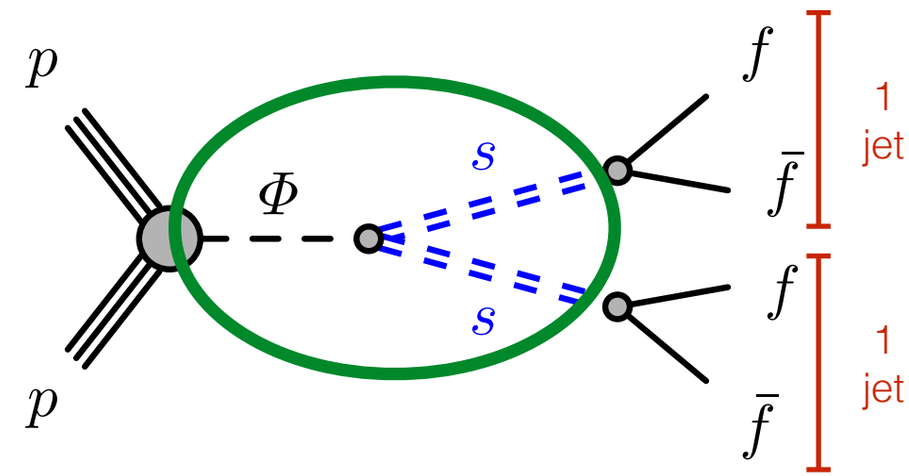
# Designing adversarial machine learning techniques to tag displaced jets in the ATLAS calorimeter

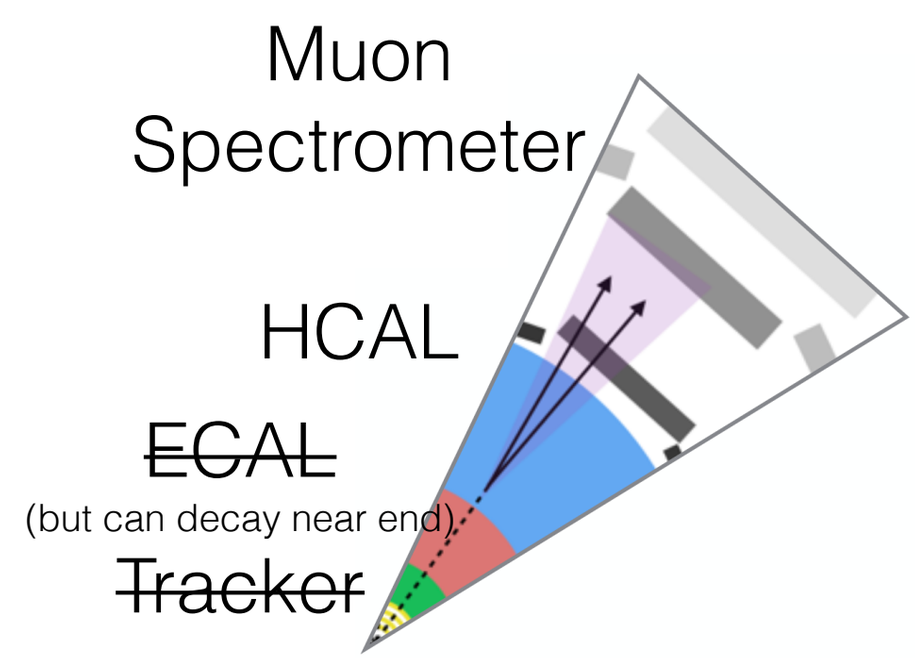
Félix Cormier, on behalf of the CalRatio analysis team

14/07/2021

# Introduction

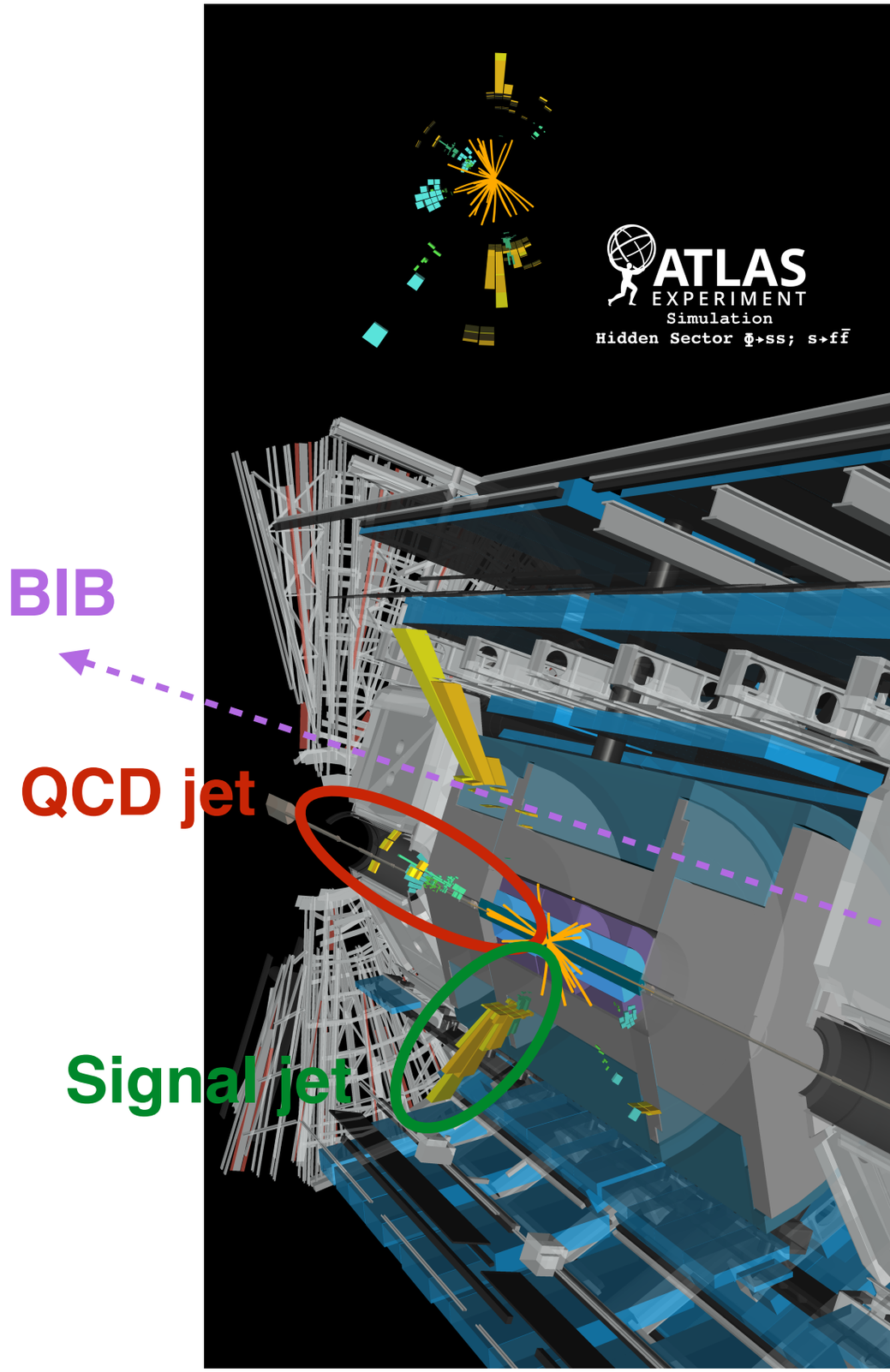
- Main benchmark model: Hidden sector, with **long-lived** neutral scalar s, mediated by heavy neutral boson $\phi$
  - All **Neutral under SM**, so only detected when decaying to standard model
  - Masses of $\Phi$: [60,125,200,400,600,1000]
  - Masses of s are 5-475 GeV
  - Low $\Phi$ masses harder to discern from background than high masses
- Signal objects are narrow jets with no collinear tracks, depositing **majority of energy in HCAL**
- In current analysis expect 1 jet from each of the Long-lived particles



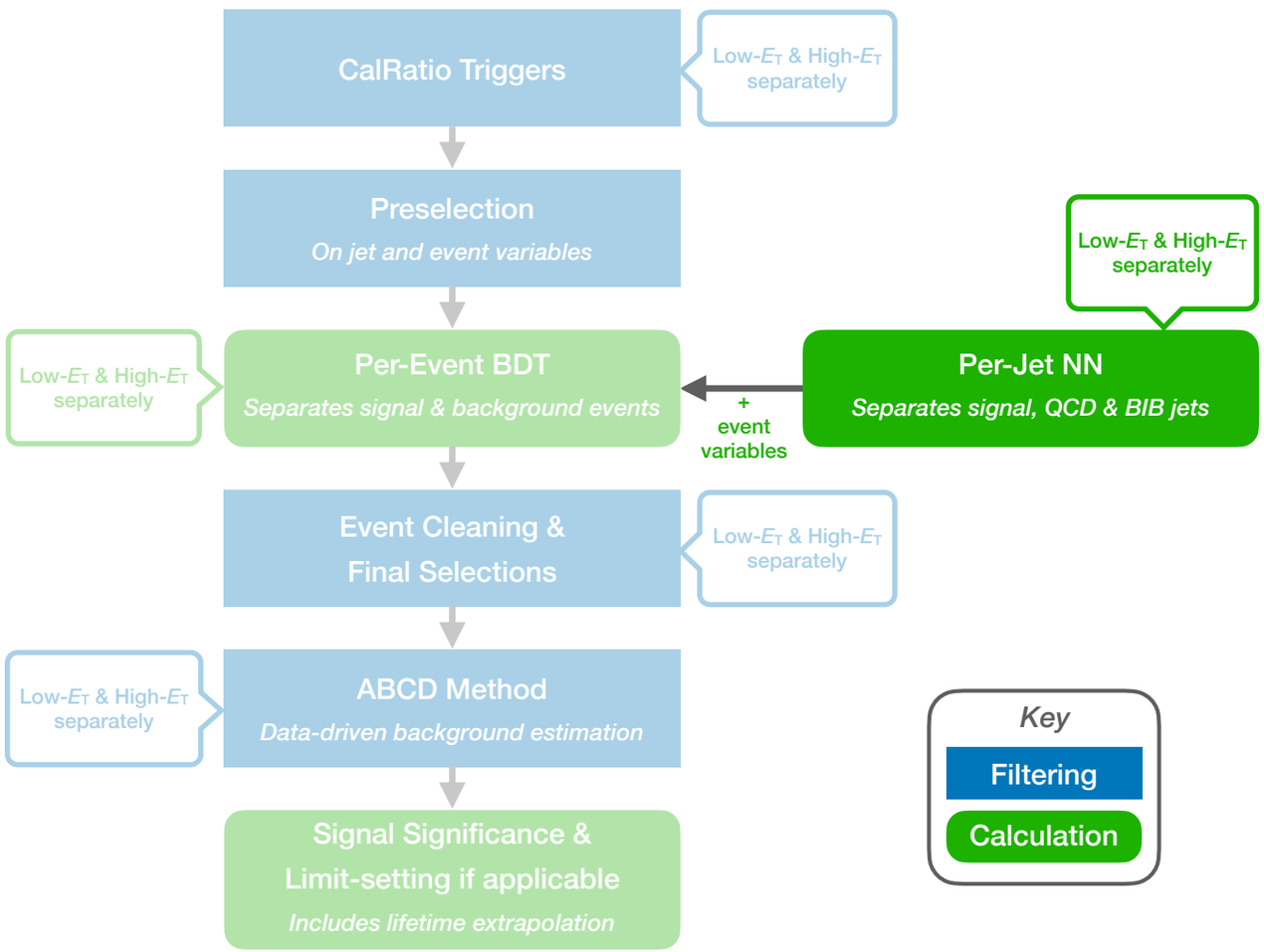



# Signal & Background

- **Signal objects** are narrow jets with no collinear tracks, depositing **majority of energy in HCAL**
- Main Backgrounds are
  - **QCD multi jets**: low probability to mimic signal, but are **dominant background** due to high cross-section
  - **Beam-induced background** (BIB): muons from bunch interacting with beam pipe/gas undergoing **bremsstrahlung in the HCAL**



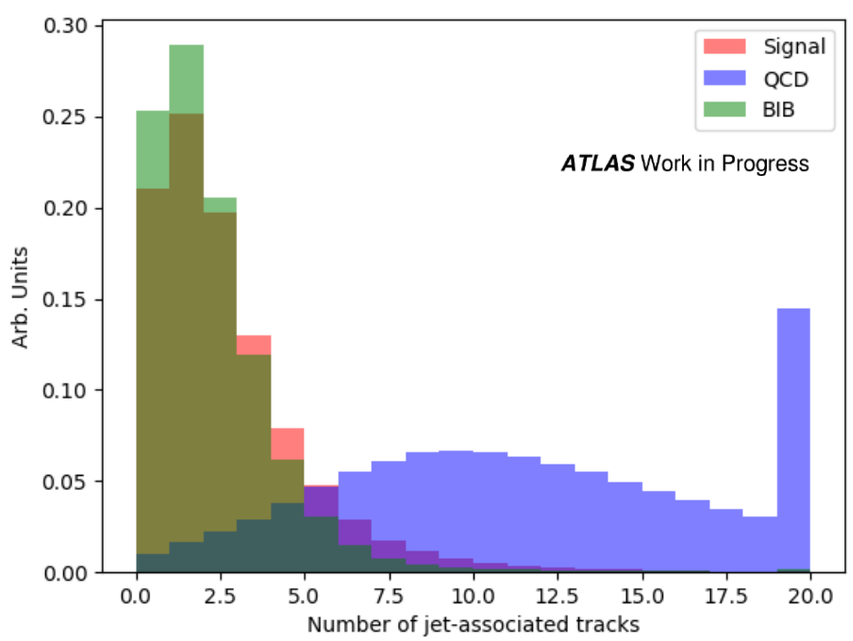
# Per-Jet Neural Network

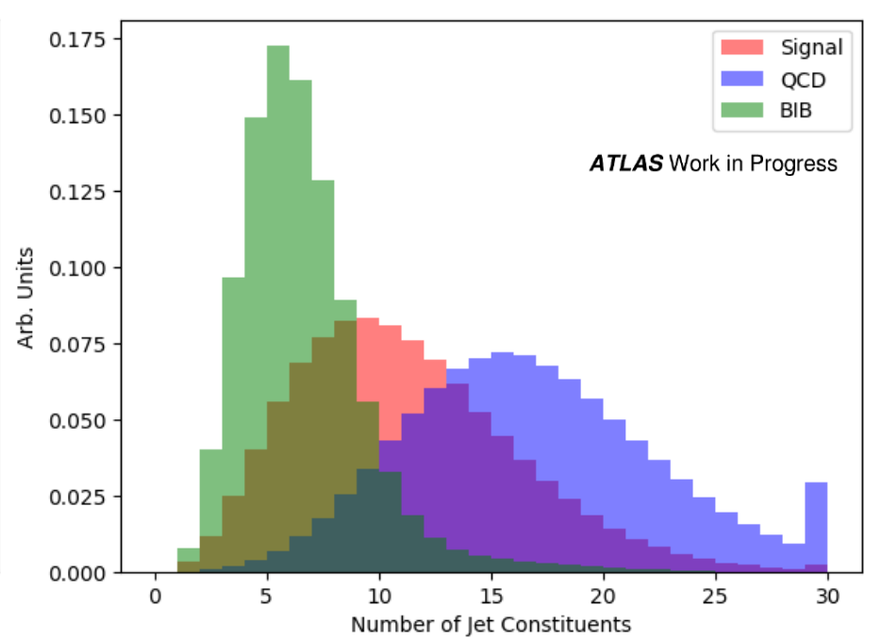


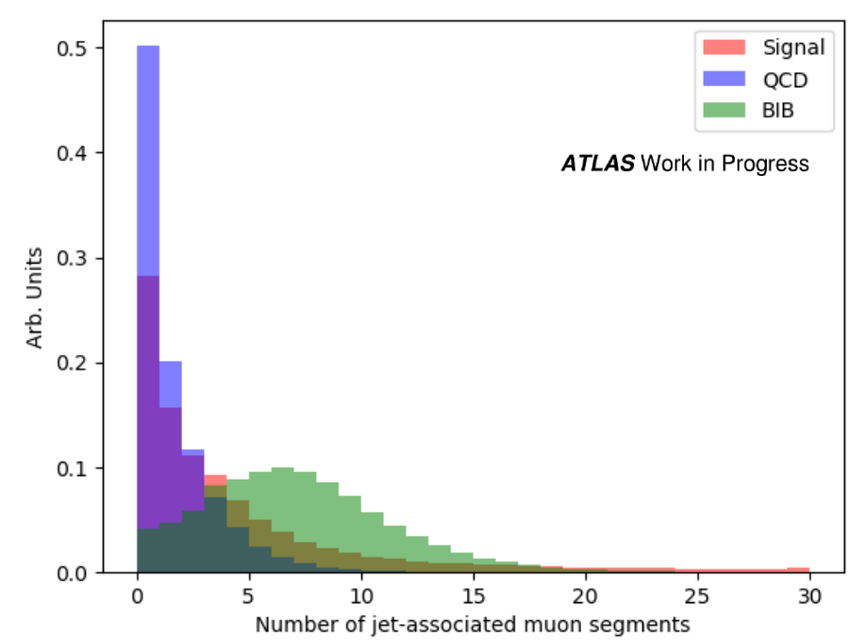

- Focus of this talk will be on the per-Jet Neural Network
  - Network **tags a jet** as signal, QCD or BIB
  - Trained on ~1 million jets of each class
  - Signal, QCD taken from **simulation**
  - BIB taken from special **data** stream

# Per-Jet NN variables

- This analysis can take advantage of all three ATLAS sub-detectors, so for each jet, use the following
- **Tracker:** Takes all tracks within jet axis
  - A signal or BIB jet should have no tracks associated - any would be a result of pileup
- **Calorimeter:** Take all topoclusters (jet constituents) used to create the jet
  - Unusual signal jet shape can be learned to tag as signal
- **Muon Spectrometer:** Take all Muon Segments within jet axis
  - Muon segments can indicate BIB generating a jet, or punch-through from signal decay late in the Hadronic calorimeter

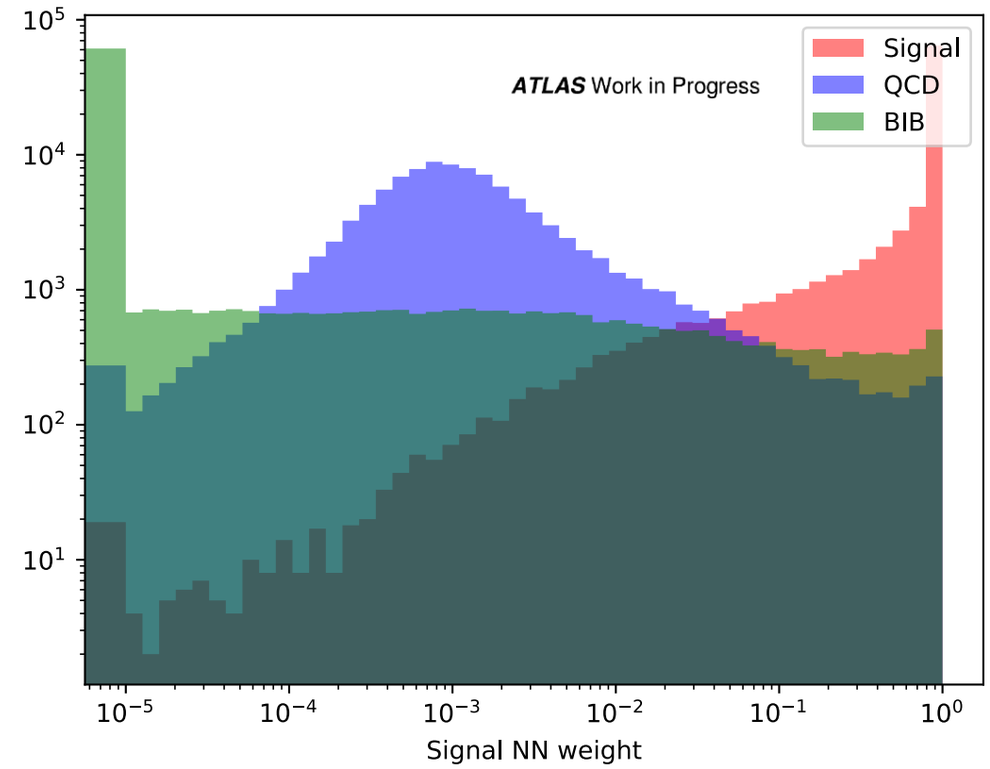


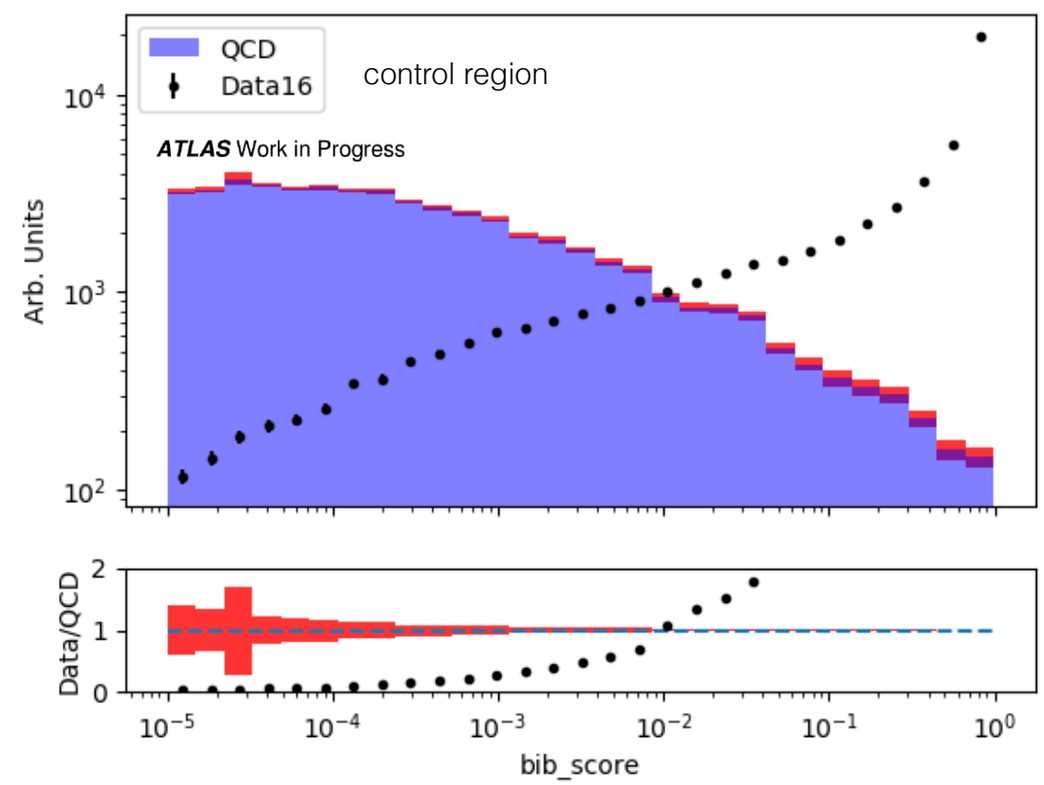



# Control Region NN output

- Training is run using keras with tensorflow backend
  - **Hyper-parameter scan** performed
  - Dropout, Regularization, Early stopping used to **reduce over-training**
- Very good performance seen in training (top right)
- **Control region** used to test if performance is same in data and simulation
  - **Dijet selection** used to create control region
  - Can see that QCD simulation and Data have different bib outputs: NN has **learned simulation/data differences!**



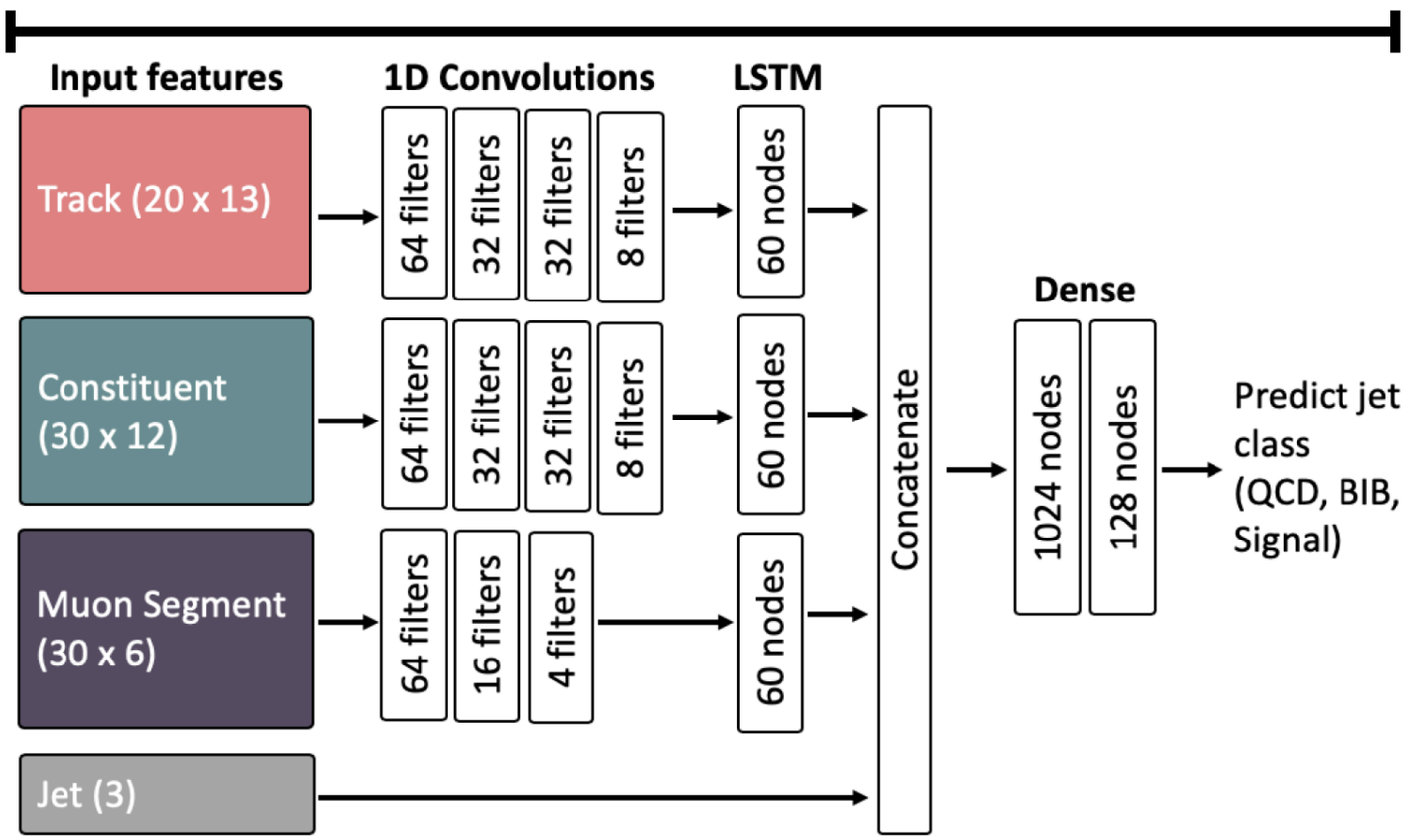



# Adversarial Training

- To reduce the dependence of the training on simulation/Data we propose to **add an adversary** to the training
- This training comprises of
  - ‘Original’ Signal vs. Background supervised training
  - ‘Adversary’ which uses the output of the ‘original’ network to guess if jets from control region are MC or Data
- The network is designed to try to **‘fool’ the adversary** by making the multiclass [Signal, QCD, BIB] weights not be different due to simulation/Data differences
- In this case, we would be making sure the network ignores simulation/Data differences in the input variables

# Adversary Architecture

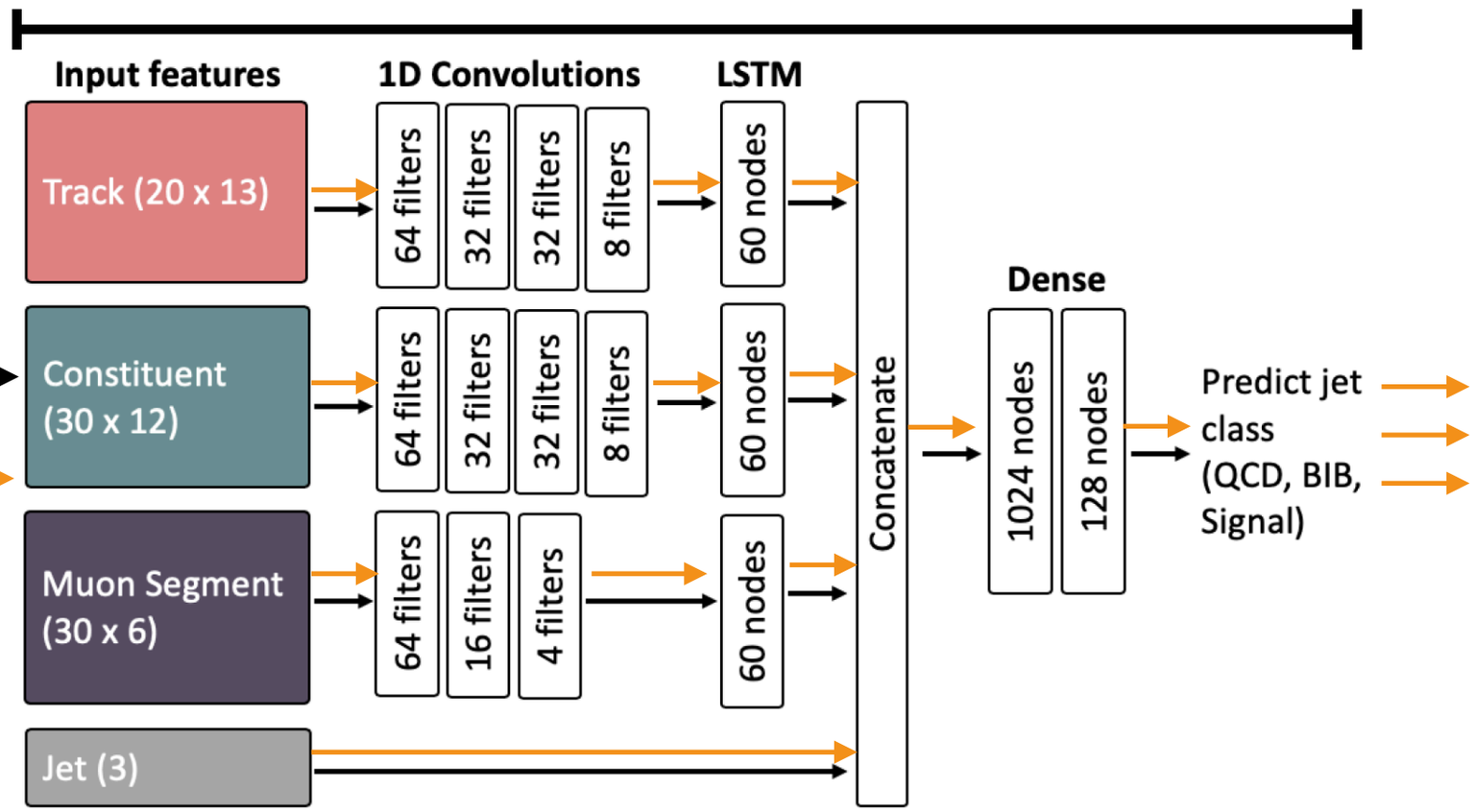
1. Main Jet NN



# Adversary Architecture

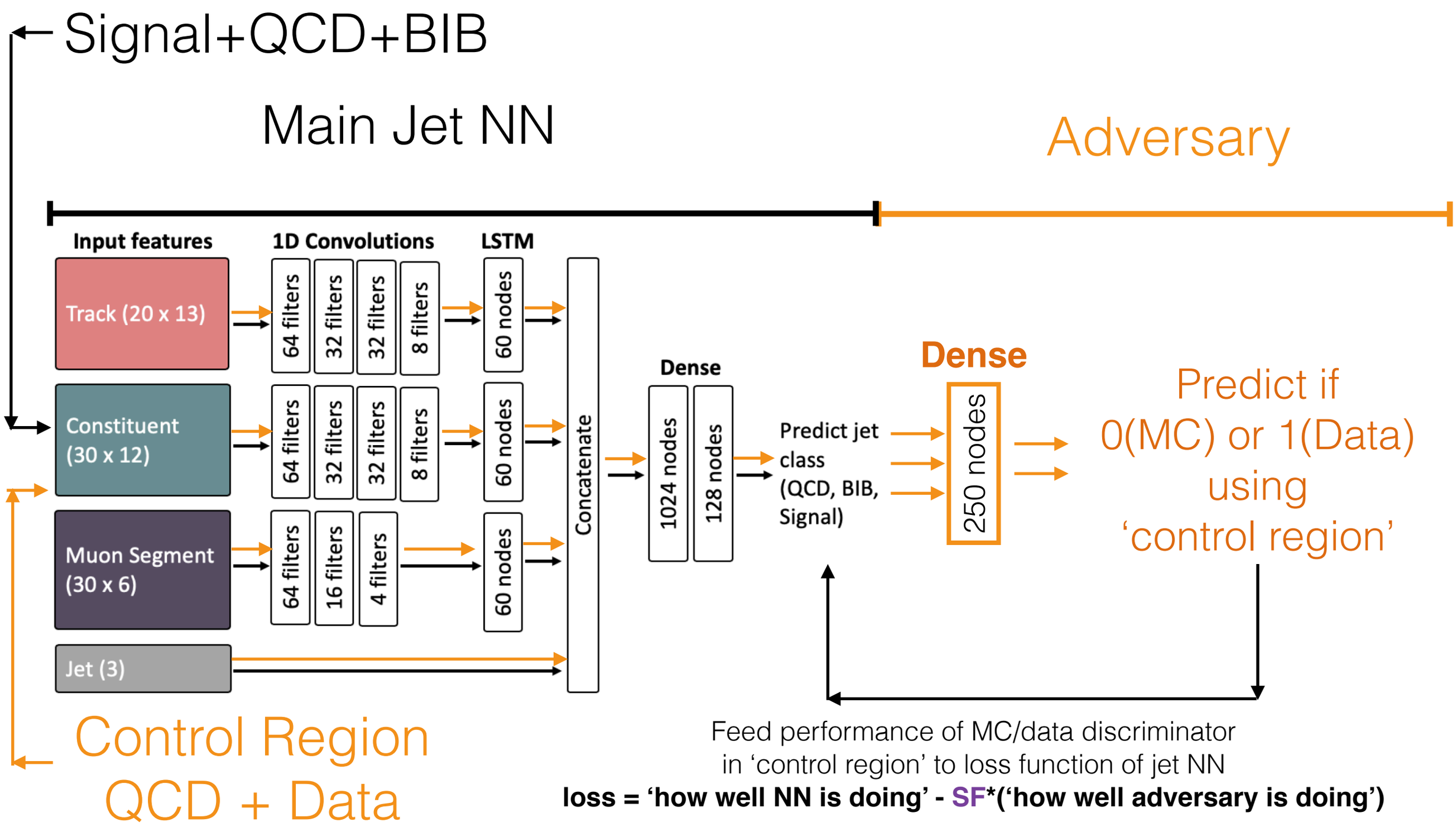
← 1. Signal/QCD/BIB

Main Jet NN




← 2. Put in QCD and Data from dijet event 'control region'

# Adversary Architecture




Feed performance of MC/data discriminator in 'control region' to loss function of jet NN

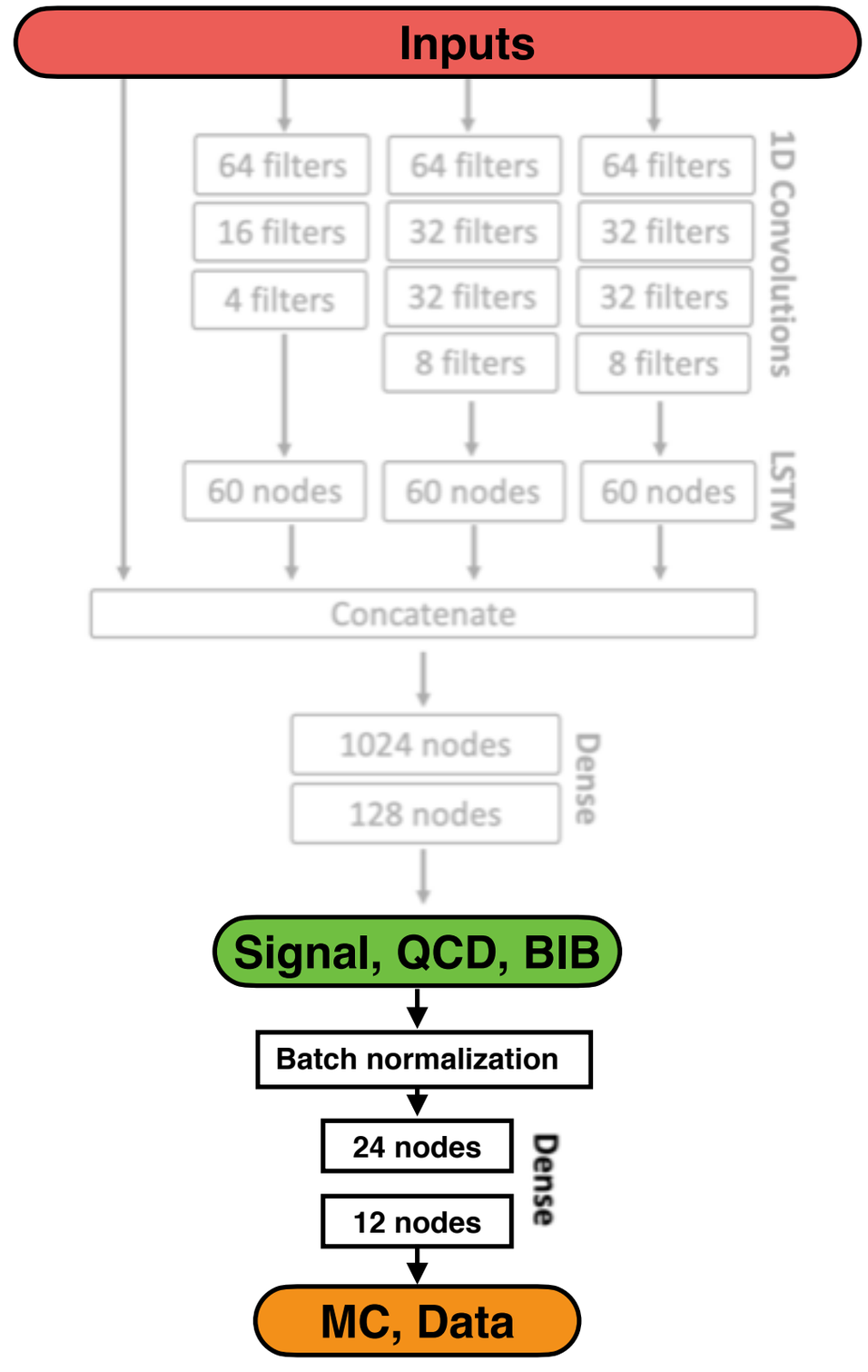
**loss = 'how well NN is doing' - SF*('how well adversary is doing')**

# Adversary Architecture

- How to actually do this in keras?
  - Two models with shared layers:
    - **main network** has as input [QCD,Signal,BIB], output is prediction [QCD,Signal,BIB]
    - **discriminator network** has as input [QCD, Data], output is prediction [QCD,Data]
  - Each mini-batch: first train **discriminator**, then **main** network
    - Due to way keras inputs work, to train main network both inputs have to be same length. So we duplicate some control region inputs to make sure they are the same lengths
  - Loss function for main network is
    $loss = CE_m(y_true_m, y_prediction_m) - SF \times BE_d(y_true_d, y_prediction_d)$
    - CE: categorical cross-entropy
    - BE: binary cross-entropy
    - m: main network
    - d: discriminator network
    - SF: adversary Scale Factor
    - Scale factor hard to find: too high and network learns that it is good to predict [QCD,Data] **wrong**, too low and it doesn't have an effect

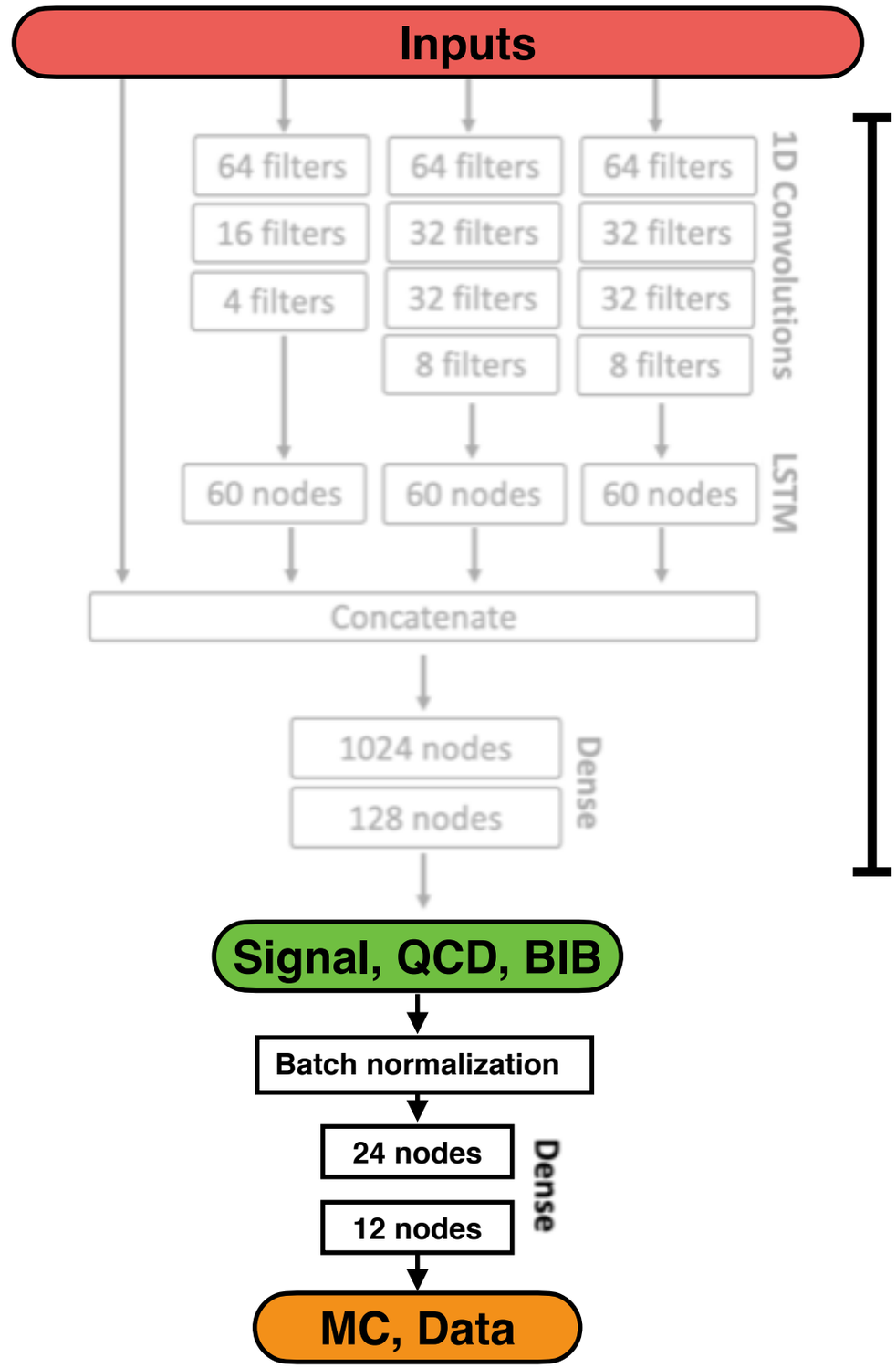
1. Adversary step

QCD, Data from control region



1. Adversary step

QCD, Data from control region

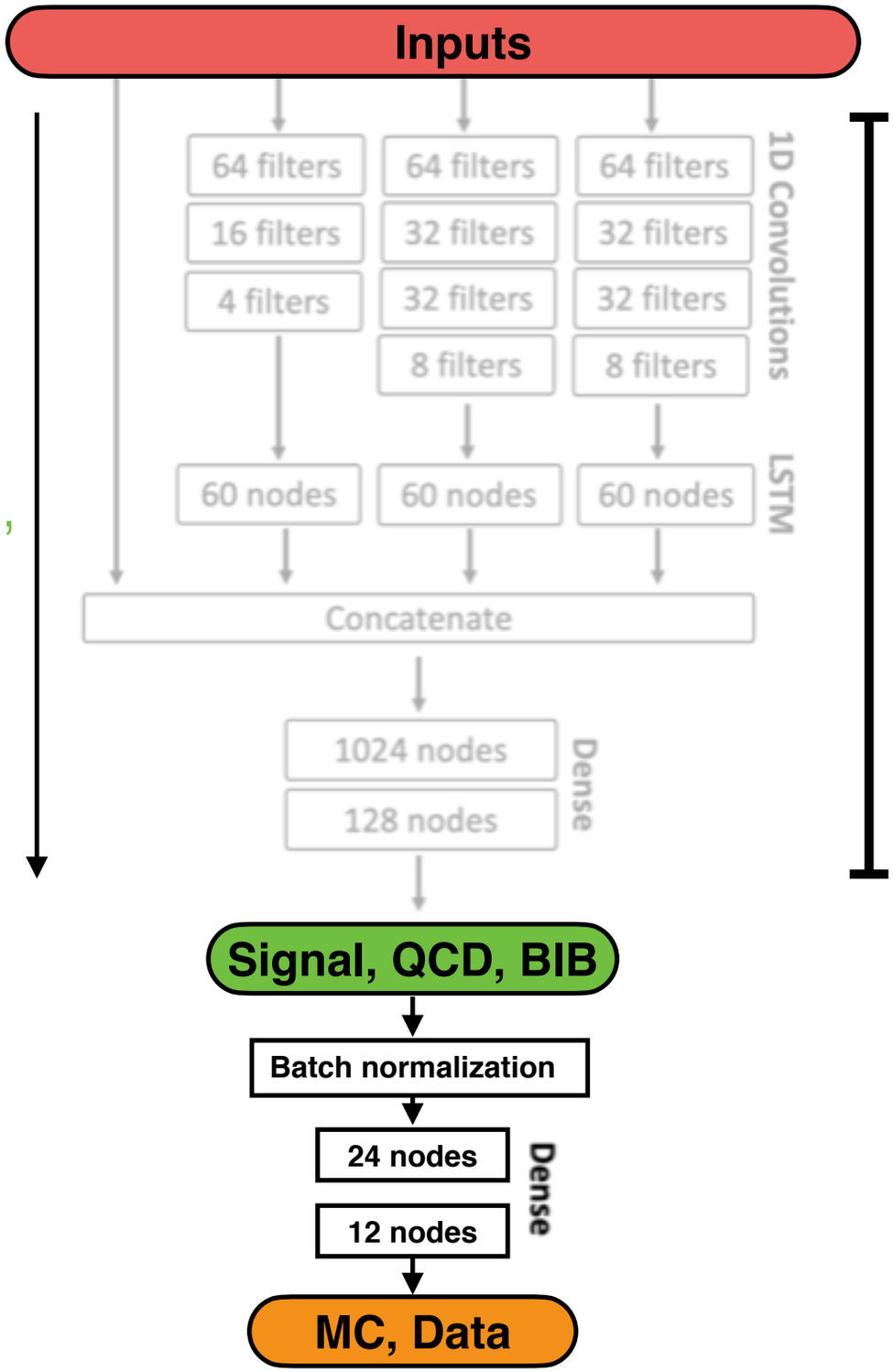



Not trainable

1. Adversary step

QCD, Data from control region

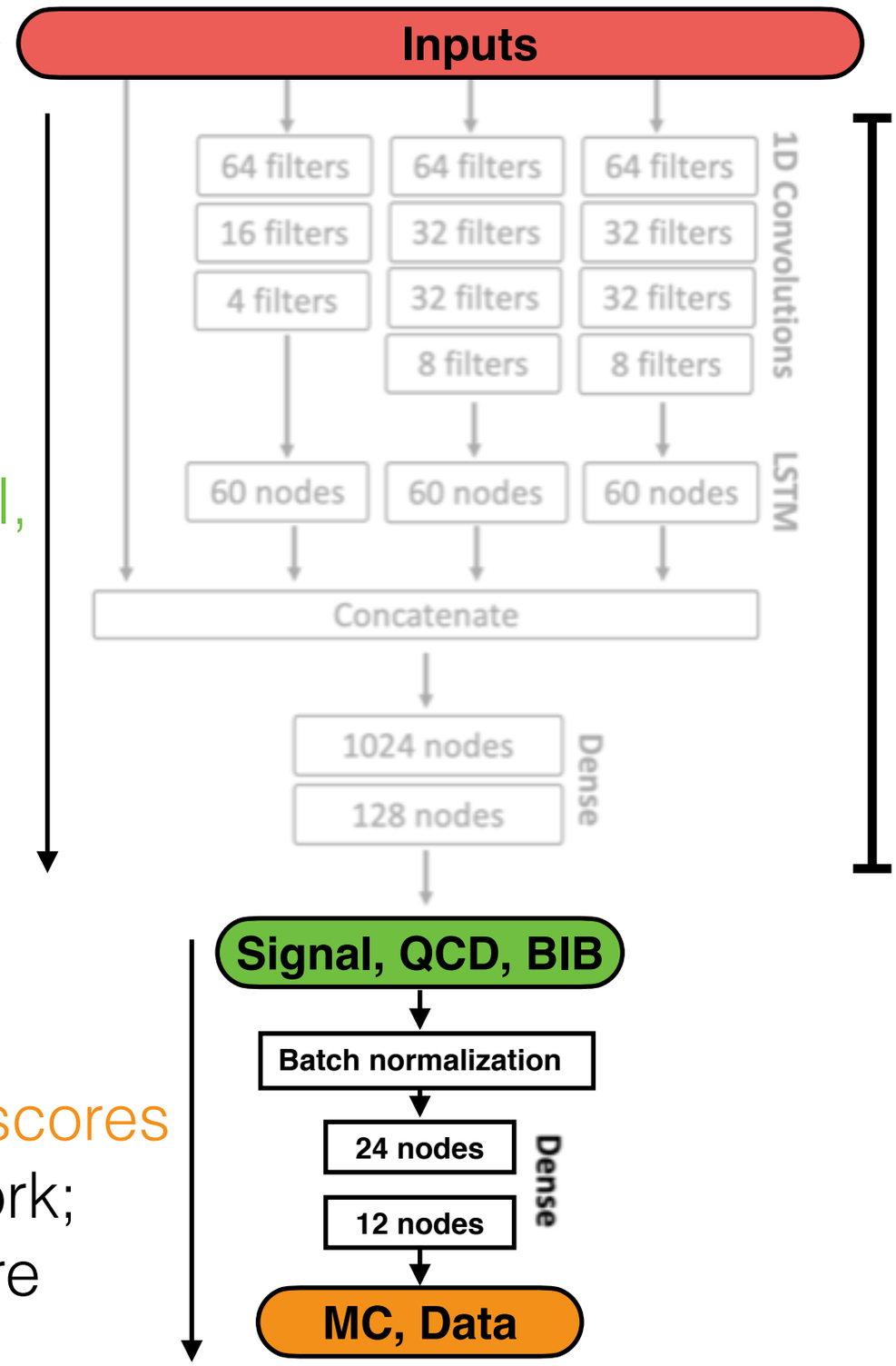
Calculates Signal, QCD, BIB scores for QCD, Data in control region




Not trainable

1. Adversary step

QCD, Data from control region



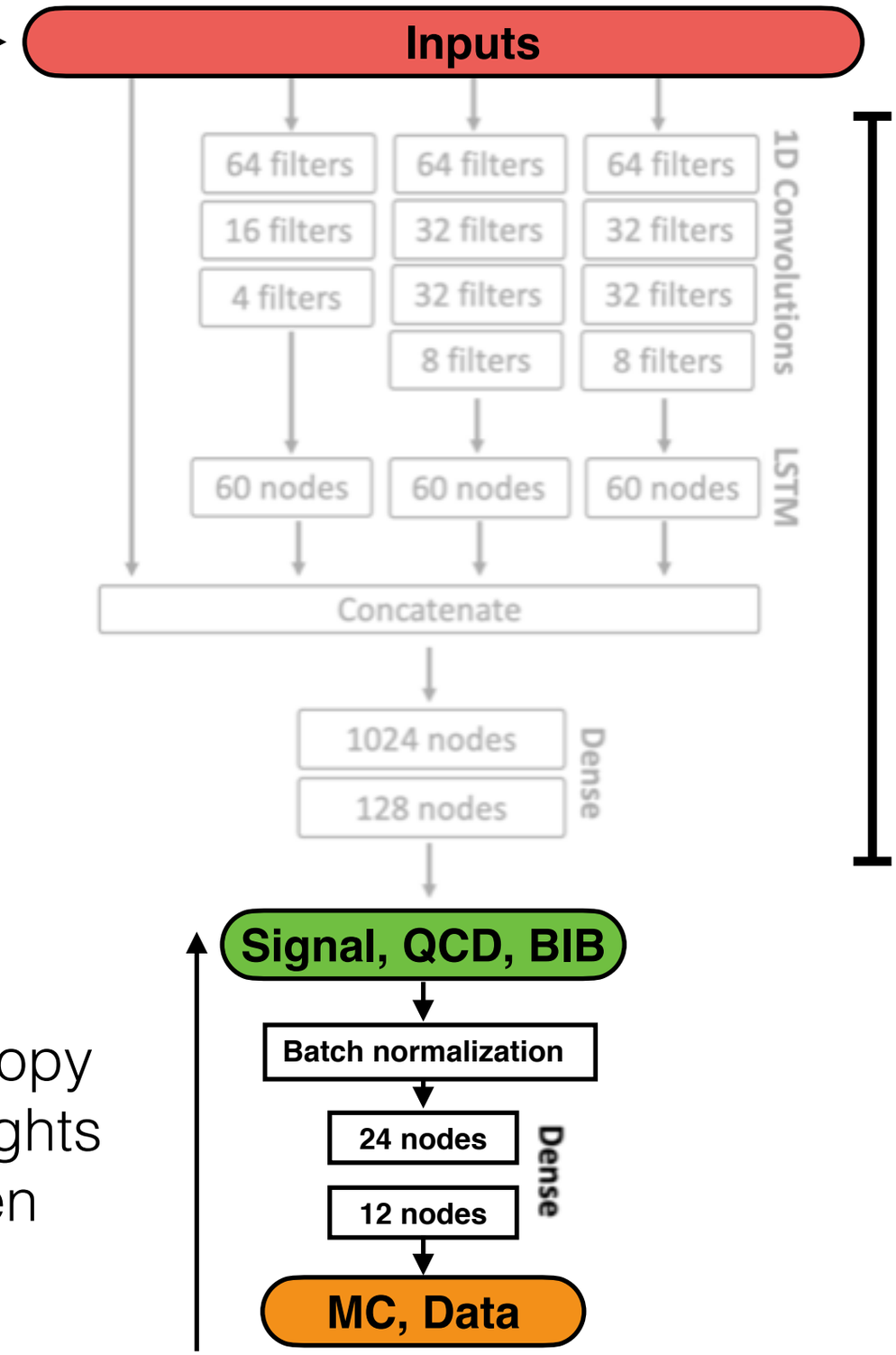

Not trainable

Calculates Signal, QCD, BIB scores for QCD, Data in control region

Calculates MC, Data scores in adversary network; trains weights here

1. Adversary step

QCD, Data from control region



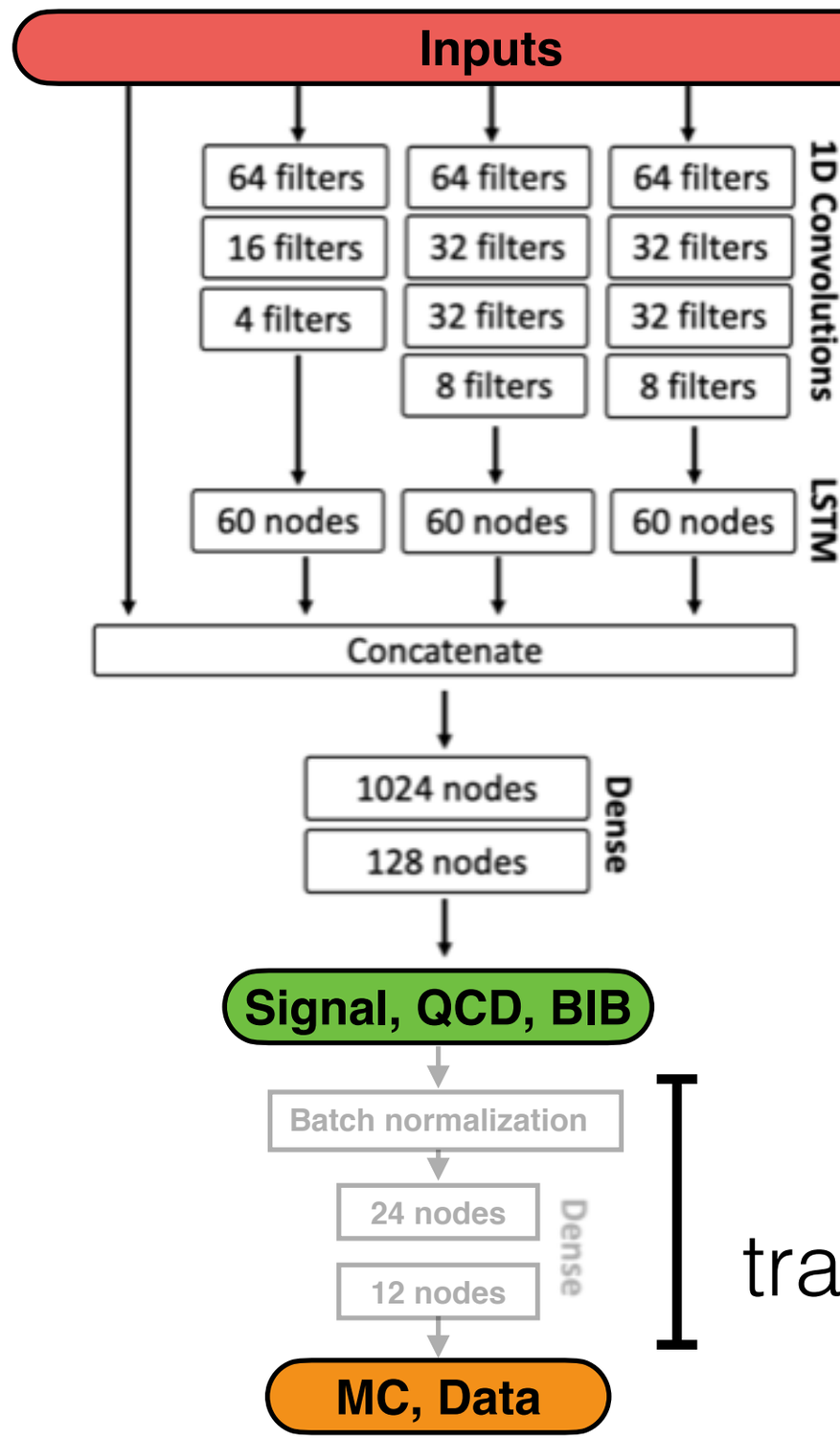

Not trainable

Backpropagate **positive loss** of MC, Data cross entropy such that trainable weights **differentiate** between QCD, Data

2. Main step

QCD, Data from control region

Signal, QCD, BIB

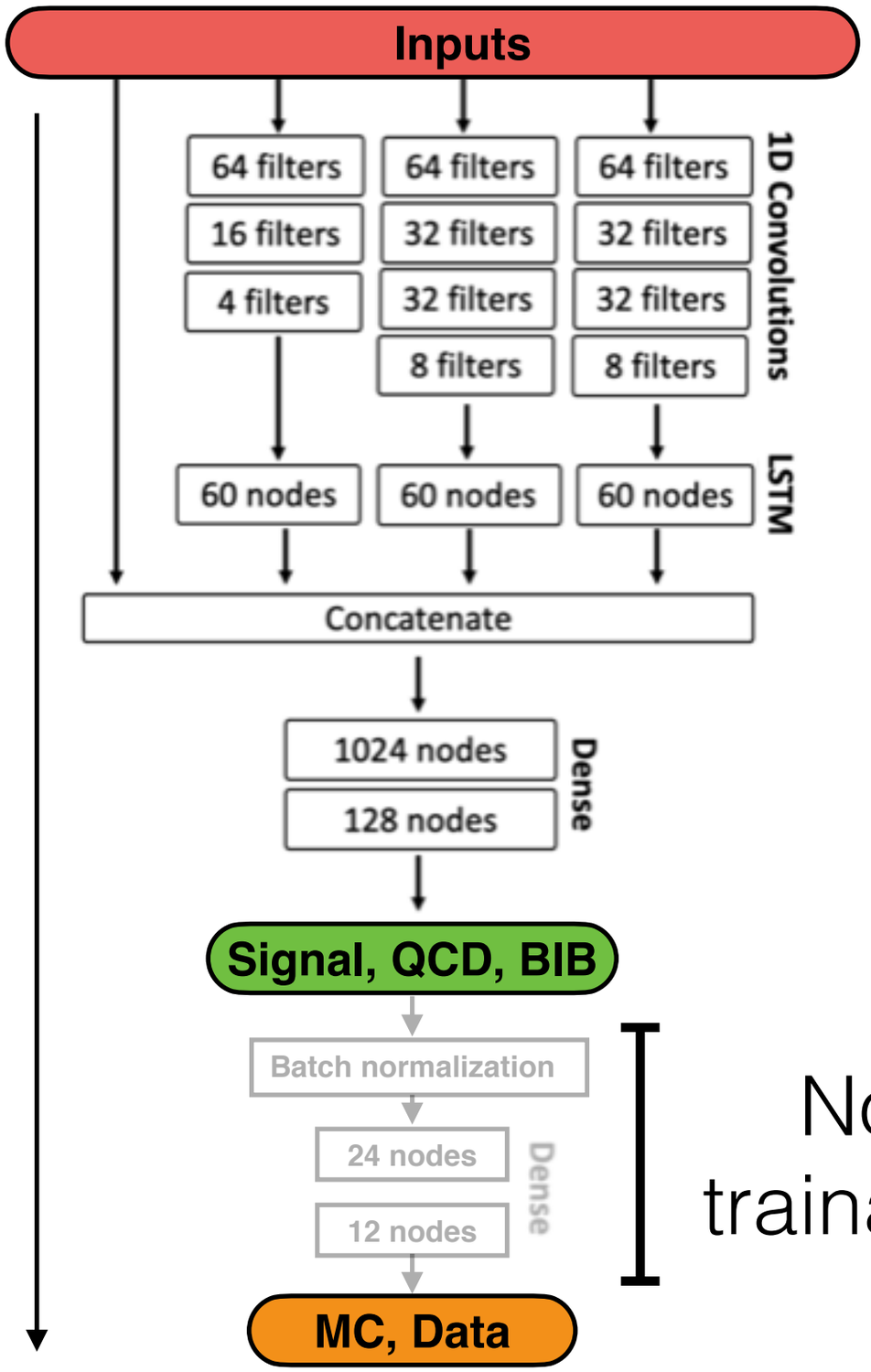



Not trainable

2. Main step

QCD, Data from control region

Signal, QCD, BIB




Calculates Signal, QCD, BIB scores for Signal, QCD, BIB inputs

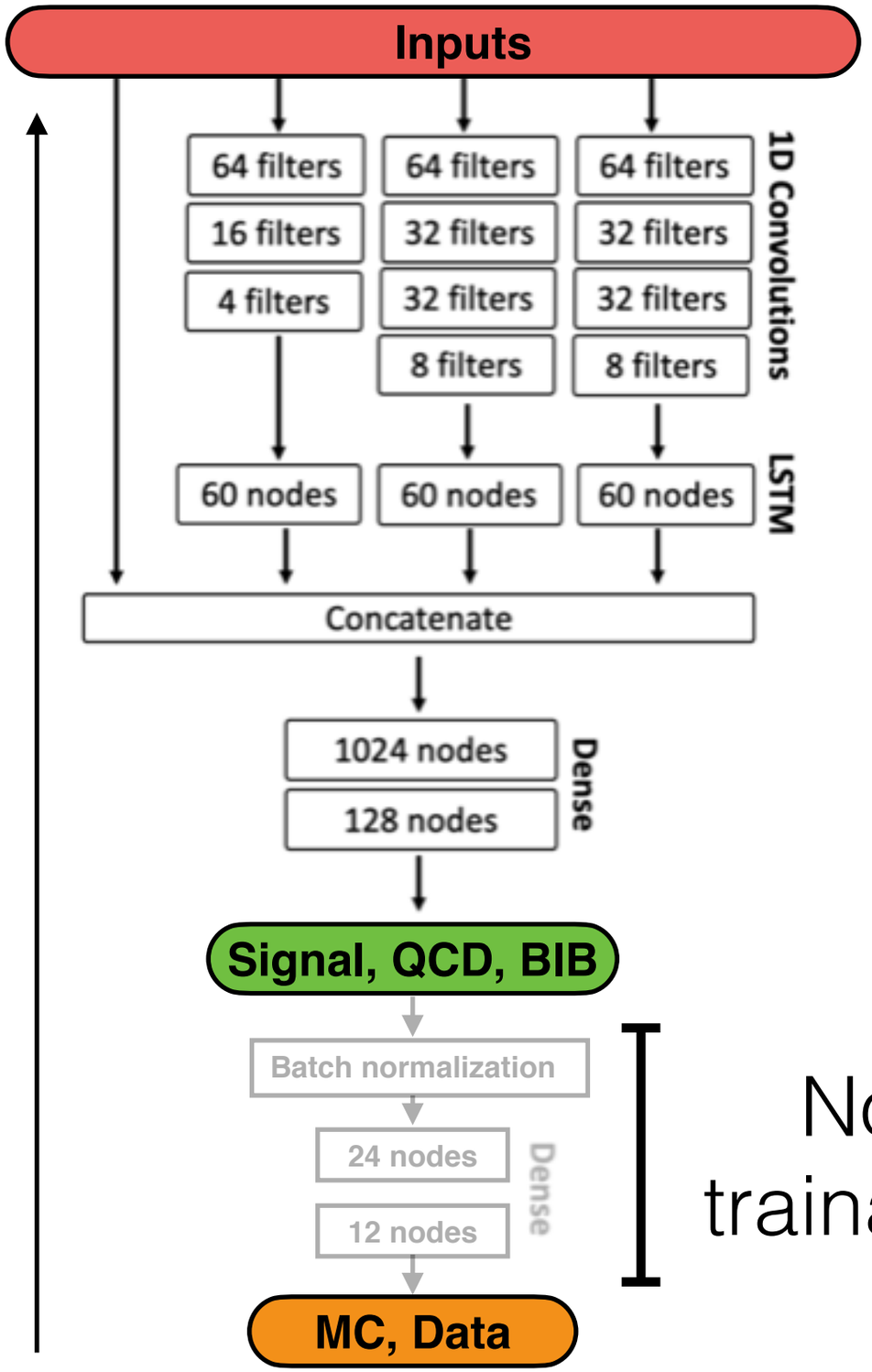
Calculates MC, Data scores for QCD, Data in control region

Not trainable

2. Main step

QCD, Data from control region

Signal, QCD, BIB



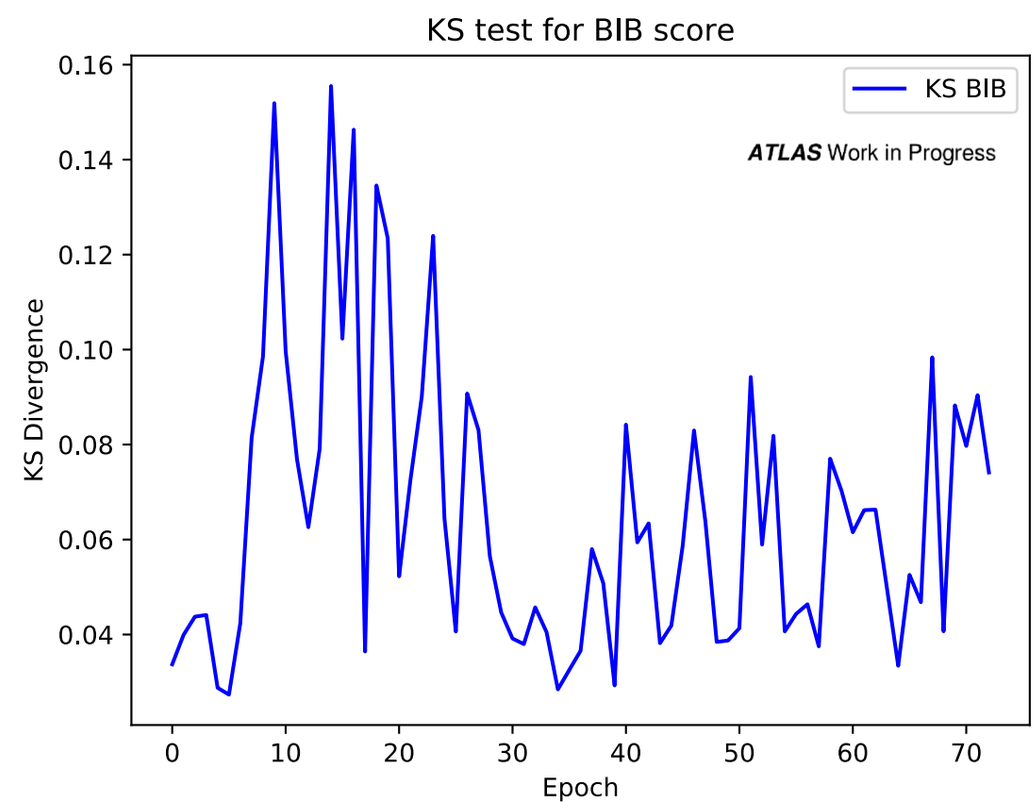

Backpropagate **negative loss** of MC, Data cross entropy such that trainable weights **do not differentiate** between QCD, Data

Backpropagate **positive loss** of Signal, QCD, BIB cross entropy such that trainable weights **differentiate** between Signal, QCD, BIB

Not trainable

# KS Test as a metric

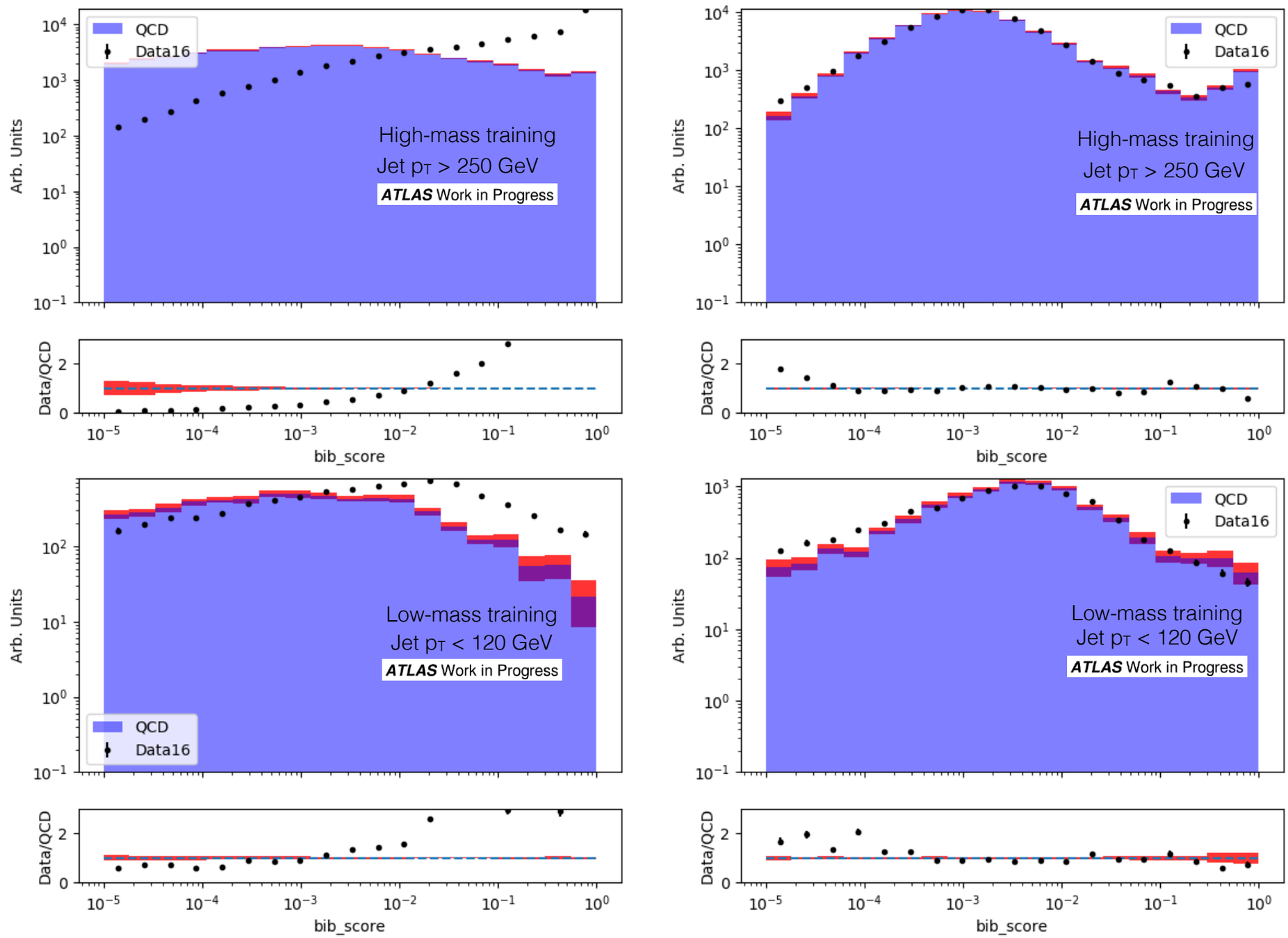
- Post-training, used the **Kolmogorov Smirnoff Test** on the output scores to select which adversarial training to keep
  - Lowest KS Divergence between Simulation and Data means **smaller difference in output score**
  - You can see the ‘tug of war’ between the signal/QCD/BIB network and the adversary
  - In the end, use training epochs with **lowest KS divergence**, **best agreement** at high scores and **best performance** at tagging jets



# Adversary Results

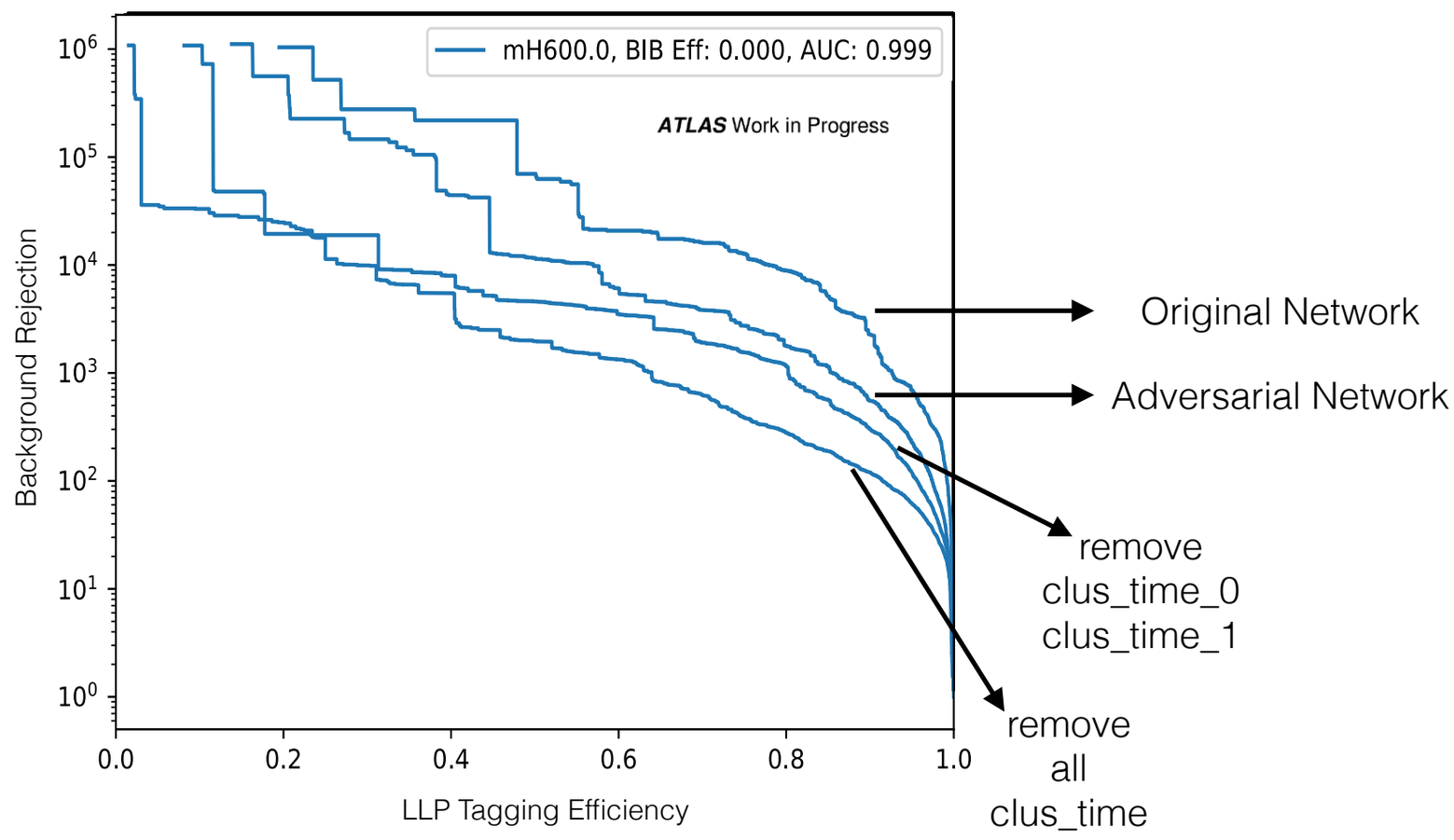
- After training with adversary, choosing best performing epoch, compare the **results of the adversary in different $p_T$ ranges**
- Can see huge improvement in **shape agreement**
- Small discrepancies at low bib_score, but search uses **high bib_score to reduce BIB**

Add adversary



# Adversarial NN performance

- Can also look at the **ROC curve** of the adversary
- Removing all or part of cluster time from training has lowest performance
  - This was considered as cluster time is variable with worse performance
- Adding adversary gives you performance between removing time and original network
  - However, **adversary gives best MC/Data agreement** in control region
  - Can reasonably conclude that the increase in performance between Adversarial and Original network is **due to learning MC/Data**, not Signal/QCD/BIB differences

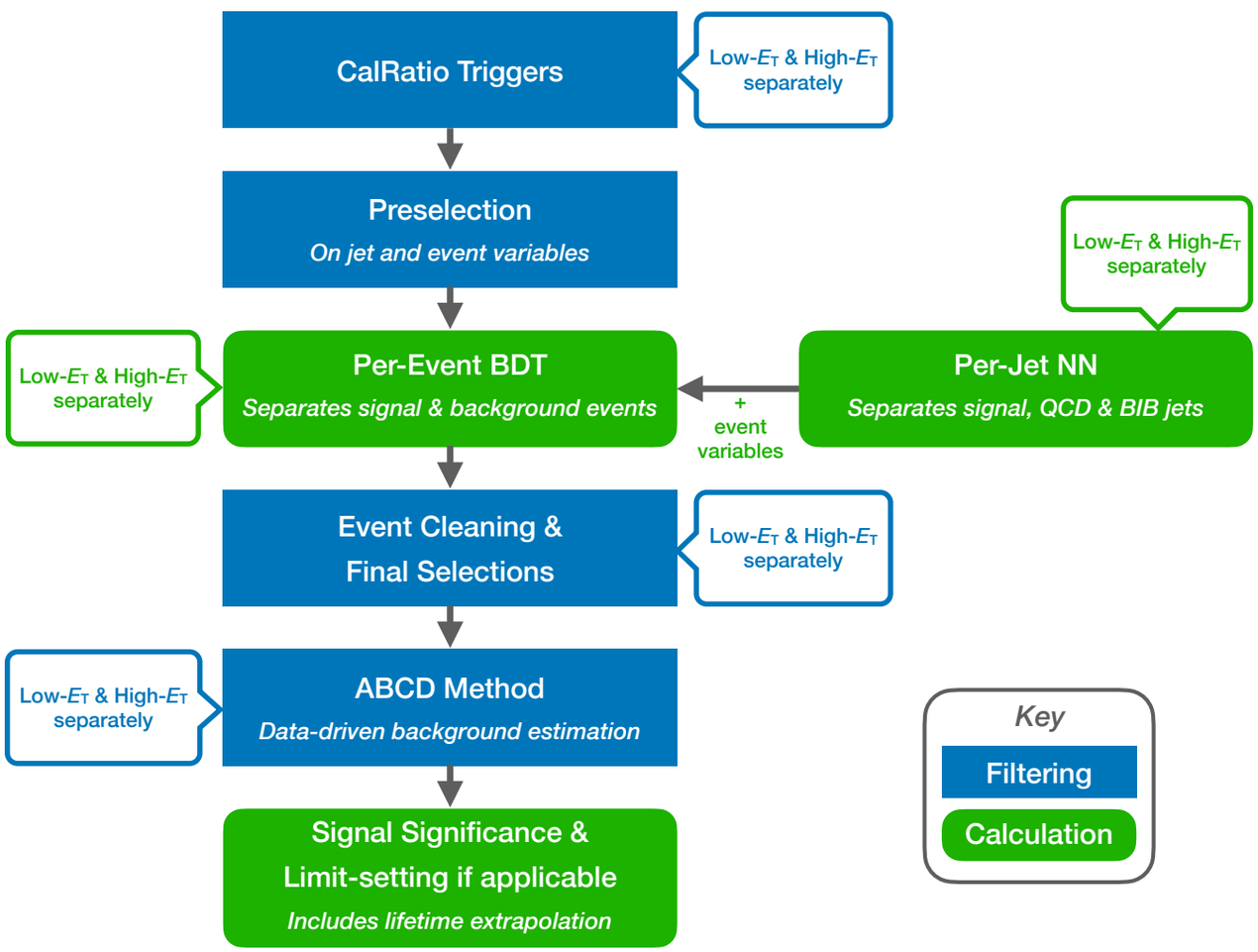


# Conclusions

- Simulation/Data differences were found in the output of our LLP jet tagging Neural Network
- An adversarial network was achieved in Keras using shared layers between two separate networks set up to solve this issue
- By monitoring both jet tagging and simulation/data performances each epoch a reasonable stopping point was chosen
- The results show much better simulation/data agreement

# Backup

# Analysis flowchart




- This analysis uses data from the **full Run 2** of the ATLAS detector
- Specialized **'CalRatio' Triggers** look for jets with high Hadronic Calorimeter energy ratio
- **NN and BDT** are used as the main discriminants in the analysis
- **Data-driven** ABCD method used for background estimation

# Adversary optimization

- Main roadblock in optimizing the adversarial training is the adversary loss going the 'wrong way'
  - Main network minimizing
    $loss = CE_m(y_true_m, y_prediction_m) - SF \times BE_d(y_true_d, y_prediction_d)$
  - Negative loss term: minimizes when it is **large**
    - It is possible for the network to minimize it by being **very wrong** (large loss)
    - But, being very wrong in this context is equal to being very right (back to a large MC/Data difference)
- Looked for a little bit of help in the GAN ML community to reduce the loss going the 'wrong way'
  - Switch from main to adversary training every mini-batch instead of every epoch
  - Spectral Normalization keeps the adversary weights from changing too fast
  - Two-scale update for main and adversary means keeping learning rate low for main mini-batch, but high for adversary mini-batch
  - Batch Normalization helps adversary take into account the many orders of magnitude of Signal, QCD, BIB scores

# Per-Jet NN Input samples

- **Signal:** Take all samples with $\Phi$, $s$ mass combinations listed previously
  - Keep half the events for training/validation
  - Only keep jets associated with an LLP which decays in the Hadronic Calorimeter
- **QCD:** JZ[2-4]W multijet samples used
  - Represents events with jet $p_T$ from 60 to 800 GeV
- **BIB:** Events obtained from data are ones failing the BIB removal algorithm at HLT
  - Then keep only jets matched to the HLT jet which fired the CalRatio trigger
    - This eliminates QCD combination in BIB samples
- **Common cuts:**
  - Jet $p_T > 40$ GeV, $|\eta| < 2.5$, *isCleanLooseBadLLP* flag
- In the end, get about 1 million jets of each class

# Per-Jet NN variables

## Track Variables

- $p_T$
- $\eta$
- $\phi$
- vertex_nParticles
- $d_0$
- $z_0$
- chiSquared
- SCTHits
- SCTHoles
- SCTShared

## Topocluster variables

- $p_T$
- $\eta$
- $\phi$
- l1hcal
- l2hcal
- l3hcal
- l4hcal
- l1ecal
- l2ecal
- l3ecal
- l4ecal
- time

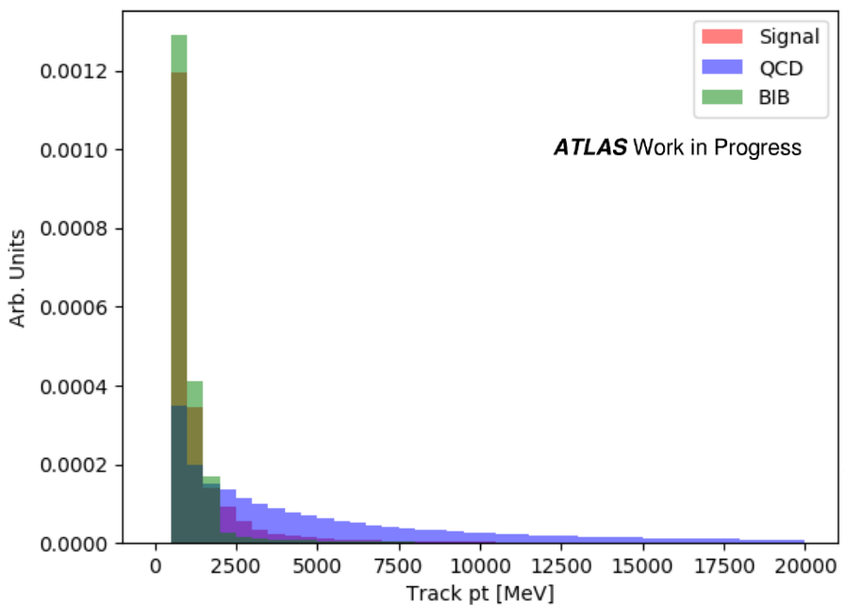
## Muon Segment variables

- $\eta$ position
- $\phi$ position
- $\eta$ direction
- $\phi$ direction
- chiSquared
- t0

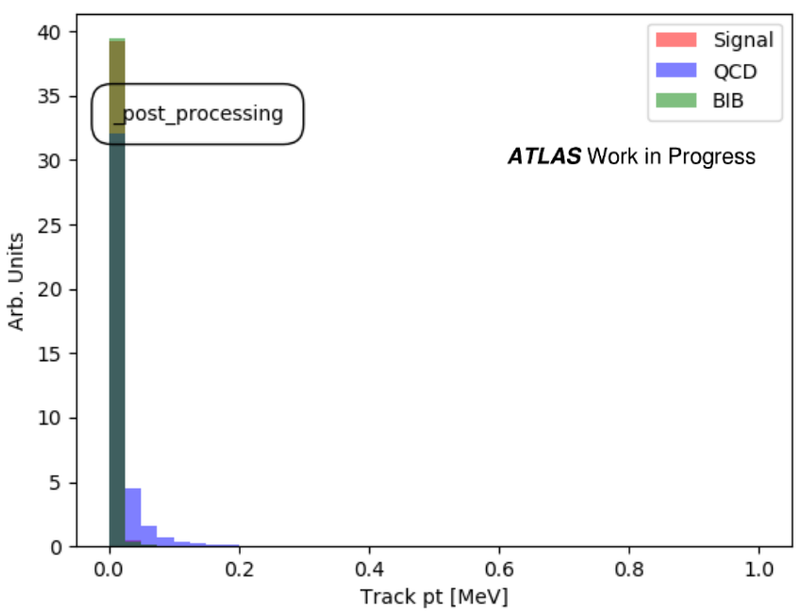
## Jet variables

- $p_T$
- $\eta$
- $\phi$

# Per-Jet NN Pre-Processing



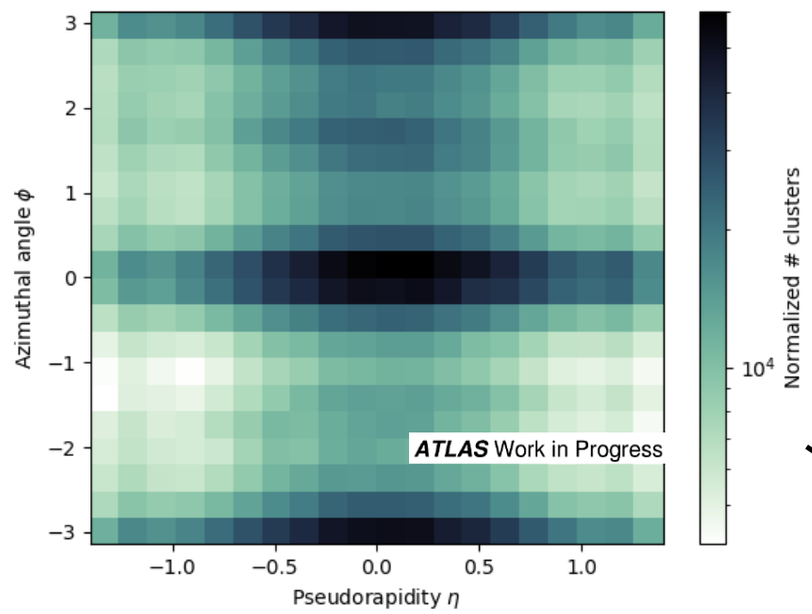

pre-processing Applied



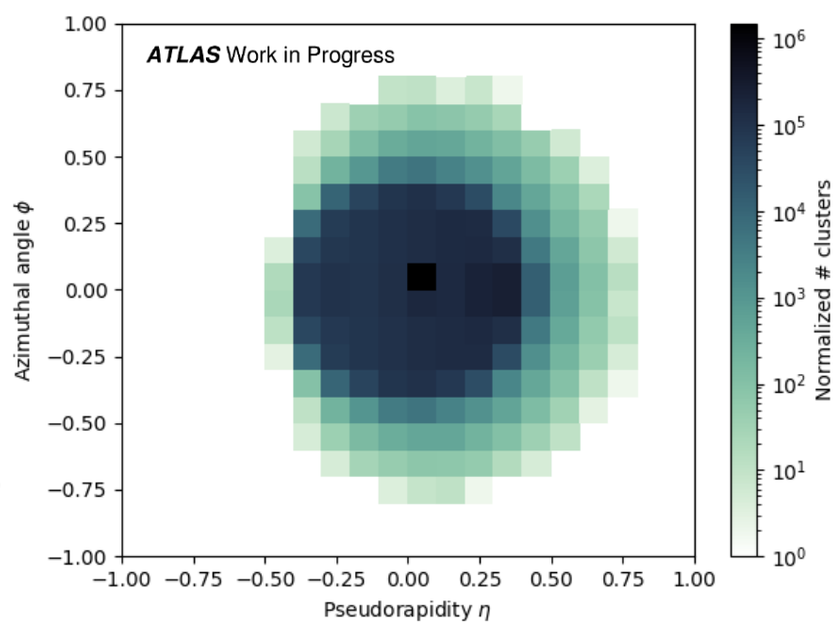

- One example of pre-processing of variables is done to track $p_T$
  - Original variable has large range of values
  - **Divide by max jet $p_T$** to get values between 0 and 1
  - This method has been shown to **speed up convergence during training**

# Per-Jet NN Pre-Processing

- For jet constituent angles, compare to highest energy constituent
  - **subtract angle** of highest energy constituent from all others
  - Do **η, $\phi$ flip** dependent on energy of second highest
  - This means the ML algorithm will learn a **consistent set of jets**

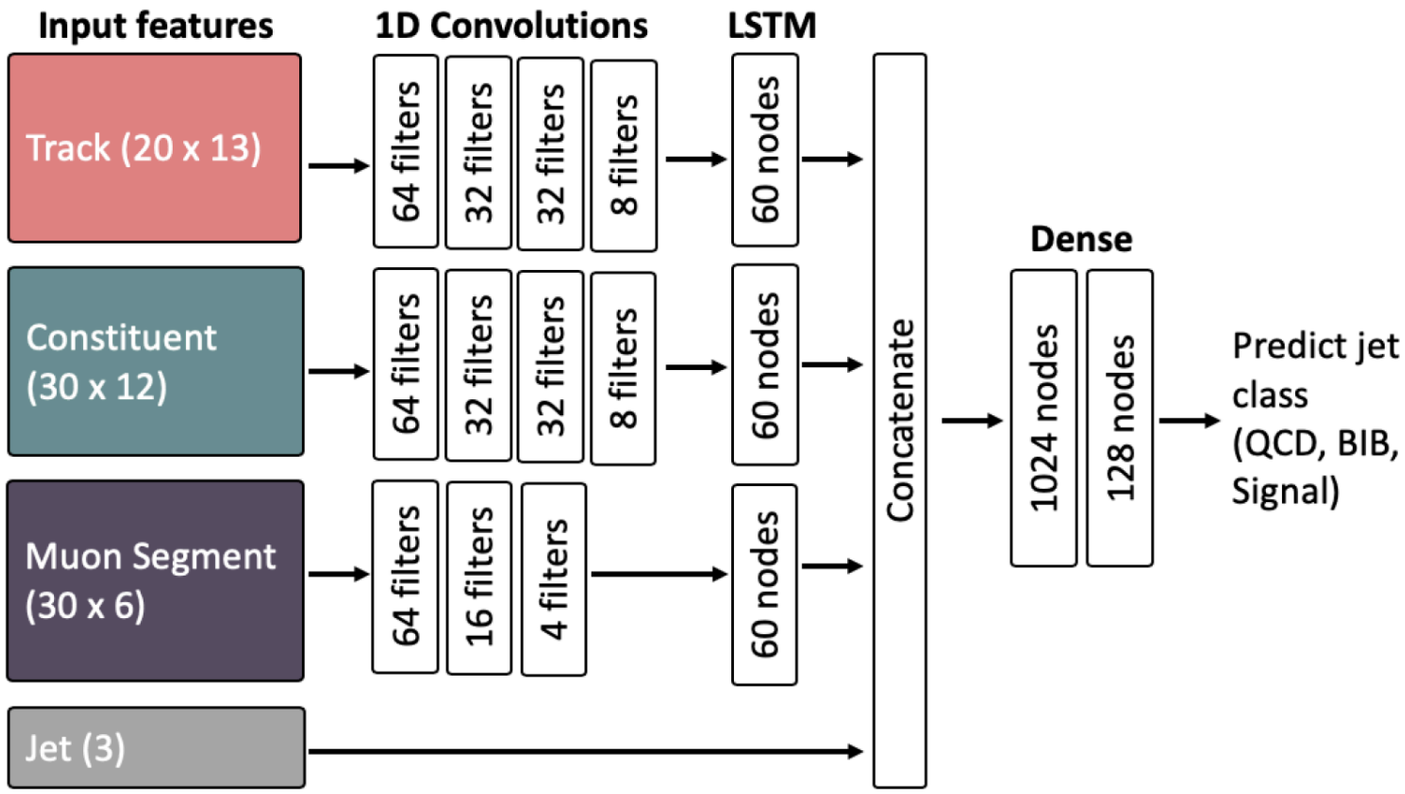



pre-processing Applied



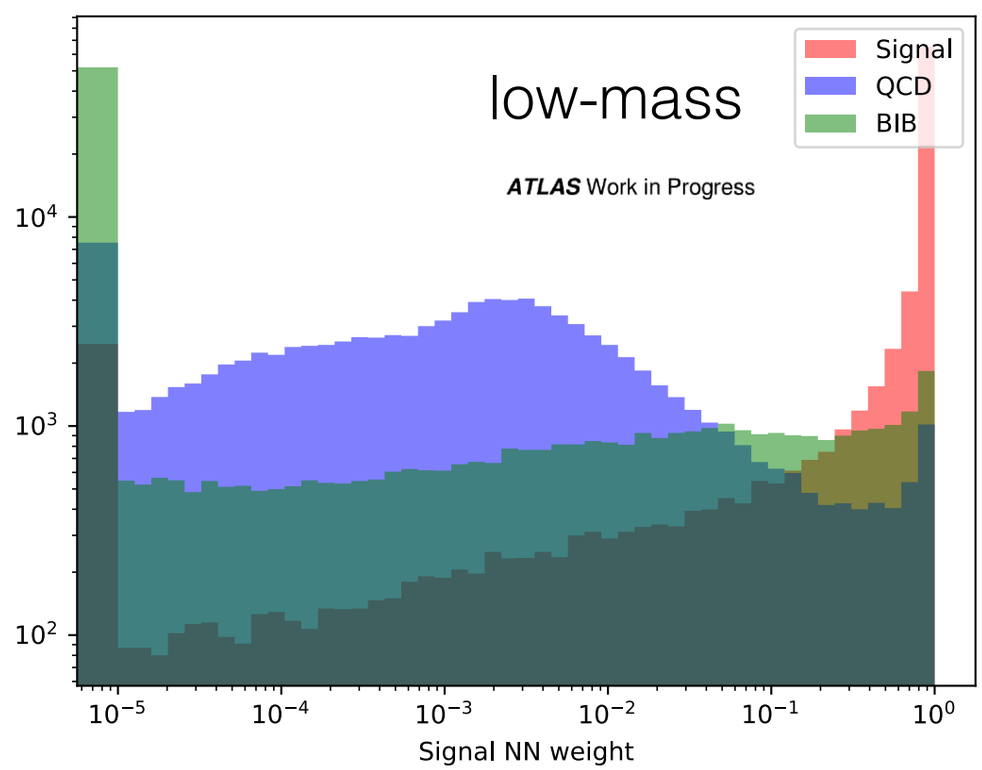
# Per-Jet NN Architecture

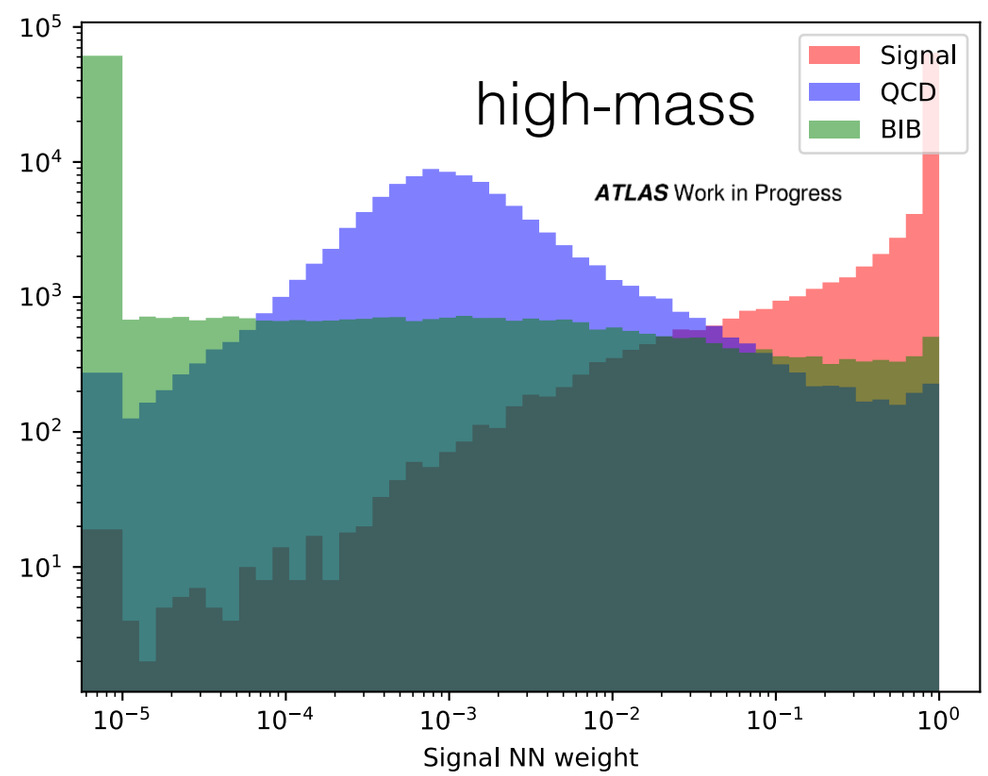
- The job of the per-jet NN is to classify a jet as Signal, QCD or BIB
- The NN is setup and trained using **Keras with Tensorflow backend**
  - 1D convolutions were added after the input
  - 1D Convolutions are excellent at **finding useful correlations between input variables**
  - LSTMs are the next layers; they are excellent at **finding patterns between subsequent training data** in ordered inputs (e.g. between different topoclusters)
  - These are all concatenated to a set of full-connected layers
  - Output are 3 weights (or scores): QCD, BIB, Signal



# Per-Jet NN Result

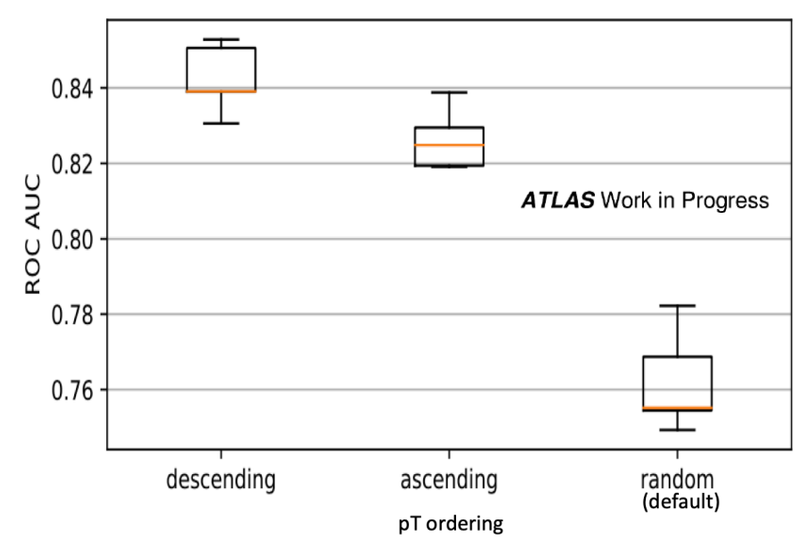
- After finalizing input samples, variables, pre-processing and architecture, a NN parameter scan was performed to optimize the training
- Training was performed with Early stopping, Dropout and Regularization to **reduce over-training**
- Training results on evaluation set shows **clear separation in output Signal Weight** for Signal, QCD, BIB



# Per-Jet NN Architecture

- Early on it was decided to base the Per-Jet NN on an LSTM (Long-Short term memory) architecture
  - LSTMs are excellent at **finding patterns between subsequent training data** in ordered inputs (e.g. between different topoclusters)
- Order chosen was:
  - **Topoclusters:** Descending topocluster $p_T$
  - **Tracks:** Descending track $p_T$
  - No ordering for muon segments
- Studied the ordering by changing the order and passing it through training and evaluation:
  - Results showed ordering greatly improved results, **meaning orderings chosen were a good choice**



# Control region selection

- To check on MC/Data differences on the inputs to the training, set up a control region to study the input variables
- For control region selection, compare MC16a & 2016 data
  - MC: JZ[2-4]W multijet samples, full sim
- Use a dijet selection:
  - HLT Triggers: Pass J400 or J420
  - At least 2 jets
  - Leading $p_{T,\ Jet\ 1} > 400\ GeV$, sub-leading $p_{T,\ Jet\ 2} > 60\ GeV$
  - Two jets back-to-back $|\phi_{Jet\ 1} - \phi_{Jet\ 2}| > 3$
  - Balanced $p_T$: $\frac{p_{T,\ Jet\ 1} - p_{T,\ Jet\ 2}}{p_{T,\ Jet\ 1} + p_{T,\ Jet\ 2}} < 0.3$
  - Missing $H_T$ cut: $H_{T,miss} < 120\ GeV$
  - Take 5 leading jets as control region jets