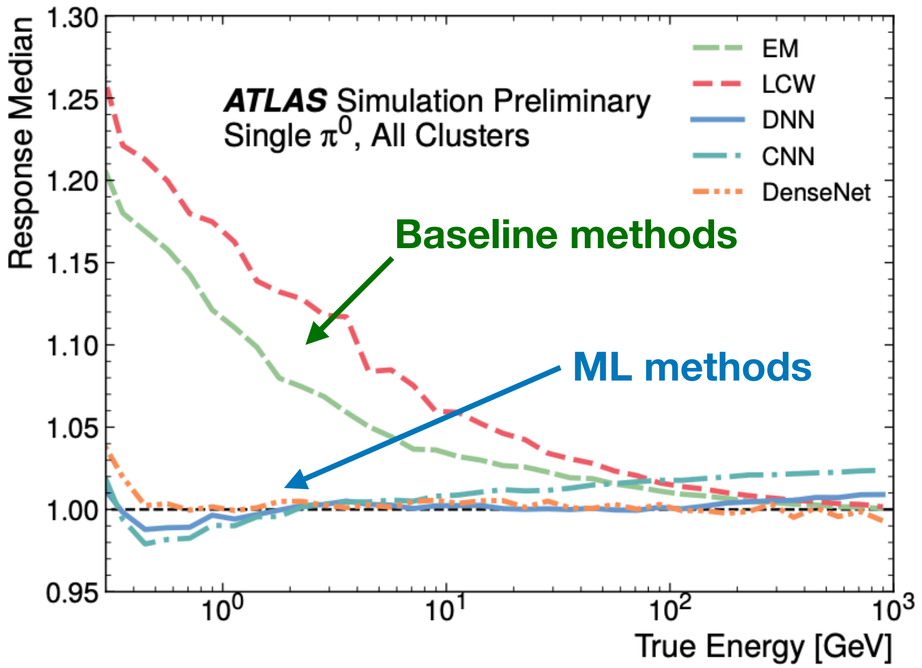
Difference

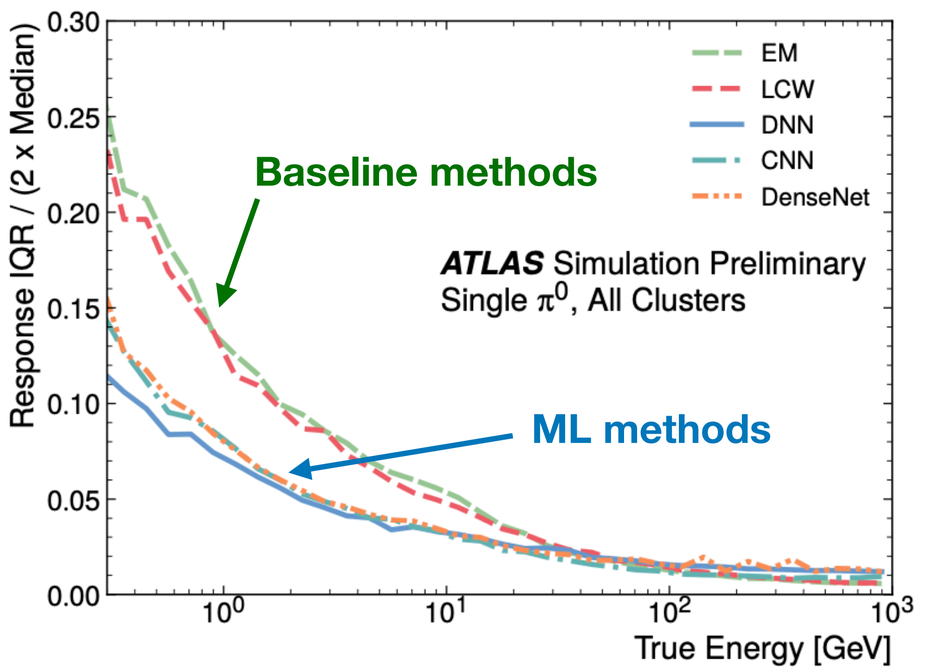


ATL-PHYS-PUB-2020-018

Topo-cluster calibration

- Cluster energy and η are provided as additional inputs in predicting the calibrated topocluster energy
- True energy is defined by the sum of all energy deposits within the topo-cluster, as defined in GEANT4
- The ML methods used all outperform the EM & LCW energy calibration baselines, particularly for pi0s
- Interquantile range (IQR) = a measure of the spread of the regression energy response around the median

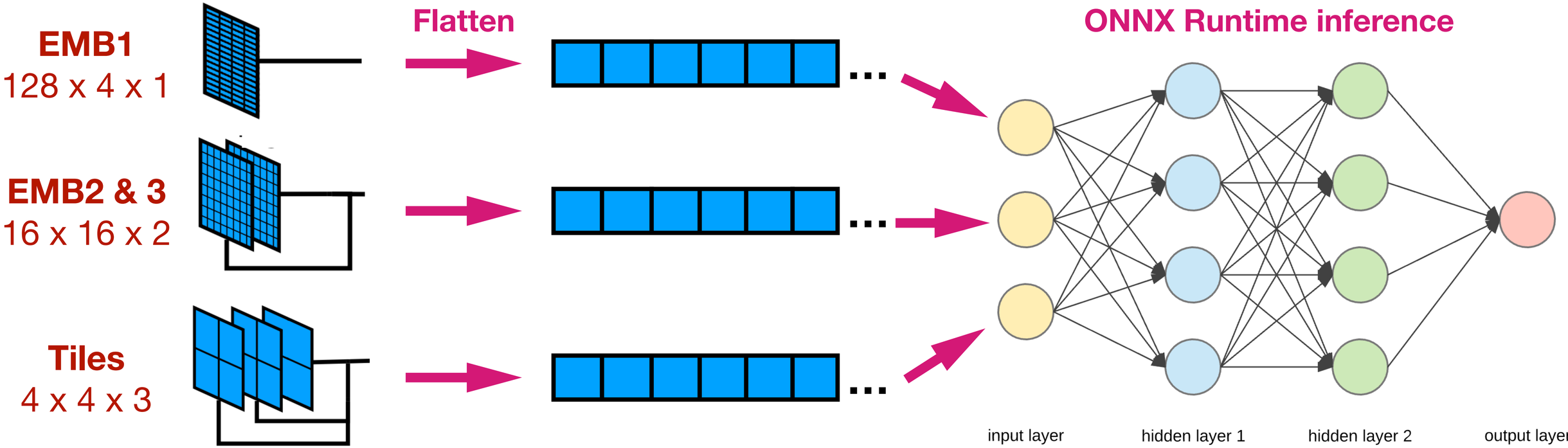




ATL-PHYS-PUB-2020-018

Topo-cluster calibration in Athena

- Topo-cluster calibration using Onnxruntime, based off of the MNIST digit classification example
- Stripped down to basics, loading in an preprocessed root file for the calorimeter images
- Also loading in a converted onnx model file for the topocluster energy regression inference
- Calorimeter images have to be **flattened** before being provided as input tensors in the inference



ATL-PHYS-PUB-2020-018

ONNX Runtime in Athena

github link

Topocluster calibration

Log out of lxplus and log back in

```
setupATLAS
asetup Athena, master, latest
mkdir build run
lsetup git
git atlas init-workdir https://gitlab.cern.ch/dhangal/athena.git
cd athena
git checkout onnx_ml
git atlas addpkg AthExOnnxRuntime
```

Change the onnx model file and input root file in the header file to match the ones shown at the end of this page.

```
cd ../build
cmake ../athena/Projects/WorkDir
make
source x86_64-centos7-gcc11-opt/setup.sh
cd ../run
cp ../athena/Control/AthenaExamples/AthExOnnxRuntime/share/AthExOnnxRuntime_jobOptions.py .
athena AthExOnnxRuntime_jobOptions.py
```

root file : /afs/cern.ch/work/d/dhangal/public/pi0_fully_processed_wFolds.root
onnx file : /afs/cern.ch/user/d/dhangal/public/model_fold0_opset10.onnx

ONNX Runtime in Athena

- Onnxruntime resides in **Athena as an external package here** (version 1.11.1) and can be imported using **AthOnnxruntimeService**

- To use in an **Athena** package, include the dependencies as illustrated below in the **CmakeLists.txt**

```
# Declare the package's name.
atlas_subdir( AthExOnnxRuntime )

# Component(s) in the package.
atlas_add_library( AthExOnnxRuntimeLib
   INTERFACE
   PUBLIC_HEADERS AthExOnnxRuntime
   INCLUDE_DIRS ${ONNXRUNTIME_INCLUDE_DIRS}
   LINK_LIBRARIES ${ONNXRUNTIME_LIBRARIES} GaudiKernel )

atlas_add_component( AthExOnnxRuntime
   src/*.h src/*.cxx src/components/*.cxx
   INCLUDE_DIRS ${ONNXRUNTIME_INCLUDE_DIRS}
   LINK_LIBRARIES ${ONNXRUNTIME_LIBRARIES} AthExOnnxRuntimeLib AthenaBaseComps GaudiKernel PathResolver AthOnnxruntimeServiceLib)
```

ONNX Runtime in Athena

github link

Topocluster calibration

Log out of lxplus and log back in

```
setupATLAS
asetup Athena, master, latest
mkdir build run
lsetup git
git atlas init-workdir https://gitlab.cern.ch/dhangal/athena.git
cd athena
git checkout onnx_ml
git atlas addpkg AthExOnnxRuntime
```

Change the onnx model file and input root file in the header file to match the ones shown at the end of this page.

```
cd ../build
cmake ../athena/Projects/WorkDir
make
source x86_64-centos7-gcc11-opt/setup.sh
cd ../run
cp ../athena/Control/AthenaExamples/AthExOnnxRuntime/share/AthExOnnxRuntime_jobOptions.py .
athena AthExOnnxRuntime_jobOptions.py
```

root file : /afs/cern.ch/work/d/dhangal/public/pi0_fully_processed_wFolds.root
onnx file : /afs/cern.ch/user/d/dhangal/public/model_fold0_opset10.onnx

EvaluateModel.h

```
// Local include(s).
#include "AthOnnxruntimeService/IONNXRuntimeSvc.h"

// Framework include(s).
#include "AthenaBaseComps/AthReentrantAlgorithm.h"
#include "GaudiKernel/ServiceHandle.h"

// ONNX Runtime include(s).
#include <core/session/onnxruntime_cxx_api.h>
```

- Header file derived from AthReentrantAlgorithm

Onnxruntime specific headers included here

Import ort from AthOnnxRuntimeService

```
/// Handle to @c AthONNX::IONNXRuntimeSvc
ServiceHandle< IONNXRuntimeSvc > m_svc{ this, "ONNXRuntimeSvc",
                                        "AthONNX::ONNXRuntimeSvc",
                                        "Name of the service to use" };

/// @}

/// The "session" of ONNX Runtime that we'll be using
std::unique_ptr< Ort::Session > m_session;
std::vector<std::vector<float>> m_input_tensor_values;
std::vector<int> m_output_tensor_values;
```

Session object to run the inference

EvaluateModel.cxx

```
// Set up the ONNX Runtime session.
Ort::SessionOptions sessionOptions;
sessionOptions.SetIntraOpNumThreads( 1 );
sessionOptions.SetGraphOptimizationLevel( ORT_ENABLE_BASIC );
Ort::AllocatorWithDefaultOptions allocator;
m_session = std::make_unique< Ort::Session >( m_svc->env(),
                                              modelFileName.c_str(),
                                              sessionOptions );
ATH_MSG_INFO( "Created the ONNX Runtime session" );
```

Sets the number of threads used to parallelize the execution within nodes

Ort::session wraps the .onnx model, any model related parameters can be accessed by enquiring m_session

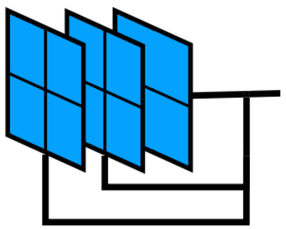
*read the input node dims for each layer and set dim[0]=1
(dim[0] refers to the batch size, 1 in this case)

```
// print input node types
Ort::TypeInfo type_info = m_session->GetInputTypeInfo(i);
auto tensor_info = type_info.GetTensorTypeAndShapeInfo();
ONNXTensorElementDataType type = tensor_info.GetElementType();
ATH_MSG_INFO("Input "<<i<<" : "<<" type= "<<type);

// print input shapes/dims
input_node_dims = tensor_info.GetShape();
ATH_MSG_INFO("Input "<<i<<" : num_dims= "<<input_node_dims.size());
for (std::size_t j = 0; j < input_node_dims.size(); j++){
    if(input_node_dims[j]<0) input_node_dims[j] =1;
    ATH_MSG_INFO("Input"<<i<<" : dim "<<j<<"= "<<input_node_dims[j]);
}
```

ONNX_TENSOR_ELEMENT_DATA_TYPE_FLOAT

Tiles : 4 x 4 x 3



```
INFO Input 0 :  name= input_3:0
INFO Input 0 :  type= 1
INFO Input 0 : num_dims= 4
INFO Input0 : dim 0= 1
INFO Input0 : dim 1= 4
INFO Input0 : dim 2= 4
INFO Input0 : dim 3= 3
```

EvaluateModel.cxx

```
input_tensor_values_tiles_ = read_tiles(Tiles);
input_tensor_values_EMB23_ = read_EMB23(EMB23);
input_tensor_values_EMB1_ = read_EMB1(EMB1_expand);
```

→ Flatten the input data

```
// create input tensor object from data values
// multiple inputs in this case requires the creation of an input tensor array
Ort::Value input_tensor[3]= {(Ort::Value::CreateTensor<float>(memory_info,
                                                              input_tensor_values_tiles_.data(),
                                                              input_tensor_size_tiles,
                                                              input_node_dims_tiles.data(),
                                                              input_node_dims_tiles.size())), …., ….}
```

Can use TensorCreator for this task

```
// score model & input tensor, get back output tensor
auto output_tensors = m_session->Run(Ort::RunOptions{nullptr},
                                     input_node_names.data(),
                                     input_tensor,
                                     input_node_names.size(),
                                     output_node_names.data(),
                                     output_node_names.size());
```

→ Running Inference through ORT

ONNX Runtime in Athena

github link

Topocluster calibration

Log out of lxplus and log back in

```
setupATLAS
asetup Athena, master, latest
mkdir build run
lsetup git
git atlas init-workdir https://gitlab.cern.ch/dhangal/athena.git
cd athena
git checkout onnx_ml
git atlas addpkg AthExOnnxRuntime
```

Change the onnx model file and input root file in the header file to match the ones shown at the end of this page.

```
cd ../build
cmake ../athena/Projects/WorkDir
make
source x86_64-centos7-gcc11-opt/setup.sh
cd ../run
cp ../athena/Control/AthenaExamples/AthExOnnxRuntime/share/AthExOnnxRuntime_jobOptions.py .
athena AthExOnnxRuntime_jobOptions.py
```

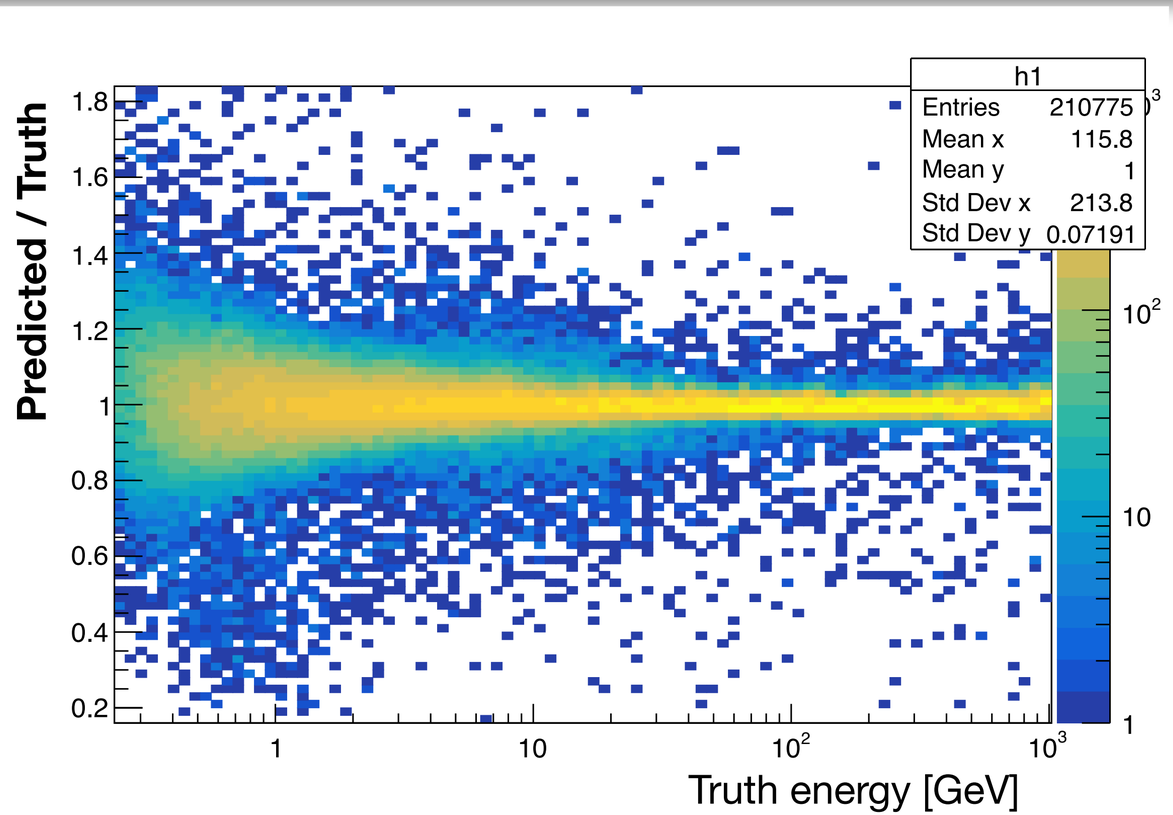
root file : /afs/cern.ch/work/d/dhangal/public/pi0_fully_processed_wFolds.root
onnx file : /afs/cern.ch/user/d/dhangal/public/model_fold0_opset10.onnx

Predicted topo-cluster energy

```
// Get pointer to output tensor float values
float* floatarr = output_tensors.front().GetTensorMutableData<float>();
```

Access first element of output tensor for inference result

```
h_calib_pred->Fill(cluster_ENG_CALIB_TOT, pow(10,floatarr[0])/cluster_ENG_CALIB_TOT);
```



Backup


ONNX
RUNTIME
GET STARTED | DOCS | NEWS | COMMUNITY | ABOUT | GITHUB


Adobe

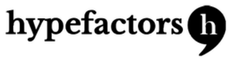

ANT GROUP


ATLAS
EXPERIMENT


CLEARBLADE
We Connect the Things.


GhostWriter


Hugging Face


hypefactors


intel


Microsoft
Visual Studio


navitaire
an AMADEUS company


NVIDIA


OPEN NLP


ORACLE


PEAKSPEED


PTW
THE DOSIMETRY COMPANY


Rockchip
瑞芯微电子


samtec


sas
THE POWER TO KNOW.


teradata.


TOPAZ
LABS


UNREAL
ENGINE


USDA


vespa


WRITER

Commonly used functions

- **SetIntraOpNumThreads** : Sets the number of threads used to parallelize the execution within nodes. When running a single node operation, ex. add, this sets the maximum number of threads to use
- **SetGraphOptimizationLevel** : These are semantics-preserving graph rewrites which remove redundant nodes and redundant computation
- **GetInputCount** : Returns the number of model inputs
- **ONNXTensorElementDataType** : Data type for input node
- **GetTensorMutableData** : Get a pointer to the raw data inside a tensor. Used to read/write/modify the internal tensor data directly

- Defined in OnnxUtils.h
- **FlattenInput_multiD_1D** : Flatten the data
- **TensorCreator** : Flattened input needs to be converted to ORT specific tensor
- **CreateORTSession** : A single instance of ORT session generated by IONNXRuntimeSvc
- **GetInputNodeInfo** : Returns input_node_dims and input_node_names from ORT session object
- **GetOutputNodeInfo** : Returns output_node_dims and output_node_names from ORT session object
- **Inference** : Takes input data and input, output information of the model and runs inside session where the model resides

ONNX Runtime on GPU

ORT GPU-build supports both CPU and GPU

With enabling following flags in CMakeLists.txt of ORT package *

```
option(onnxruntime_USE_ROCM "Build with AMD GPU support" OFF)
option(onnxruntime_CROSS_COMPILING "Cross compiling onnx runtime" OFF)
option(onnxruntime_USE_CUDA "Build with CUDA support" OFF)
```

*We do need CUDA installed in the machine

With above build we can use a single inference code with both CPU and GPU using a flag

```
    Ort::SessionOptions session_options;
#ifdef USE_CUDA
    Ort::ThrowOnError(OrtSessionOptionsAppendExecutionProvider_CUDA(session_options, 0));
#endif
    session_ = Ort::Session(env_, model_path_.c_str(), session_options);
```

Ort::Value input_tensor [n_inputs] = { Ort::Value::CreateTensor<float>(memory(CPU/GPU), Values of flattened 1D tile array, input tile array size (1*4*4*3), Model input dim. (1, 4, 4, 3), Model input dim. size (4)) , …. , …. }

ONNX Runtime in Athena

- Onnxruntime resides in Athena as an external package here (version 1.11.1) and can be imported using AthOnnxruntimeService

- How to use in **Athena**

  In packages' CmakeLists.txt

  - **External dependencies**
    - `find_package( onnxruntime )`
  - **atlas_add_library**
    - `INCLUDE_DIRS ${ONNXRUNTIME_INCLUDE_DIRS}`
    - `LINK_LIBRARIES ${ONNXRUNTIME_LIBRARIES}`
  - **atlas_add_component**
    - `INCLUDE_DIRS ${ONNXRUNTIME_INCLUDE_DIRS}`
    - `LINK_LIBRARIES ${ONNXRUNTIME_LIBRARIES}`