ONNX based inference in ATLAS

ILC software meeting
https://agenda.linearcollider.org/event/10344/

15th May 2024

Antonio Giannini

About me

A. Giannini, antonio.giannini@cern.ch

- PhD degree at Universita' Federico II, Naples (Italy)
- Post-doc at USTC, Hefei (China)
- Area of interest
  - Diboson physics (BSM searches and VBS measurements)
  - Boosted objects tagging, W/Z and Hbb/cc
  - ML applications, event classification, background estimations, jet tagging, Anomaly detection
- ATLAS responsibilities
  - HDBS Physics group ML liaison 2021 - 2023
  - Conveners of Diboson searches sub-group (HDBS-DBL), 2022 - 2024

HDBS = Higgs and DiBoson Searches
DBL = DiBoson Lab

Outline

ONNX based inference in ATLAS

- Introduction
  - typical workflow
  - why inference?
- ML inference in ATLAS
  - historical evolution, from lwtnn to ONNX
- Applications overview
  - (lwtnn) for analysis level
  - ONNX for jet tagging

[*ATHENA software (public) documentation*]

Wrap-up

Let me remember to cite the proper reference for discussing the ATLAS software!
ATL-SOFT-PUB-2021-001.pdf

ML models vs analyses

- ***ML applications have had a huge boost in Physics analyses at collider experiments***
    - during my ATLAS experience, ~O(1) in 2017 up to ~O(10) in 2024 of ML applications in analyses
    - searches better use cases for applications/development than measurements
- ***Wide variety of python based frameworks are used now***
    - TensorFlow, Keras, PyTorch, sci-kit learn, etc etc
    - relatively easy to get started, many tutorials around, many people willing to pass the knowledge
- This implies
    - a large population of “custom” frameworks doing mostly the same things
    - a large population of datasets, intermediate/output formats, ROOT, pandas, h5, etc
- ***All of these ingredients can make really hard the reproducibility of results!!!***

Typical workflow: entry level

Central production

MC production

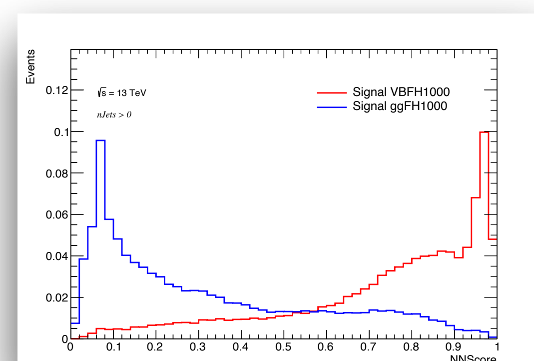
Derivation Framework

Analysis Framework

ROOT ntuples as input for the statistical interpretation

Python, pandas dataframes, Keras, …

Model: architecture, weights and scaling parameters



NNHelperTool for the ATLAS run-2 high-mass VV semi-leptonic search [Eur. Phys. J. C 80 (2020) 1165] used as an interface to use the [lwtnn] tool by ML conveners. Tested already in b-Tagging and Tau-Identification in ATLAS reconstruction.

Typical workflow: next level

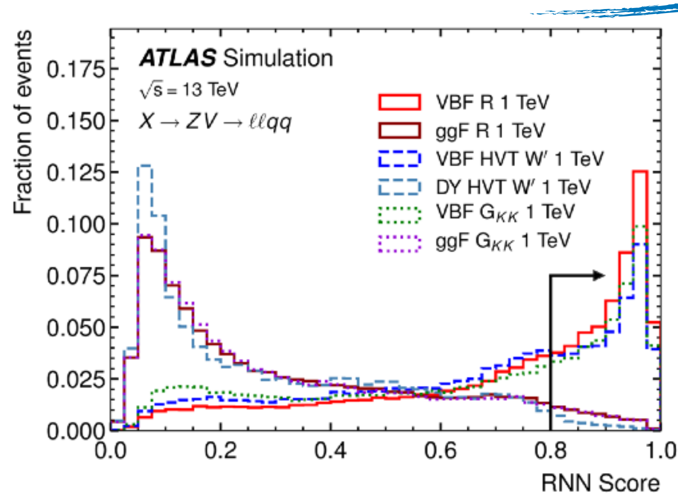
MC production

Central production

Derivation Framework, with dedicated information

"Analysis" Framework

Python, pandas dataframes, Keras, …



Model: architecture, weights and scaling parameters

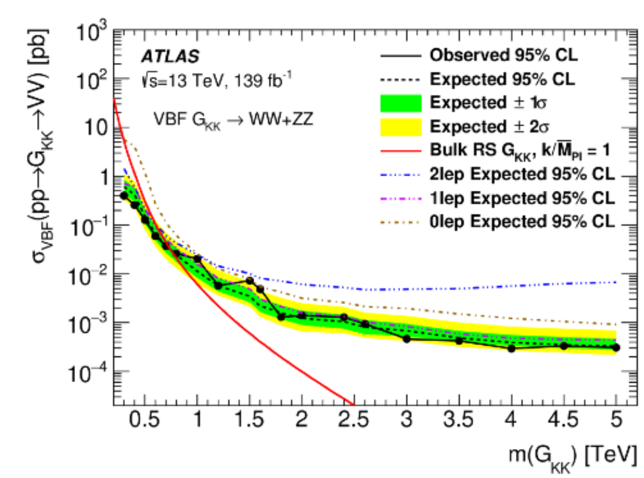
MC production

Central production

Derivation Framework with light information

Analysis Framework

ROOT ntuples as input for the statistical interpretation

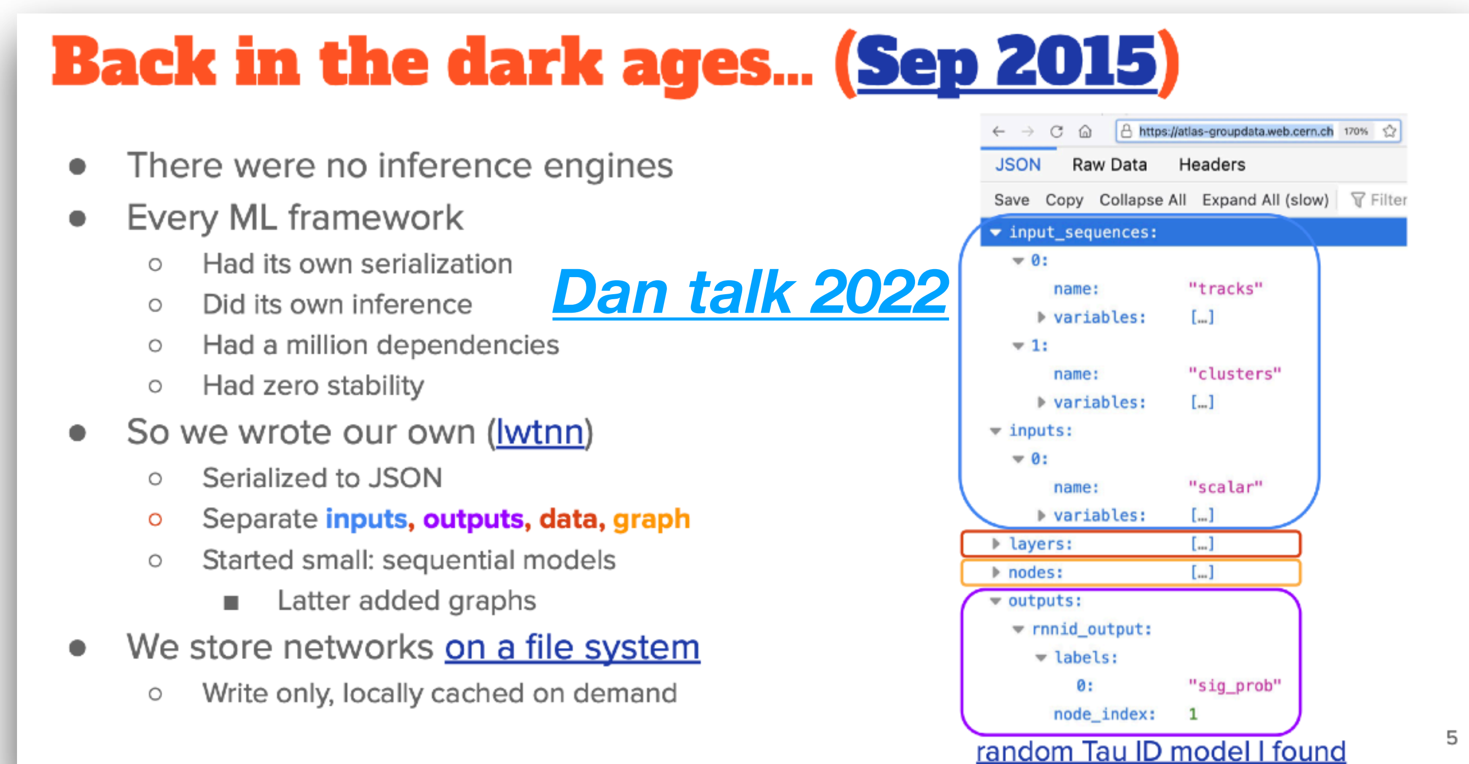


Why ML models inference in C++

- ML models inference in C++ is crucial for two main aspects
- ***Reproducibility of results outside the collaboration***
  - event classifier (NN>x), final discriminant (to complete the analysis ingredients)
  - anomaly detections (when the ML is the main analysis ingredient…)
- ***Deployment of a tool that can be used for several analyses***
  - jet tagging, low-level variables based classifiers, etc
- ***Inference is not a plus, nor an aesthetic element, it is a significant part of the R&D of a ML tool for physics!***

ML inference in ATLAS

- Two main tools have been used so far
  - *lwtnn: many analyses applications in the latest few years*
  - *ONNX: appeared in the last 3-4 years, so far, mainly jet tagging approaches (b-tag, Hbb/cc, W/Z) have been implemented*
- An historical perspective
  - my memories about this start around 2016/17 or so, at the time we had lwtnn tool available and it saved my PhD thesis!
  - found a funny definition from Dan Guest (former ATLAS ML convener, lwtnn author) ***"Dark ages"*** before lwtnn was made



- In 2018/19 when I implemented the inference for my first analysis only few (lwtnn) use cases were available:
  - tau RNN-ID, b-tagging, 1 SUSY multi jets analysis

lwtnn implementation

- lwtnn has been widely used in ATLAS
  - quite user friendly, you can get familiar with it in an afternoon and wrap it in your favourite tool/analysis code
- It is available in ATLAS software but you can also install it on your environment
- Model conversion support, very human readable!

Supported Layers *lwtnn github*

In particular, the following layers are supported as implemented in the Keras sequential and functional models:

	K sequential	K functional
Dense	yes	yes
Normalization	See Note 1	See Note 1
Maxout	yes	yes
Highway	yes	yes
LSTM	yes	yes
GRU	yes	yes
Embedding	sorta	issue
Concatenate	no	yes
TimeDistributed	no	yes
Sum	no	yes


```json
{   "class_name": "Model",
    "config": { "name": "model_1",
                "layers": [{"name": "input_1",
                            "class_name": "InputLayer",
                            "config": { "batch_input_shape": [null, 2, 4],
                                        "dtype": "float32",
                                        "sparse": false,
                                        "name": "input_1"
                                    },
                            "inbound_nodes": []
                            },
                            {"name": "jet_masking",
                            "class_name": "Masking",
                            "config": { "name": "jet_masking",
                                        "trainable": true,
                                        "mask_value": -99.0},
                                        "inbound_nodes": [[["input_1", 0, 0, {}]]]},
                                        {   "name": "jet_lstm1",
                                            "class_name": "LSTM",
                                            "config": { "name": "jet_lstm1",
                                                        "trainable": true,
                                                        "batch_input_shape": [null, null, 4],
                                                        "dtype": "float32",
                                                        "return_sequences": true,
                                                        "return_state": false,
                                                        "go_backwards": false,
                                                        "stateful": false,
                                                        "unroll": false,
                                                        "units": 25,
                                                        "activation": "tanh",
                                                        "recurrent_activation": "hard_sigmoid",
                                                        "use_bias": true,
```


paper HEPData

VBF-RNN architecture .json

ONNX in a nutshell

- ONNX runtime (https://github.com/onnx)
  - more an industry based usage
  - based on ***Open Neural Network eXchange*** files
  - supports many more operators than lwtnn (factor 10 maybe)
  - but not universal support
  - it is a bit less user friendly than lwtnn
- Installation made in the ATLAS software (ATHENA), ONNX is available as an external dependency
  - ***External/onnxruntime***
  - for ATLAS user: no need to install anything, just use it!!!

ONNX

ONNX 1.17.0 documentation

Search

Introduction to ONNX
API Reference
ONNX Operators
Sample operator test code
Abs
Acos
Acosh
Add
AffineGrid
And
ArgMax
ArgMin
Asin
Asinh
Atan
Atanh
AveragePool
BatchNormalization
Bernoulli
BitShift
BitwiseAnd
BitwiseNot
BitwiseOr
BitwiseXor
BlackmanWindow
Cast
CastLike
Ceil
Celu
CenterCropPad
Clip
Col2Im
Compress
Concat

ONNX Operators

Lists out all the ONNX operators. For each operator, lists out the usage guide, parameters, examples, and line-by-line version history. This section also includes tables detailing each operator with its versions, as done in Operators.md.

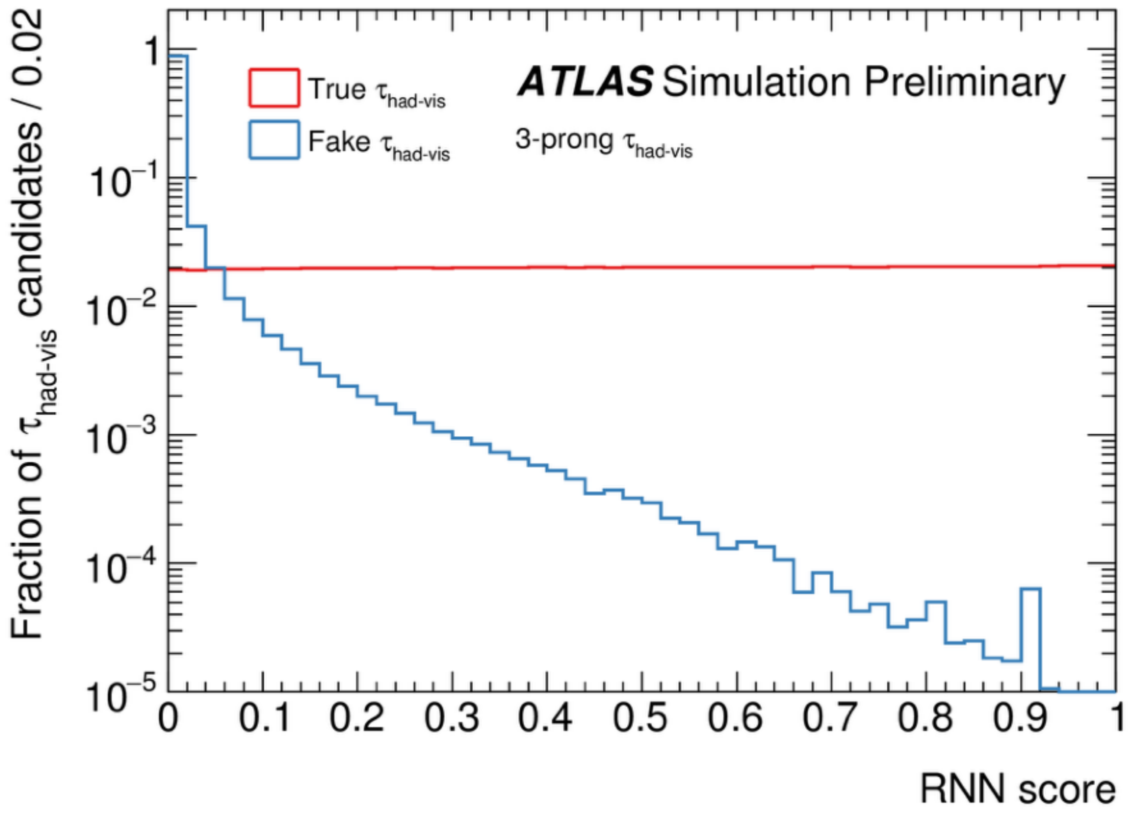
All examples end by calling function *expect*. which checks a runtime produces the expected output for this example. One implementation based on onnxruntime can be found at Sample operator test code.

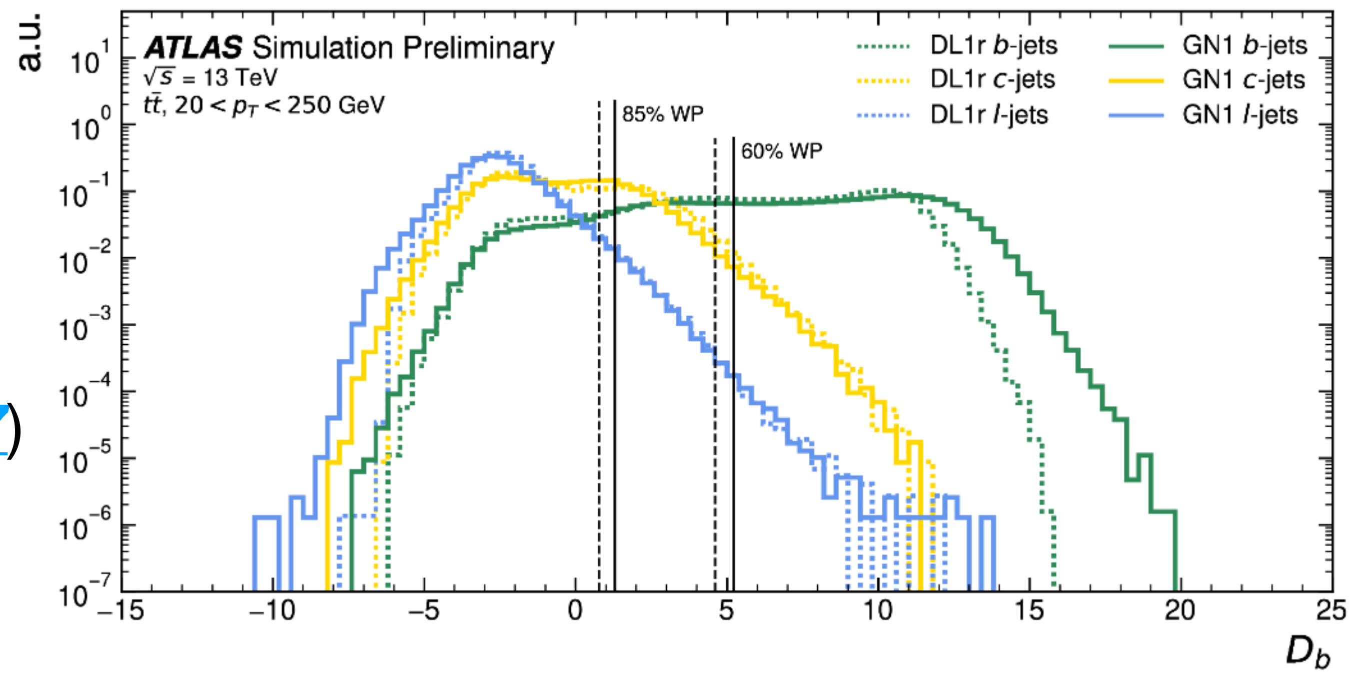
ai.onnx | ai.onnx.ml | ai.onnx.preview.training

operator	versions	differences
Abs	13, 6, 1	13/6, 13/1, 6/1
Acos	22, 7	22/7
Acosh	22, 9	22/9
Add	14, 13, 7, 6, 1	14/13, 14/7, 13/7, 14/6, 13/6, 7/6, 14/1, 13/1, 7/1, 6/1
AffineGrid	20	
And	7, 1	7/1
ArgMax	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
ArgMin	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
Asin	22, 7	22/7
Asinh	22, 9	22/9
Atan	22, 7	22/7
Atanh	22, 9	22/9
AveragePool	22, 19, 11, 10, 7, 1	22/19, 22/11, 19/11, 22/10, 19/10, 11/10, 22/7, 19/7, 11/7, 10/7, 22/1, 19/1, 11/1, 10/1, 7/1
BatchNormalization	15, 14, 9, 7, 6, 1	15/14, 15/9, 14/9, 15/7, 14/7, 9/7, 15/6, 14/6, 9/6, 7/6, 15/1, 14/1, 9/1, 7/1, 6/1
Bernoulli	22, 15	22/15
BitShift	11	
BitwiseAnd	18	
BitwiseNot	18	
BitwiseOr	18	
BitwiseXor	18	
BlackmanWindow	17	

Overview of ATLAS inference use cases

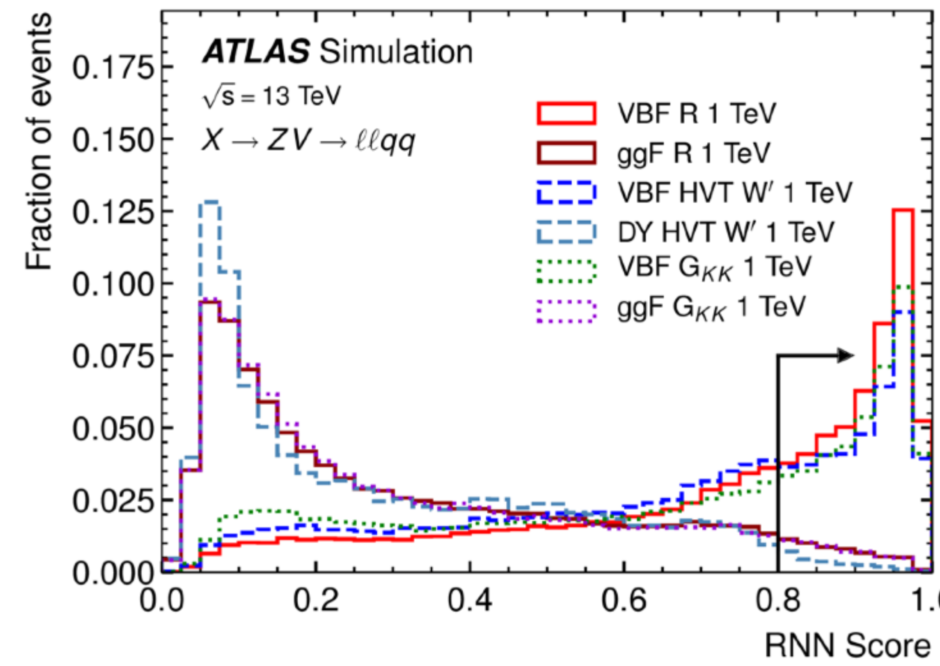
- lwtnn
  - tau RNN-ID ([ATL-PHYS-PUB-2019-033]())
  - b-tagging RNN/DeepSets ([ATL-PHYS-PUB-2020-014]())
  - VBF-RNN tagger in VV semi-leptonic ([Eur. Phys. J. C 80 (2020) 1165]())
  - Anomaly detection search (YXh) in X(qq)h(bb) events (no routine but available in HEPData) ([Phys. Rev. D 108 (2023) 052009]())
- ONNX
  - Multi b-jets + MET SUSY search ([Eur. Phys. J. C 83 (2023) 561]())
  - b-tagging ([ATL-PHYS-PUB-2022-027]())
  - Hbb/cc boosted tagger ([ATL-PHYS-PUB-2023-021]())





Analysis level application example (lwtnn)

- Run model conversion (keras —> .json)
- Features scaling config file supported
- Simple tool to interface the physics input preparation and load the ML model



neural_net.json 80.54 KiB


```
{
  "defaults": {},
  "inputs": [
    {
      "name": "NJets",
      "offset": -3.393768,
      "scale": 0.660834
    },
    {
      "name": "Zcand_m",
      "offset": -90.953651,
      "scale": 0.210485
    },
    {
      "name": "Zcand_pt",
      "offset": -626.260071,
      "scale": 0.00172
    },
    {
```


Features scaling

(Physics) inputs preparation

Retrieve the ML model

[CxAODFramework/Run2PaperVersion//NNHelper.cxx]()

```
lwt::VectorMap NNHelper::get_JetVars(const std::vector<lwt::Input>& input
  lwt::VectorMap out, out_mask;
  // ramp through the input multiplier
  const size_t total_inputs = inputs.size(); // nVariables = [pT, eta, ph
  for (size_t nnn = 0; nnn < total_inputs; nnn++) {
    const auto& input = inputs.at(nnn);
    out[input.name] = {};
    out_mask[input.name] = {};

    double mean = 0., std = -1.;

    if( input.name=="variable_0" ){
      if(UseMax6Jets) out.at(input.name) = { Jet1_pt , Jet2_pt , Jet3_pt
      if(UseMax2Jets) out.at(input.name) = { Jet1_pt , Jet2_pt };
      mean = mean_pT;
      std  = std_pT;
    }
```

```
void NNHelper::LoadRNN(){

  if(IsDebug) std::cout << " --- LoadRNN ---" << std::endl;

  lwt::LightweightGraph tagger(config, config.outputs.begin()->first);

  LightweightGraph::SeqNodeMap seq = get_sequences(config.input_sequences);
  LightweightGraph::NodeMap in_nodes;
  //in_nodes = get_generated(config.inputs);

  // Loop over the output names and compute the output for each
  for (const auto& output: config.outputs) {
    auto out_vals = tagger.compute(in_nodes, seq, output.first);
    if(IsDebug){
      std::cout << output.first << ":" << std::endl;
      for (const auto& out: out_vals) {
        std::cout << out.first << " " << out.second << std::endl;
      }
    }
    RNNScore = out_vals["out_0"];
    if(IsDebug) std::cout << RNNScore << "    " << Jet1_pt << "    " << Jet1_e
  }
```

Jet tagger application: W/Z tagger

- Historical Boosted Jet Tagger (BJT) tool with implementation of many W/Z/top jet taggers
  - ***atlas/athena/BoostedJetTaggers***
  - cut based taggers, lwtnn DNN, ONNX CNN/DNN
  - constituents based W/Z tagger has been implemented using ONNX
- Same config structure as for baseline taggers,
  - just more info are added as needed

Usual Config file

pT-parametrisations

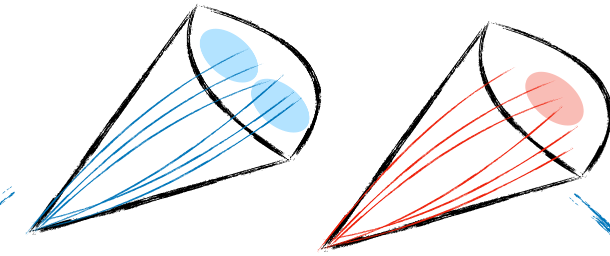
```
DecorationName: SmoothWContained50DisCoJet

pTCutLow: 200.0

pTCutHigh: 3000.0

MassCutLow: (7.180e+01)+(1.124e-02)*pow(x,1)+(-1.721e-05)*pow(x,2)+(4.113e-09)*pow(x,3)

MassCutHigh: (1.255e+02)+(-7.448e-02)*pow(x,1)+(5.402e-05)*pow(x,2)+(-1.118e-08)*pow(x,3)

DisCoDNNCut: (6.731e-01)+(3.434e-05)*pow(x,1)+(1.595e-08)*pow(x,2)+(-7.777e-12)*pow(x,3)

CNNTaggerFileName: CNNTaggers/ONNXModel_VVJJ_Model_m175RScaling_18feb22/model.onnx
nPixelsEta: 40
aEta: -1.
bEta: 1.
nPixelsPhi: 40
aPhi: -1.
bPhi: 1.
nColors: 3
DoRScaling: 1
### the jet pT is in MeV
RScaling_p0: 0.0761
RScaling_p1: 379.e+3

### DisCo model
DisCoTaggerFileName: CNNTaggers/ONNXModel_VVJJ_Model_DisCoJet_13mar22/model.onnx

### features scaling
pT_mean: 1002977.7292303614
pT_std: 626014.5212713333
CNN_mean: 0.44037336334870836
CNN_std: 0.25781137380904184
D2_mean: 1.8977217190151305
D2_std: 1.1563601945608166
nTracks_mean: 32.33996946809005
nTracks_std: 12.06663413423996
```

CNN model

DisCo-DNN model

First constituents based tagger: ONNX helper

- Dedicated class to interface the “physics” data to the ONNX usage
  - [*JSSMLTool.cxx*]
  - inspired from the main ATHENA example, [*AthenaExamples/ AthExOnnxRuntime*]
- ONNX usage:
  - use the shared ATHENA service [*AthOnnxruntimeService*]
  - inputs in Ort::Value format
- So far, implemented DNN based and CNN based architectures



(Physics) inputs preparation

```
// for reading jet images
std::vector<std::vector<float>> JSSMLTool::ReadJetImagePixels( std::vector<TH2D> Images )
{

  int number_of_images = 1;
  int n_rows = m_nbins_eta;
  int n_cols = m_nbins_phi;
  int n_colors = m_ncolors;

  std::vector<std::vector<float>> input_tensor_values;
  input_tensor_values.resize(1, std::vector<float>(n_rows*n_cols*n_colors));

  for(int iImage=0; iImage<number_of_images; ++iImage){ // to be removed...
    for(int iRow=0; iRow<n_rows; ++iRow){
      for(int iColumn=0; iColumn<n_cols; ++iColumn){
        for(int iColor=0; iColor<n_colors; ++iColor){
          input_tensor_values[iImage][ (n_colors*n_cols*iRow) + iColumn*n_colors + iColor]
        }
      }
    }
  }
  return input_tensor_values;
}
```

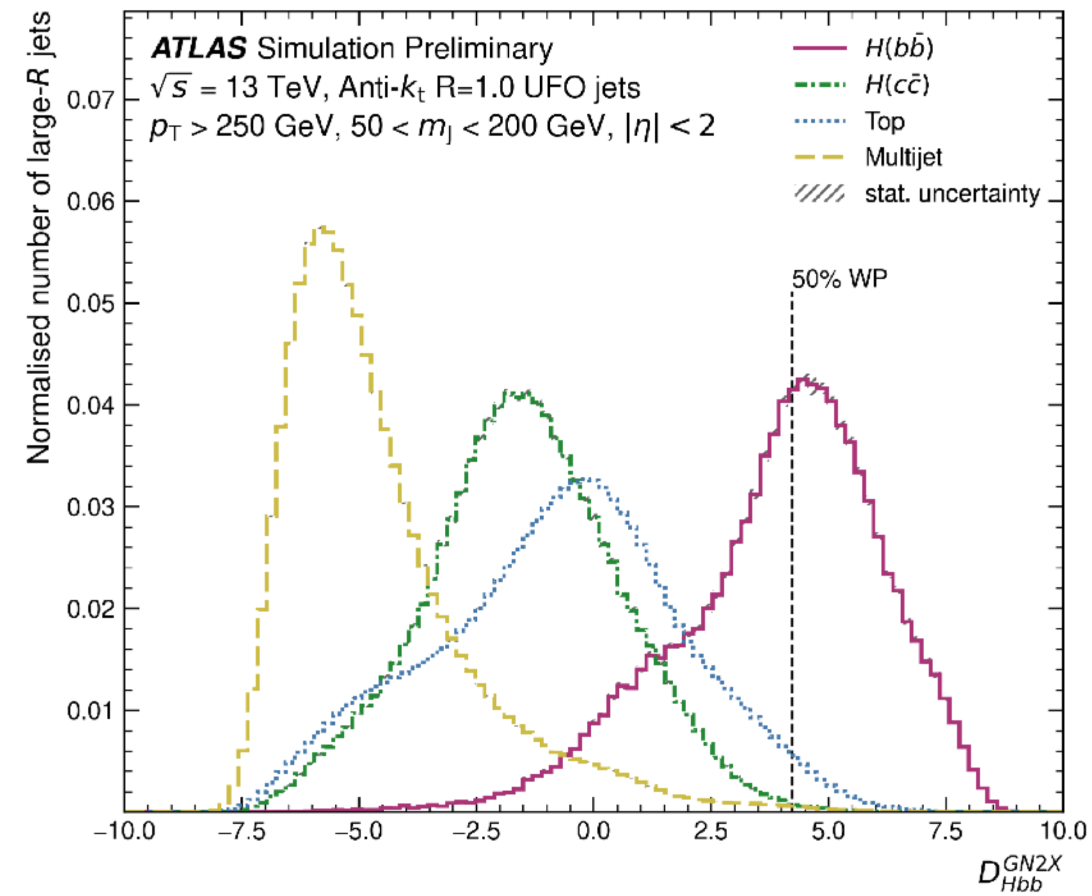
```
// for reading DNN inputs
std::vector<std::vector<float>> JSSMLTool::ReadJSSInputs() //functio
{

  int nDim (1);
  int nVars = m_nvars;

  std::vector<std::vector<float>> input_tensor_values;
  input_tensor_values.resize(1, std::vector<float>(nVars));

  m_JSSInputMap = {
    {0,"pT"}, {1,"CNN"}, {2,"D2"}, {3,"nTracks"}, {4,"ZCut12"},
    {5,"Tau1_wta"}, {6,"Tau2_wta"}, {7,"Tau3_wta"},
    {8,"KtDR"}, {9,"Split12"}, {10,"Split23"},
    {11,"ECF1"}, {12,"ECF2"}, {13,"ECF3"},
    {14,"Angularity"}, {15,"FoxWolfram0"}, {16,"FoxWolfram2"},
    {17,"Aplanarity"}, {18,"PlanarFlow"}, {19,"Qw"},
  };

  for(int i=0;i<nDim;++i){ // to be removed...
    for(int v=0; v<nVars; ++v){
      input_tensor_values[i][v] = m_JSSVars[m_JSSInputMap[v]];
        }
  }
  return input_tensor_values;
}
```

Retrieve the ML model

```
//preparing container to hold input data

size_t input_tensor_size = m_nbins_eta*m_nbins_phi*m_ncolors;
std::vector<float> input_tensor_values(input_tensor_size);

int testSample = 0;
input_tensor_values = input_tensor_values_[testSample];

//preparing container to hold output data
int output_tensor_values = output_tensor_values_[testSample];

// create input tensor object from data values
auto memory_info = Ort::MemoryInfo::CreateCpu(OrtArenaAllocator, OrtMemTypeDefault);
Ort::Value input_tensor = Ort::Value::CreateTensor<float>(memory_info, input_tensor_va
assert(input_tensor.IsTensor());

auto output_tensors = m_session->Run(Ort::RunOptions{nullptr}, m_input_node_names.data
assert(output_tensors.size() == 1 && output_tensors.front().IsTensor());
```

Jet tagger application example: Hbb/cc tagger

- Implementation in the FlavorTagging ATLAS groups
- Supported taggers (based on GNN/transformers!)
  - ***small-R jets b-/c-tagging***
  - ***large-R jets Xbb/cc tagging***
- More advanced architecture already available
  - more work done in the metadata and handling of the tool
  - you can get nice inspiration!



BTagging/BTagConfig.py

```
        "AntiKt10UFOCSSKSoftDropBeta100Zcut10": [
            "BTagging/20230413/gn2xv00/antikt10ufo/network.onnx",
            "BTagging/20230413/gn2xwithmassv00/antikt10ufo/network.onnx",
            "BTagging/20230705/gn2xv01/antikt10ufo/network.onnx",
        ],
```

model version configuration

```
    # Associate tracks to the jet
    result.merge(JetParticleAssociationAlgCfg(
        inputFlags,
        jetcol,
        trackCollection,
        JetTrackAssociator,
    ))
```

(Physics) inputs preparation

FlavorTagDiscriminants/GNN.cxx

```
GNN::GNN(const std::string& nn_file, const GNNOptions& o):
  m_onnxUtil(nullptr),
  m_jetLink(jetLinkName),
  m_defaultValue(o.default_output_value)
{
  // Load and initialize the neural network model from the given file path.
  std::string fullPathToOnnxFile = PathResolverFindCalibFile(nn_file);
  m_onnxUtil = std::make_shared<OnnxUtil>(fullPathToOnnxFile);

  // Extract metadata from the ONNX file, primarily about the model's inputs.
  auto lwt_config = m_onnxUtil->getLwtConfig();

  // Create configuration objects for data preprocessing.
  auto [inputs, constituents_configs, options] = dataprep::createGetterConfig(
      lwt_config, o.flip_config, o.variable_remapping, o.track_link_type);
```

Extract metadata from the ONNX model

First GNN based tagger: ONNX helper

FlavorTagDiscriminants/GNN.cxx

- Implementation in the FlavorTagging ATLAS groups
- Also in this case the core part is using ONNX session
  - after physics data preparation into tensor data
  - inputs prepared in Ort::Value format

Retrieve the ML model

```
  std::vector<float> input_tensor_values;

  // create input tensor object from data values
  auto memory_info = Ort::MemoryInfo::CreateCpu(
    OrtArenaAllocator, OrtMemTypeDefault
  );
  std::vector<Ort::Value> input_tensors;
  for (auto const &node_name : m_input_node_names){
    input_tensors.push_back(Ort::Value::CreateTensor<float>(
      memory_info, gnn_inputs.at(node_name).first.data(), gnn_inputs.at(node_name).fi
      gnn_inputs.at(node_name).second.data(), gnn_inputs.at(node_name).second.size())
    );
  }

  // casting vector<string> to vector<const char*>. this is what ORT expects
  std::vector<const char*> input_node_names;
  input_node_names.reserve(m_input_node_names.size());
  for (const auto& name : m_input_node_names) {
    input_node_names.push_back(name.c_str());
  }
  std::vector<const char*> output_node_names;
  output_node_names.reserve(m_output_nodes.size());
  for (const auto& node : m_output_nodes) {
    output_node_names.push_back(node.name_in_model.c_str());
  }

  // score model & input tensor, get back output tensor
  // Although Session::Run is non-const, the onnx authors say
  // it is safe to call from multiple threads:
  //  https://github.com/microsoft/onnxruntime/discussions/10107
  Ort::Session& session ATLAS_THREAD_SAFE = *m_session;
  auto output_tensors = session.Run(Ort::RunOptions{nullptr},
    input_node_names.data(), input_tensors.data(), input_node_names.size(),
    output_node_names.data(), output_node_names.size()
  );
```

The talk in one slide

	lwtnn (jet tagging, flavour tagging, analyses)	ONNX (jet tagging)	ONNX (flavour tagging)
available (examples)		yes	
ATHENA links	[*BoostedJetTaggers/JSSWTopTaggerDNN.cxx*]	[*AthOnnxRuntimeBJT/JSSMLTool.cxx*]	[*FlavorTagDiscriminants/GNN.cxx*]
model converter	supported	need an easy script	
supported operators	~20	many more (but not universal)	
model metadata	human friendly .json	need to decode the information	
data pre-paration		we are the physicists!	
feed the inputs	config for features scaling	similar but need to take care of the scaling	
tool configuration	depends by the case	more user level configuration (easy to implement your custom tagger!)	more central configuration
architectures exploited so far	DNN/RNN/etc	DNN/CNN	DNN/GNN
ATLAS fw step	Derivation/Analysis level	Analysis level	Derivation level

Conclusions

ONNX based inference in ATLAS

- Inference is a crucial point in ML tool developments
  - production of a specific tool (1 jet tagger —> many analyses)
  - reproducibility of physics results
- Since the “dark age” people have extensively started to use tools for inference
  - lwtnn mainly, migrating to ONNX now
  - ONNX supported as ATLAS external, wider operators support
- ONNX example code application in the (public) ATHENA software

Thanks for listening, hope it can be useful for you!!!

One ~~ring to rule all~~ of them

One inference tool to run all the analyses

Do it! Without inference it will be really hard to reproduce your fancy ML model!

backup

ATHENA software

https://atlassoftwaredocs.web.cern.ch/athena/athena-intro/

Athena Introduction

Last update: 06 Mar 2024 [History] [Edit]

Athena is based on the common Gaudi framework that is used by ATLAS, LHCb and FCC.

Athena code is hosted in the CERN GitLab service where the repository owned by the `atlas` user plays a special role as the repository from which all production releases will be built. In the ATLAS development workflow this repository is also the merge target for developments from individual users or groups. This repository includes a number of different *branches* (defined as "an independent line of development") being actively committed to at any given time, which may be synchronized to varying degrees (either manually or via automated procedures) depending on their purpose. There is much more information about the development workflow in the ATLAS git development tutorial.

The main projects are:

Project	**Purpose**
Athena	Reconstruction and Derivation production*
AthGeneration	For event generation
AthSimulation	For full Geant4 simulation
AthAnalysisBase	Athena based analysis
AnalysisBase	Non-athena ROOT based analysis
DetCommon	For reading trigger configuration when e.g. configuring L1 Hardware

Motivation: Production ML in HEP

HEP Land

- Lots of C++
- Event Pipelines
- Lots of legacy
- Moves slow

ML Land

- Python
- Batched
- Minimal legacy
- Moves fast

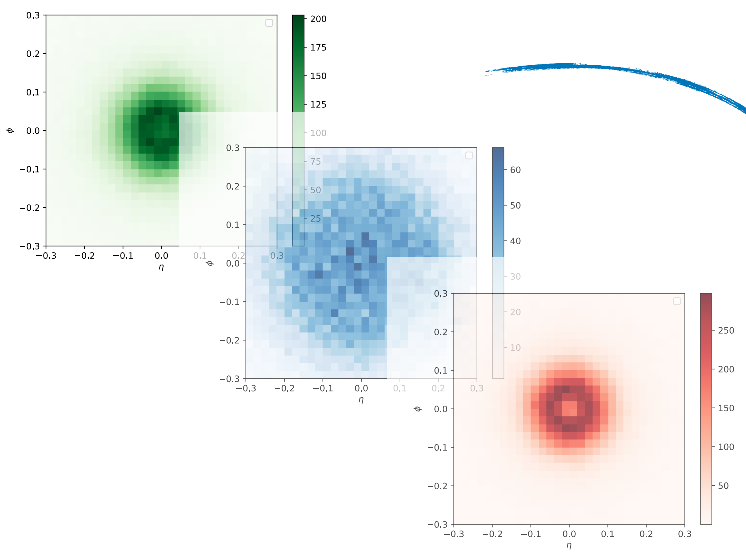
- Many Dependencies

2

What do need to retrieve? And how? ONNX runtime

ONNX runtime link

	CNN	(DisCo)DNN
Data pre-processing	ImageBuilding	Access to JSS variables
Inputs	3x40x40 pixels	N variables after features scaling
model	via ONNX	via ONNX



CNNScore

Low-level, jet constituents (taste-aware network)

DisCoJet is made by 2 scores

DNNScore
(Mass de-correlated via DisCo)

High-Level Jet variables

Jet D2

Jet nTracks

Other O(15) JSS variables

Jet Mass

Tagger WP(s)

DisCo reference and implementation here: arxiv.2001.05310

DisCoJet in the BJT: implementation (I)

[Retrieving NN scores]

Initialise the tool:

- we need help (*AthExOnnxRuntime_BJT*) to interface to ONNX session
- exploit the nice configuration file feature to set whatever parameter you would need

```cpp
if(m_LoadCNNTagger){
  // init tool
  std::string CNNTaggerFileName = m_configReader.GetValue("CNNTaggerFileName", "aaa");
  std::string m_ModelPath = ("/data/BoostedJetTaggers/SmoothedWZTaggers/"+CNNTaggerFileName);

  ATH_MSG_INFO("SmoothedWZDisCoJetTagger::MLBosonTagger()" << "   + ModelPath " << m_ModelPath );
  m_MLBosonTagger = new AthONNX::CxxApiAlgorithm("MLBosonTagger", NULL, m_ModelPath);
  ATH_CHECK( m_MLBosonTagger -> initialize() );

  // get model paramters from the config file
  ATH_MSG_INFO("SmoothedWZDisCoJetTagger::MLBosonTagger() read value from config" );

  // set parameters
  ATH_CHECK( m_MLBosonTagger->setProperty("nPixelsEta", (int)m_configReader.GetValue("nPixelsEta", -99)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("aEta", m_configReader.GetValue("aEta", -99.)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("bEta", m_configReader.GetValue("bEta", -99.)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("nPixelsPhi", (int)m_configReader.GetValue("nPixelsPhi", -99)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("aPhi", m_configReader.GetValue("aPhi", -99.)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("bPhi", m_configReader.GetValue("bPhi", -99.)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("nColors", m_configReader.GetValue("nColors", -99)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("DoRScaling", (bool)m_configReader.GetValue("DoRScaling", -99)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("RScaling_p0", m_configReader.GetValue("RScaling_p0", -99.)) );
  ATH_CHECK( m_MLBosonTagger->setProperty("RScaling_p1", m_configReader.GetValue("RScaling_p1", -99.)) );

  m_MLBosonTagger -> PrintInfo();

}
```

Loop jet by jet:

```cpp
// Retrieve the CNN score
float jet_cnn (-99.);
float jet_disco (-99.);

if(m_LoadCNNTagger){
  jet_cnn = GetCNNScore(&jet);
  (*m_dec_cnn)(jet) = jet_cnn;
}
if(m_UseDisCoTagger){
  auto scaler = ReadScaler(); // this shoud be moved in the init
  auto JSSVars = GetJSSVars(jet, scaler);
  jet_disco = GetDisCoDNNScore(jet, JSSVars);
  (*m_dec_disco)(jet) = jet_disco;
}
```

Retrieve the info you need:

- CNN Score
- DNN Score

```cpp
// use ML tool on constituents
m_MLBosonTagger -> MakeJetImage("Charged" , jet, csts_charged );
m_MLBosonTagger -> MakeJetImage("Neutral" , jet, csts_neutral );
m_MLBosonTagger -> MakeJetImage("Combined", jet, csts_combined);

// evaluate the model
ATH_CHECK( m_MLBosonTagger -> execute() );

// save ML score
CNNScore = m_MLBosonTagger -> GetScore();
//  std::cout << " --- CNNScore, " << CNNScore << std::endl;

// reset images and go to next jet
m_MLBosonTagger -> ResetImages();

return CNNScore;
```

Actions for the CNN:

- build images
- load the score

Good practise!
check ALWAYS that the C++ implementation is matching (within float precision) your result from keras!

```cpp
float SmoothedWZDisCoJetTagger::GetDisCoDNNScore(const xAOD::Jet& jet, co

  // init value
  float DisCoDNNScore (-99.);

  // set inputs
  //auto JSSVars = GetJSSVars(jet);
  //std::map<std::string, double> JSSVars = GetJSSVars(jet);

  m_MLBosonTagger_DisCo -> SetJSSInputs(JSSVars);

  // evaluate the model
  ATH_CHECK( m_MLBosonTagger_DisCo -> executeDisCo() );

  // save ML score
  DisCoDNNScore = m_MLBosonTagger_DisCo -> GetDNNScore();
  //  std::cout << " --- CNNScore, " << CNNScore << std::endl;

  return DisCoDNNScore;

}
```

Actions for the DNN:

- retrieve all JSS vars
- load the score

DisCoJet in the BJT: implementation (II)

[Actual tagger WP]

```cpp
ATH_MSG_INFO( "Smoothed WZ Tagger tool initialized" );
ATH_MSG_INFO( "  Mass cut low      : " << m_strMassCutLow );
ATH_MSG_INFO( "  Mass cut High     : " << m_strMassCutHigh );
if( m_UseCNNTagger )
  ATH_MSG_INFO( "  CNN cut low       : " << m_strCNNCut );
else if( m_UseDisCoTagger )
  ATH_MSG_INFO( "  DisCoDNN cut low       : " << m_strDisCoCut );
else
  ATH_MSG_INFO( "  D2 cut low        : " << m_strD2Cut );
if ( m_useNtrk )
  ATH_MSG_INFO( "  Ntrk cut low      : " << m_strNtrkCut );
ATH_MSG_INFO( "  Decorate          : " << m_decorate );
ATH_MSG_INFO( "  DecorationName    : " << m_decorationName );
```

Configuring the tagger:

- once all the variables are on the table, you just need to define the WP (i.e. cuts)
- example.1: UFO 3-Var Tagger
  - Mass, D2, nTracks
- example.2: UFO 3-Var CNN low-level Tagger
  - Mass, CNN, nTracks
- example.1: UFO 2-Var DisCoJet Tagger
  - Mass, DisCoJet

Evaluate the cuts:

- the parametric cuts (as the pT) coming from the config file

```cpp
/// Evaluate the values of the upper and lower mass bounds and the d2 cut
float cut_mass_low  = m_funcMassCutLow ->Eval(jet_pt);
float cut_mass_high = m_funcMassCutHigh->Eval(jet_pt);
float cut_d2 (-99.), cut_cnn (-99.), cut_disco (-99.);
if(m_UseCNNTagger)         cut_cnn        = m_funcCNNCut      ->Eval(jet_pt);
else if(m_UseDisCoTagger)  cut_disco      = m_funcDisCoCut    ->Eval(jet_pt);
else                       cut_d2         = m_funcD2Cut       ->Eval(jet_pt);
```

```cpp
/// Evaluate the cut criteria on mass and d2
if(m_UseCNNTagger){
  ATH_MSG_VERBOSE( "Cut Values : MassWindow = [" << cut_mass_low << "," << cut_mass_
  ATH_MSG_VERBOSE( "Cut Values : JetMass = " << jet_mass << ", CNN = " << jet_cnn )
  if ( jet_cnn > cut_cnn ) m_accept.setCutResult( "PassCNN", true );
}
else if(m_UseDisCoTagger){
  ATH_MSG_VERBOSE( "Cut Values : MassWindow = [" << cut_mass_low << "," << cut_mass_
  ATH_MSG_VERBOSE( "Cut Values : JetMass = " << jet_mass << ", CNN = " << jet_disco
  if ( jet_disco > cut_disco ) m_accept.setCutResult( "PassDisCoDNN", true );
}
else{
  ATH_MSG_VERBOSE( "Cut Values : MassWindow = [" << cut_mass_low << "," << cut_mass_
  ATH_MSG_VERBOSE( "Cut Values : JetMass = " << jet_mass << ", D2 = " << jet_d2 );
  if ( jet_d2 < cut_d2 ) m_accept.setCutResult( "PassD2", true );
}
```

Apply the cuts (tagger WP):

- apply the cut and store the results for the user!

The full Tagger infrastructure has been reused!!!

DisCoJet in the BJT: running time

- The full infrastructure of the tool has been extended for this tagger
  - besides the specific variable or score, the actual 'Tagger' steps are always the same
- The user will not see any difference rather than in the specific print-outs and in the output results!

```
SmoothedContainedWTagg...INFO    Using config file : SmoothedContainedWTagger_AntiKt10VanillaSD_FixedSignalEfficiency50_2VarDisCoJet_v1.dat
SmoothedContainedWTagg...INFO    SmoothedWZTagger::MLBosonTagger()   + ModelPath /data/BoostedJetTaggers/SmoothedWZTaggers/CNNTaggers/ONNXModel_VVJJ_Model_m175RScaling_18feb22/model.onnx
PathResolver             WARNING Locating dev file dev/MLTest/2020-03-31/t10k-images-idx3-ubyte. Do not let this propagate to a release
PathResolver             WARNING Locating dev file dev/MLTest/2020-03-31/t10k-labels-idx1-ubyte. Do not let this propagate to a release
MLBosonTagger            INFO    Using model file: /srv/workDir/usr/WorkDir/0.0.1/InstallArea/x86_64-centos7-gcc8-opt//data/BoostedJetTaggers/SmoothedWZTaggers/CNNTaggers/
ONNXModel_VVJJ_Model_m175RScaling_18feb22/model.onnx
MLBosonTagger            INFO    Using pixel file: /cvmfs/atlas.cern.ch/repo/sw/database/GroupData/dev/MLTest/2020-03-31/t10k-images-idx3-ubyte
MLBosonTagger            INFO    Using pixel file: /cvmfs/atlas.cern.ch/repo/sw/database/GroupData/dev/MLTest/2020-03-31/t10k-labels-idx1-ubyte
MLBosonTagger            INFO    Created the ONNX Runtime session
MLBosonTagger            INFO    Output 0 : num_dims= 2
MLBosonTagger            INFO    Output0 : dim 0= 1
MLBosonTagger            INFO    Output0 : dim 1= 1
SmoothedContainedWTagg...INFO    SmoothedWZTagger::MLBosonTagger() read value from config
SmoothedContainedWTagg...INFO    SmoothedWZTagger::MLBosonTagger()   + ModelPath /data/BoostedJetTaggers/SmoothedWZTaggers/CNNTaggers/ONNXModel_VVJJ_Model_DisCoJet_13mar22/model.onnx
PathResolver             WARNING Locating dev file dev/MLTest/2020-03-31/t10k-images-idx3-ubyte. Do not let this propagate to a release
PathResolver             WARNING Locating dev file dev/MLTest/2020-03-31/t10k-labels-idx1-ubyte. Do not let this propagate to a release
MLBosonTaggerDisCo       INFO    Using model file: /srv/workDir/usr/WorkDir/0.0.1/InstallArea/x86_64-centos7-gcc8-opt//data/BoostedJetTaggers/SmoothedWZTaggers/CNNTaggers/
ONNXModel_VVJJ_Model_DisCoJet_13mar22/model.onnx
MLBosonTaggerDisCo       INFO    Using pixel file: /cvmfs/atlas.cern.ch/repo/sw/database/GroupData/dev/MLTest/2020-03-31/t10k-images-idx3-ubyte
MLBosonTaggerDisCo       INFO    Using pixel file: /cvmfs/atlas.cern.ch/repo/sw/database/GroupData/dev/MLTest/2020-03-31/t10k-labels-idx1-ubyte
MLBosonTaggerDisCo       INFO    Created the ONNX Runtime session
MLBosonTaggerDisCo       INFO    Output 0 : num_dims= 2
MLBosonTaggerDisCo       INFO    Output0 : dim 0= 1
MLBosonTaggerDisCo       INFO    Output0 : dim 1= 1
SmoothedContainedWTagg...INFO    SmoothedWZTagger::MLBosonTagger() read value from config
SmoothedContainedWTagg...INFO    Smoothed WZ Tagger tool initialized
SmoothedContainedWTagg...INFO      Mass cut low      : (7.180e+01)+(1.124e-02)*pow(x,1)+(-1.721e-05)*pow(x,2)+(4.113e-09)*pow(x,3)
SmoothedContainedWTagg...INFO      Mass cut High     : (1.255e+02)+(-7.448e-02)*pow(x,1)+(5.402e-05)*pow(x,2)+(-1.118e-08)*pow(x,3)
SmoothedContainedWTagg...INFO      DisCoDNN cut low       : (6.731e-01)+(3.434e-05)*pow(x,1)+(1.595e-08)*pow(x,2)+(-7.777e-12)*pow(x,3)
SmoothedContainedWTagg...INFO      Decorate          : 1
SmoothedContainedWTagg...INFO      DecorationName    : SmoothWContained50DisCoJet
SmoothedContainedWTagg...INFO      Pt cut low        : 200
SmoothedContainedWTagg...INFO      Pt cut high       : 3000
SmoothedContainedWTagg...INFO    After tagging, you will have access to the following cuts as a Root::TAccept : (<NCut>) <cut> : <description>)
SmoothedContainedWTagg...INFO      (0)  PassMassLow : mJet > mCutLow
SmoothedContainedWTagg...INFO      (1)  PassMassHigh : mJet < mCutHigh
SmoothedContainedWTagg...INFO      (2)  PassDisCoDNN : DisCoDNNJet > DisCoDNNCut
SmoothedContainedWTagg...INFO    Decorators that will be attached to jet :
SmoothedContainedWTagg...INFO      SmoothWContained50DisCoJet_Cut_mlow : lower mass cut
SmoothedContainedWTagg...INFO      SmoothWContained50DisCoJet_Cut_mhigh : upper mass cut
SmoothedContainedWTagg...INFO    Additional decorators that will be attached to jet :
SmoothedContainedWTagg...INFO      SmoothWContained50DisCoJet_Cut_DisCoDNN : DisCoDNN cut
SmoothedContainedWTagg...INFO    Initializing SmoothedWZTagger tool
```

Keeping the same messages and debugging as the baseline tagger

https://indico.cern.ch/event/972791/contributions/4121098/attachments/2156523/3637588/FastML_2020.pdf

Onnxruntime Tensor (Ort::Value)

.onnx model takes input in Ort::Value format; (here)
e.g. MNIST hand written digit 3D array (1, 28, 28) needs to be converted to 3D Ort::Value before being fed to a .onnx model with input layer dimension: [-1,28,28]