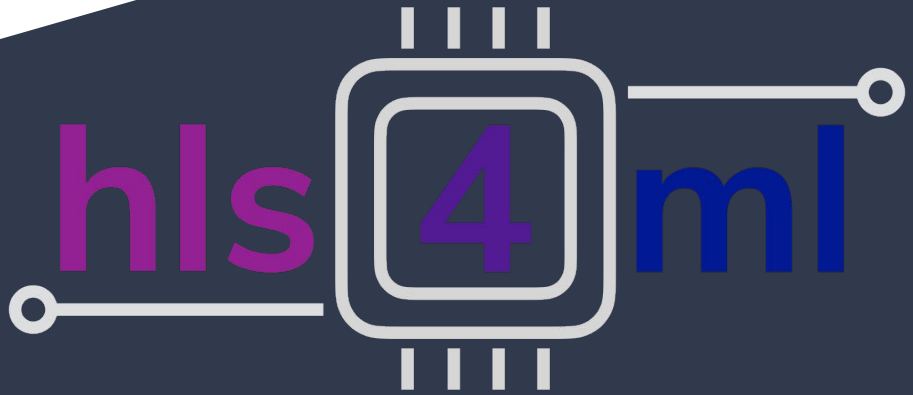
Application of Conifer and HLS4ML in Converting a MET Tagger to FPGA Firmware

Locked-in Leptons

Fatima Rodriguez NIU
Frank Strug UIC

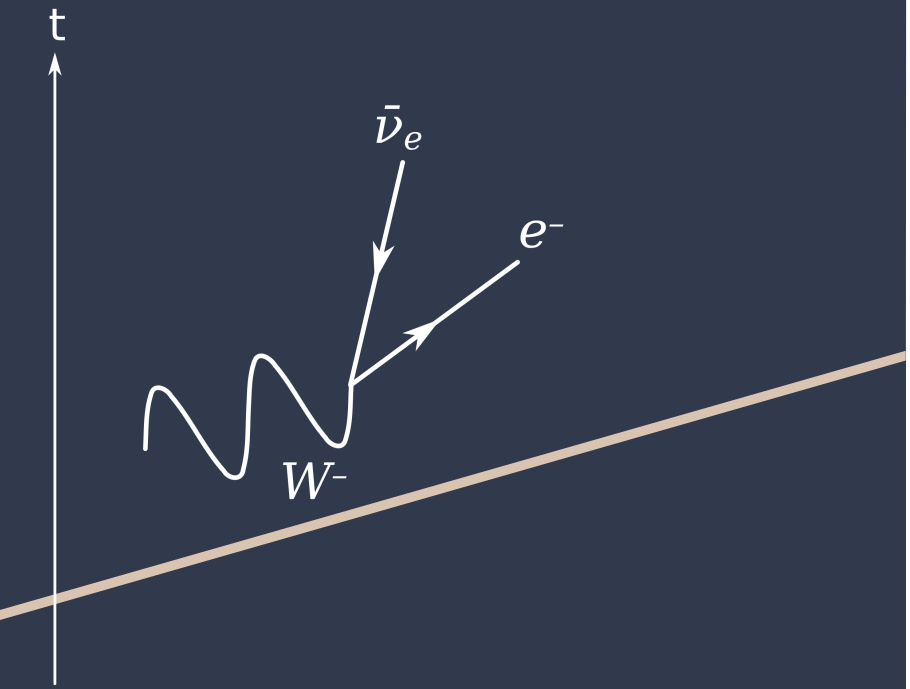
Northern Illinois University
Your Future. Our Focus.



Outline

- Objective and Motivation
- Review of FPGAs
- Introduction to HLS4ML
- Boosted Decision Trees Application
- Dense Neural Network Application
- Conclusion

Objective and Motivation



- **Objective:** Train an ML model to identify events with true MET and convert this MET tagger to FPGA firmware code that can be loaded onto a FPGA.
- **Problem:** ‘Invisible’ particles leave no signatures in a detector, so it is difficult to discern when momentum imbalance is genuine or caused by detector effects. Correct identification of MET from invisible particles important to analyses interested in studying physics of such particles (e.g. Neutrino physics and dark matter searches).
- **Project:** If we can train an accurate discriminator, it could be useful to deploy model on an FPGA at the High Level Trigger (HLT) to trigger on genuine MET. Using Conifer and HLS4ML, we will investigate feasibility of converting ML models to FPGA firmware code by bit-accurate emulation and potential synthesis-level optimizations.

Meet the Data - Data Sources

- Need to constrain training samples for scope of project.
- Neutrinos give true MET. W->enu will serve as a proxy for our signal sample. Assigned label 1.
- Electrons in an event can give rise to false MET signatures due to electron resolution. Z->ee will serve as a proxy for our background training sample. Assigned label 0.

Process	Events (nlep = 1)	Events (nlep = 2)	Total
W+ -> e+nu	12,634,281	41,502	12,675,783
W- -> e-nu	9,395,683	31,893	9,427,576
Z -> ee	16,442,971	21,848,524	38,291,495

Table 1: Training data provided by ATLAS Open Data metadata. From the 13 TeV 2020 dataset for education. Simulated to LO accuracy with up to 3 jets. W samples generated and hadronized with Powheg-Box V2 + Pythia 8. Z samples generated and hadronized with Sherpa 2.2.

Meet the Data - Training

- Data is then combined, shuffled, and split into training and testing set.
- We use 70% of total data for training and the remaining 30% is used for testing model post-training and post conversion to FPGA firmware.
- Model is trained on MET, 2 leading leptons, and 3 leading jets kinematics. 17 total features.

Process	Events (nlep = 1)	Events (nlep = 2)	Total
W+ -> e+nu	12,634,281	41,502	12,675,783
W- -> e-nu	9,395,683	31,893	9,427,576
Z -> ee	16,442,971	21,848,524	38,291,495

Table 1: Training data provided by ATLAS Open Data metadata. From the 13 TeV 2020 dataset for education. Simulated to LO accuracy with up to 3 jets. W samples generated and hadronized with Powheg-Box V2 + Pythia 8. Z samples generated and hadronized with Sherpa 2.2.

Open Data Object Quality Criteria

Final-state categories	Leading object p_T (min) [GeV]	Collection name
$N_l = 1$	25	1lep
$N_l \leq 1$	25	2lep
$N_l = 3$	25	3lep
$N_l \leq 4$	25	4lep
$N_{\text{largeRjet}} \leq 1$ & $N_l = 1$	250 (large-R jet), 25 (lepton)	1largeRjet1lep
$N_{\tau-\text{had}} = 1$ & $N_l = 1$	20 (τ_h), 25 (lepton)	1lep1tau
$N_\gamma \leq 2$	35	GamGam

Table 2: Final state collection definitions and reconstruction algorithm preselection requirements of 13 TeV 2020 data from ATLAS Open Data.Tagger trained on 2lep and 1lep collection.

Electron (e)	Muon (μ)	Photon (γ)
InDet & EMCAL rec.	InDet & MS rec.	InDet & EMCAL rec.
loose identification	loose identification	tight identification
loose isolation	loose isolation	loose isolation
$p_T > 7$ GeV	$p_T > 7$ GeV	$E_T > 25$ GeV
$\|\eta\| < 2.47$	$\|\eta\| < 2.5$	$\|\eta\| < 2.37$

Hadronically decaying τ-leptons (τ_h)	Small-R jets	Large-R jets
InDet & EMCAL rec.	EMCAL & HCAL rec.	EMCAL & HCAL rec.
medium identification	anti-kt, R = 0.4	anti-kt, R = 1.0
$p_T > 20$ GeV	$p_T > 20$ GeV	$p_T > 250$ GeV
$\|\eta\| < 2.5$	$\|\eta\| < 2.5$	$\|\eta\| < 2.0$
1 or 3 associated tracks	b-tagging (MV2c10)	trimming: $R_{sub} = 0.2$, $f_{cut} = 0.05$

Open Data Object Quality Criteria (cont.)

Electrons & Muons	Small-R jets	Photons	Large-R jets	τ_h
$p_T > 25$ GeV	$p_T > 25$ GeV		$p_T < 1500$ GeV	$p_T > 25$ GeV
lep_ptcone30 < 0.15	JVT > 0.59	photon_ptcone30 < 0.065	mass > 50 GeV	
lep_etcone20 < 0.15		photon_etcone20 < 0.065		

Table 3: Final state collection definitions and reconstruction algorithm preselection requirements of 13 TeV 2020 data from ATLAS Open Data continued.

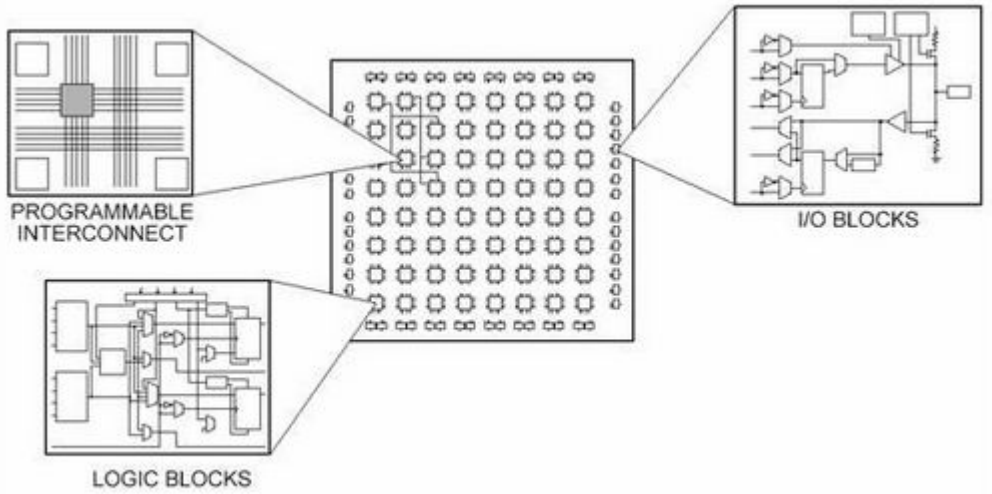
Review of FPGAs

Field-programmable gate arrays (FPGAS) are integrated circuits that can be reconfigured to meet designers needs. Unlike CPUs which are serial machines, FPGA chips contain an array of programmable logic blocks, allowing for parallel computation.

Advantages

- Flexibility & Adaptability
- High Performance
- Power Efficient

Parts of an FPGA



Different parts of an FPGA [source]

Every FPGA has a finite number of predefined resources with programmable interconnects, configurable logic blocks (CLB), and I/O blocks.

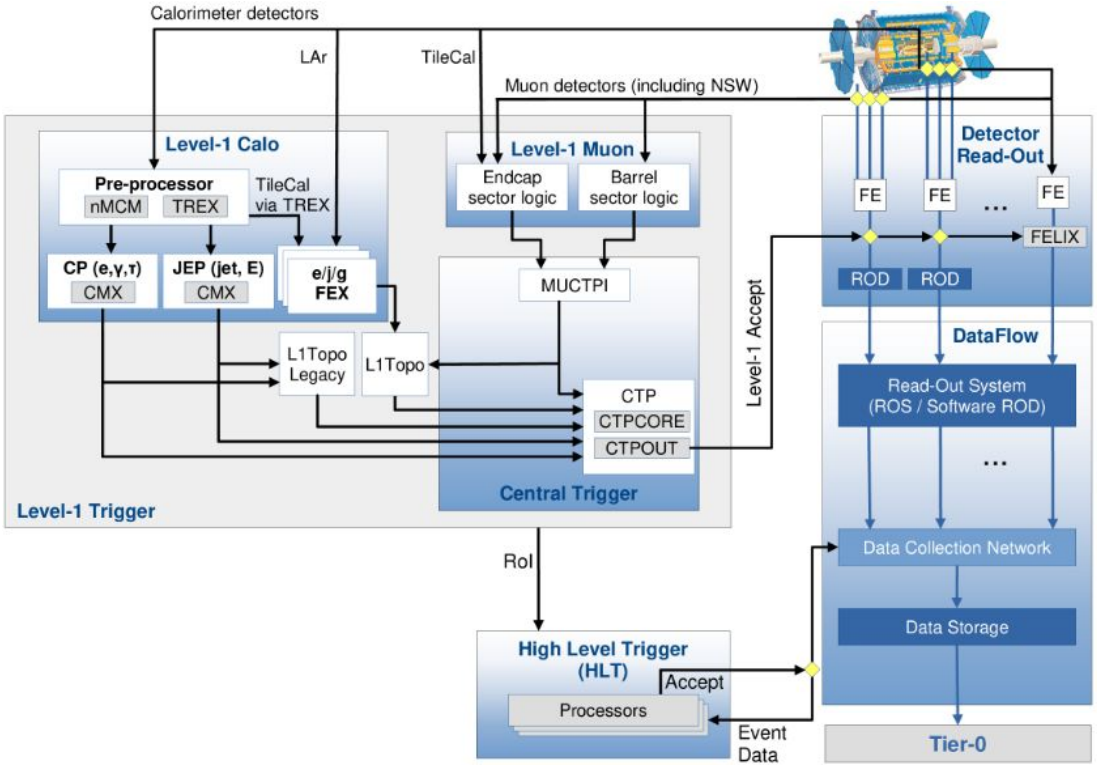
CLBs have three main components: flip-flops, lookup tables (LUTs), and digital signal processing blocks (DSPs)

Flip-Flops are binary shift registers used to synchronize logic and have logical states between clock cycles.

LUTs are arrays of data that map input values to output values.

Digital Signal Processing blocks are specialized hardware which implement addition, multiplication, and multiply-accumulate operations efficiently.

Application to Particle Physics



We've learned from Jahred's talk [ref. here] that each piece of an experiment requires dedicated electronics, which are mostly programmed using ASICS: Application Specific Integrated Circuit.

ASICs are custom circuits but not reconfigurable.

FPGAs can serve as a great alternative due to their flexibility and less power usage.

HLS & Vitis

High-Level Synthesis (HLS) is the process of converting logic written in high-level programming languages, like C/C++, to lower-level hardware description languages like Verilog or VHDL.

Vitis is a software platform that provides tools to write HLS code, simulate and verify designs using testbenches, synthesis HLS code into RTL, and generate a bitstream to program an FPGA.

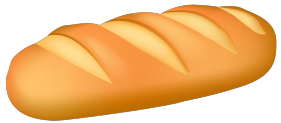
HLS + C++

Vitis

Simulation - making small test batch

Synthesis - building custom bread oven

Bitstream - gather all ingredients and instructions

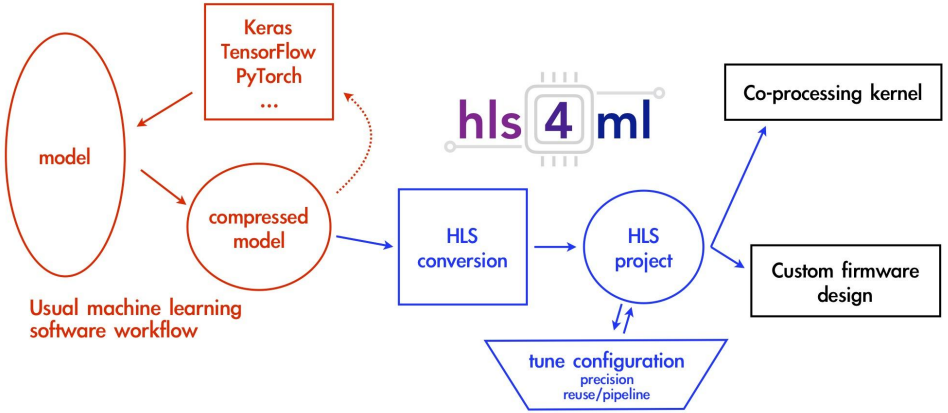


Load bitstream into oven to make perfect bread every time :)

Introduction to hls4ml

HLS4ML is a Python package for machine learning inference in FPGAs.

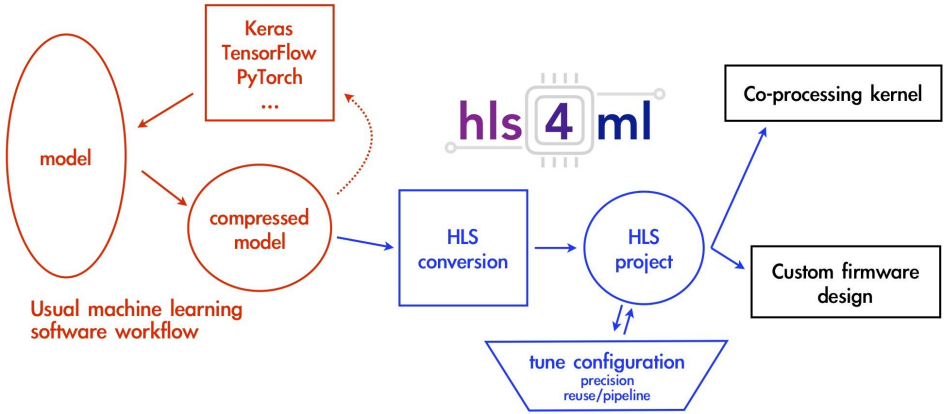
It takes a trained ML model and converts it to HLS C++. Vitis HLS can then compile and simulate the code to synthesis and deploy onto an FPGA.



ML model deployment workflow with hls4ml. [source]

HLS4ML Workflow

- Train some supported python model
- Prepare configuration dictionary
  - Customization highly model dependent
  - Choose target board
  - Adjust precision
- Compile and emulate FPGA firmware code
  - Confirm model behavior preserved after conversion
- Synthesize model
  - Investigate resource utilization and performance



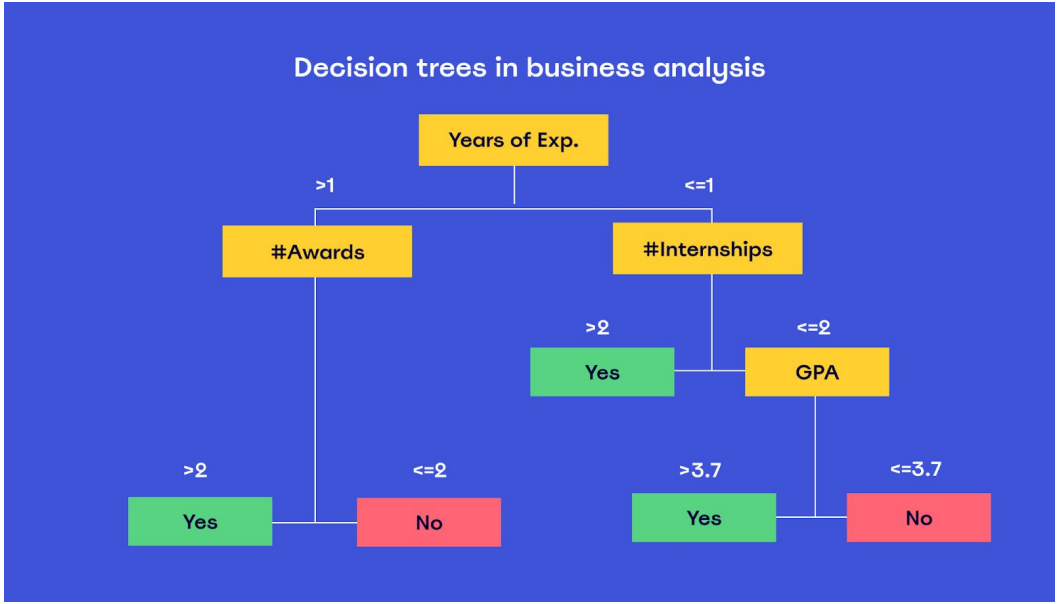
ML model deployment workflow with hls4ml. [source]

Boosted Decision Trees (BDTs) in Conifer (HLS4ML)

- **Conifer** is a python package originally developed as part of HLS4ML which “translates trained Boosted Decision Trees to FPGA firmware for extreme low latency inference.”
- Conifer supports translating popular python BDT libraries (i.e. scikit-learn and XGBoost) to FPGA firmware with XILINX HLS.
- **Applications**: ATLAS Calorimeter TAU Trigger, Tracking Detector Frontend Data Reduction, and Enhancing Blood Vessel Segmentation.

Decision Trees and Regression

- A **decision tree** is a structure in which each internal node represents a test on some input feature with external nodes representing the decision based on the value of said attribute.
- Decision tree regression done when predicting continuous values (probability of being in a ‘class’).



Example decision tree for determination of offering employment.

Gradient Boosting and Decision Trees

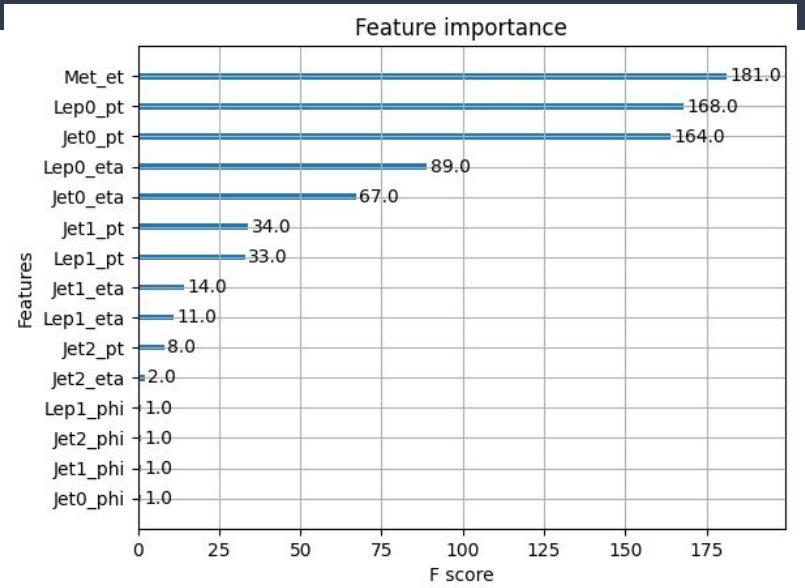
- **Gradient boosting** is the method of combining many weak prediction models to form one strong model. In the application of decision trees, this is the process of training many decision tree regressions to create an ensemble.
- Alleviate overfitting and inaccuracy of single tree models.

From Winnie the Pooh (2011), Tigger teachers Eeyore how to be a tigger. Now Tigger is not the only one!

Boosted Decision Tree Hyperparameters

- We will investigate two different BDT frontends.
  - Scikit-learn.GradientBoostingClassifier
  - XGBoost
- Hyperparameters
  - Max depth: 5
  - Learning rate : 1.0
  - Number of trees: 25
- Frontend Comparisons

	Scikit-learn	XGBoost
AUC (%)	94.4	94.4
Accuracy	0.88229	0.88258
Training time	2hrs 56min	3min 2s

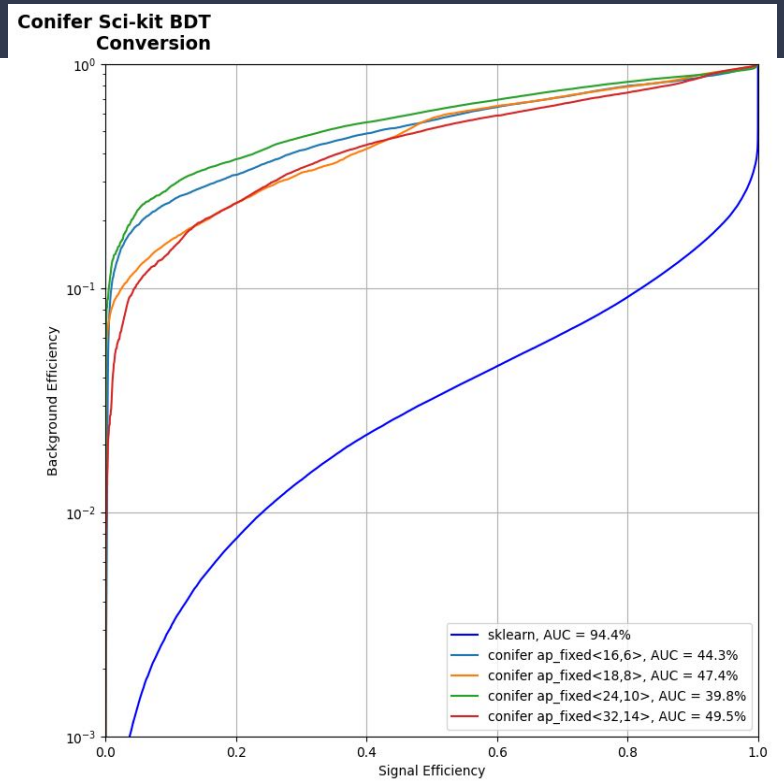


Feature importance for XGBoost MET tagger.

Conifer Conversion of Scikit-learn Model - Emulation

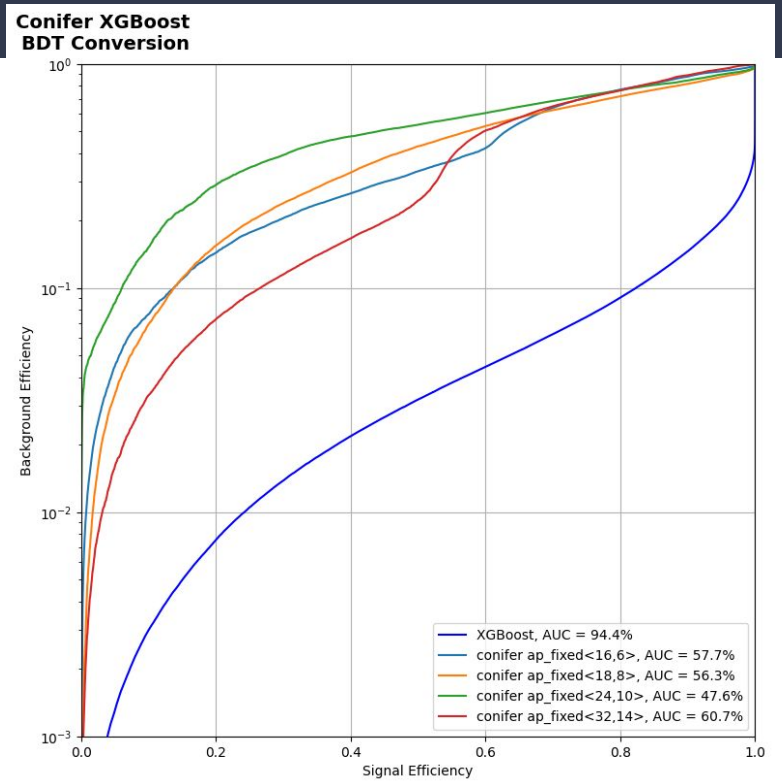
- Default conifer configuration modified to match board used in HLS4ML tutorial
-
```
Conifer HLS Configuration
-----------------------------------------------------
Backend:             xilinxhls
ProjectName:         my_prj
OutputDir:           model_bdt_sklearn/
XilinxPart:          xcvu9p-flgb2104-2L-e
ClockPeriod:         5
Unroll:              True
AcceleratorConfig:   None
Precision:           ap_fixed<18,8>
-----------------------------------------------------
```

- Conifer conversion does worse than randomly guessing



Conifer Conversion of XGBoost Model – Emulation

- Default conifer configuration modified to match board used in HLS4ML tutorial
- ```
  Conifer HLS Configuration
  -------------------------------------------------
  Backend:             xilinxhls
  ProjectName:         my_prj
  OutputDir:           model_bdt_sklearn/
  XilinxPart:          xcvu9p-flgb2104-2L-e
  ClockPeriod:         5
  Unroll:              True
  AcceleratorConfig:   None
  Precision:           ap_fixed<18,8>
  -------------------------------------------------
  ```
- Conifer conversion in general did not preserve behavior of model, but performs better for XGBoost than scikit-learn.


  ```

Synthesis with Conifer - XGBoost

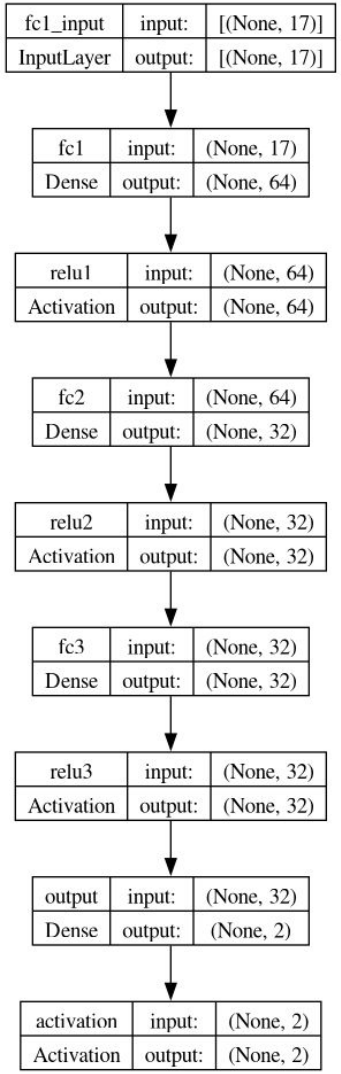
Precision	ap_fixed<16,6>	ap_fixed<18,8>	ap_fixed<24,10>	ap_fixed<32,14>
Latency	3	3	3	3
Interval	1	1	1	1
LUT	28568	30264	34710	40498
LUT (vsynth)	6396	7376	9146	12723
FF	1183	1192	1414	1688
FF (vsynth)	854	863	985	1158
AUC (%)	57.7	56.3	47.6	60.7
Accuracy	0.53020	0.69349	0.70905	0.58917

Applying Dense Neural Network (DNN) in hls4ml

In a **Dense Neural Network**, each layer receives an output from the neuron in the previous layer.

In this application, our model:

- 17 input features
- 4 dense layers
- First three hidden layers use ReLU activation function.
- Last layer uses softmax activation function because final output is binary classification.
- Batch size = 1024
- Epochs = 10
- Total Parameters: 4354

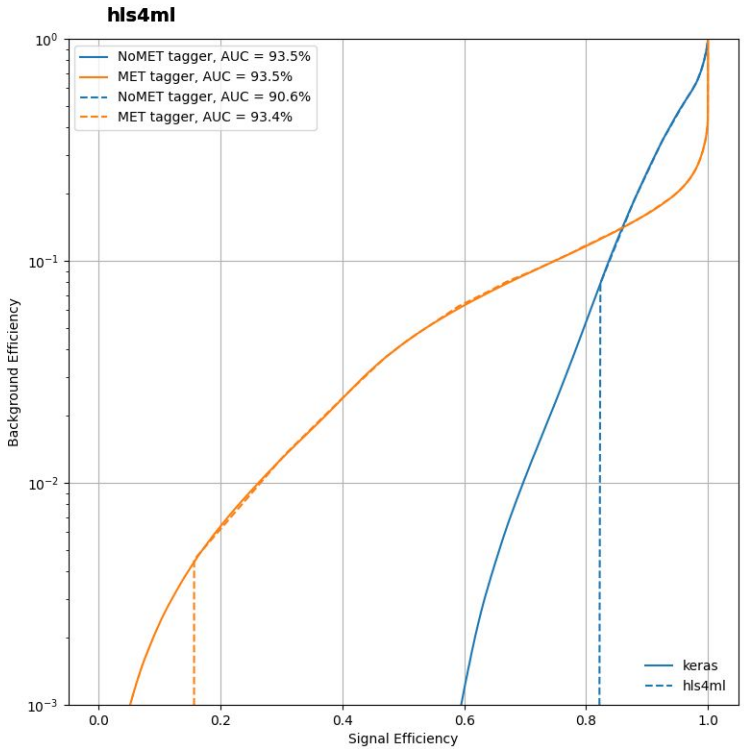


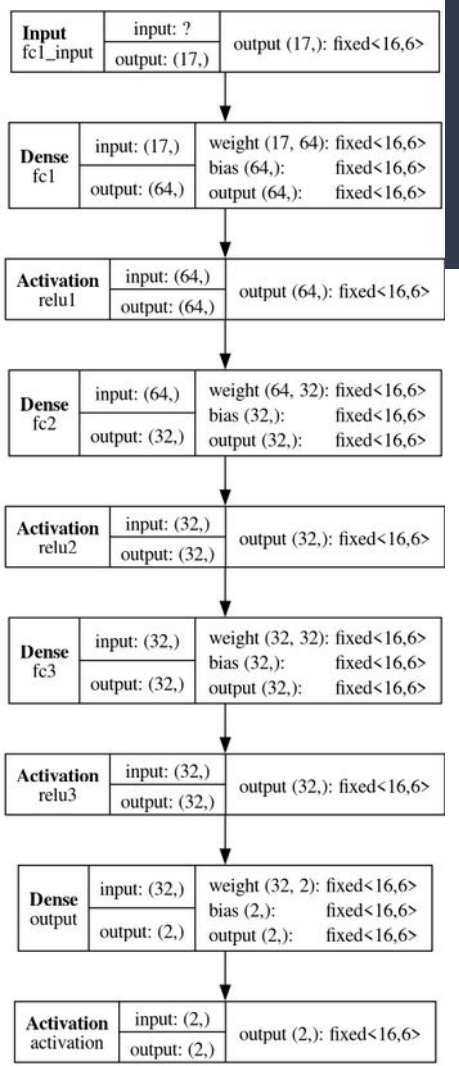
hls4ml Comparison with Keras

Accuracy:

Keras = 0.8920124592675143

hls4ml = 0.8915191370141214





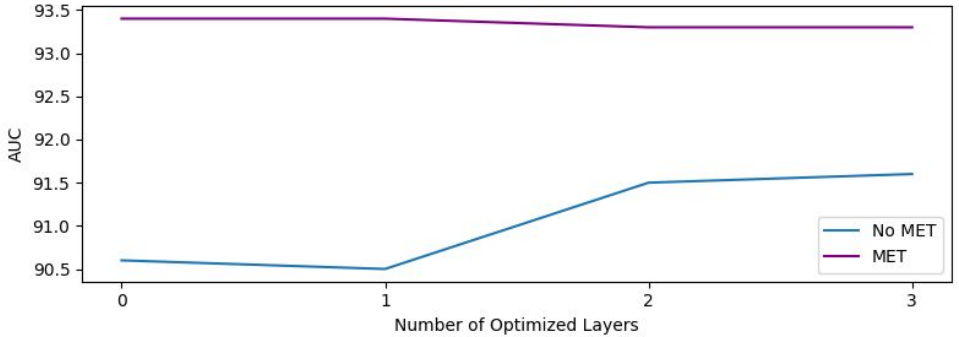
Optimizing hls4ml Model

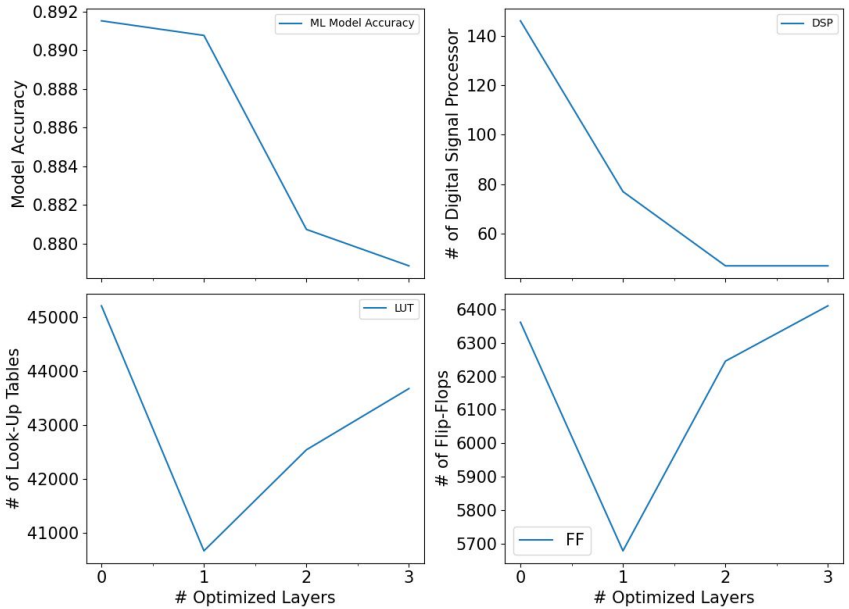
One method of optimizing hl4ml Model is by lowering bit precision per layer.

```
config['LayerName']['fc1']['Precision']['weight'] = 'ap_fixed<8,2>'
```

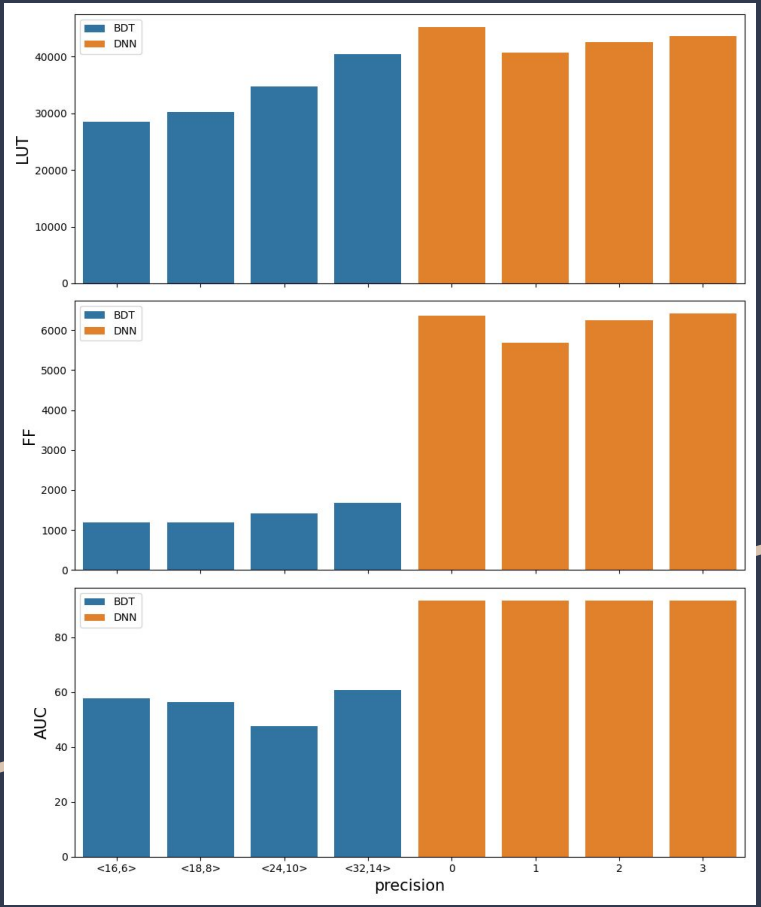
Lowering bit precision → less hardware resource usage (LUTs, DSPs, BRAMs) & faster inference

However, lower precision can degrade model performance, as shown on the left.





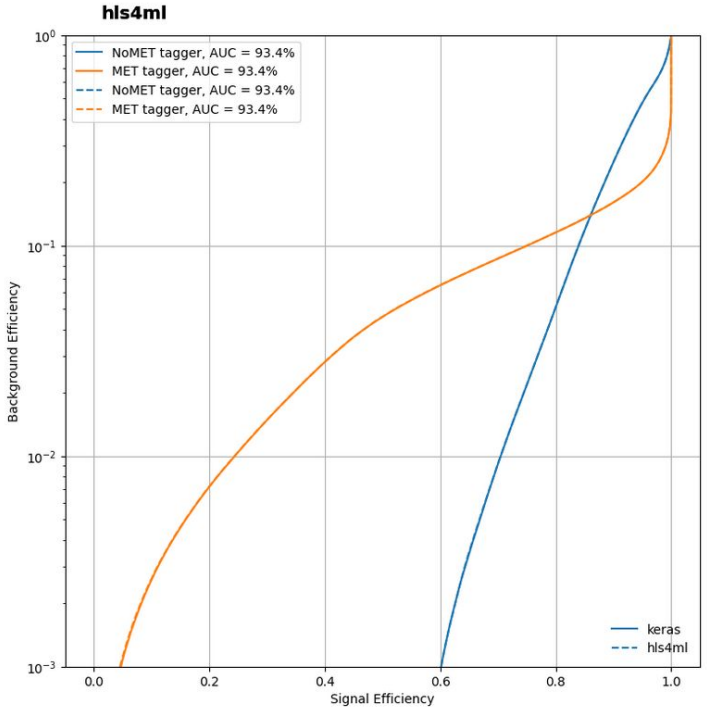
Conclusion



- We were able to successfully synthesize a DNN MET tagger to FPGA firmware which adequately preserved model behavior.
  - A high degree of optimization possible
- We were unsuccessful in synthesizing a BDT MET tagger which preserved model behavior.
  - Unclear root cause of behavior divergence.
    - Little customizability
  - Evidence that conifer can be applied successfully
- Potential Further Development
  - Attempt to design model which can successfully identify MET with l1 information.
- Github repository with source code.

Backup Slide

```
Keras  Accuracy: 0.8623322287304689
hls4ml Accuracy: 0.8622325858977454
Out[89]: <matplotlib.legend.Legend at 0x7f1ee0f2f2c0>
```



{'CSynthesisReport': {'TargetClockPeriod': '5.00',
 'EstimatedClockPeriod': '4.371',
 'BestLatency': '9',
 'WorstLatency': '9',
 'IntervalMin': '1',
 'IntervalMax': '1',
 'BRAM_18K': '1',
 'DSP': '187',
 'FF': '6360',
 'LUT': '46604',
 'URAM': '0',
 'AvailableBRAM_18K': '5376',
 'AvailableDSP': '12288',
 'AvailableFF': '3456000',
 'AvailableLUT': '1728000',
 'AvailableURAM': '1280'}}