Interim Report: Constraining the Higgs coupling to charm quarks with the ATLAS detector at the LHC

Jonathan Shlomi

September 3, 2019

Abstract

In order to enhance the sensitivity of the analysis for measuring the Higgs coupling to charm and bottom quarks, flavour tagging algorithms must be improved, specifically in their ability to distinguish between bottom and charm jets. By increasing the accuracy by which tracks from secondary decays are grouped into vertices we could build a more accurate picture of the underlying physics event, and therefore better distinguish between the jet flavors.

This research proposes an algorithm which learns to identify secondary decay vertices in jets with Graph Neural Networks (GNN). Preliminary results show an improvement in vertex finding quality over the existing algorithm used in ATLAS.

The next phase of the research will focus on optimizing the GNN approach to maximize performance in the context of the Higgs to $c\bar{c}$ physics analysis, and deploying the new algorithm in ATLAS software.

Contents

1 Project overview

Flavor tagging is based on identifying secondary decays inside jets, that is the key signature which separates b/c from light, and the characteristics of that decay separate b from c. The goal of this research is to focus on advanced machine learning techniques to increase the ability to distinguish between b and c jets, motivated by the desire to increase the sensitivity of the Higgs to $b\bar{b}$ and $c\bar{c}$ analysis.

In the beginning of the project we planned to create jet images and adapt techniques that are used for computer vision tasks. I became aware of a more appropriate data format and machine learning sub-field - Graph Neural Networks (**GNN**) [1]. Describing the jet, the collection of charged tracks (and potentially calorimeter clusters) as a graph is a more suitable data format for the task compared to synthetic images.

A critical part of improving vertex finding is to define a performance metric for the algorithm. Looking only at the downstream task (flavor tagging) performance is insufficient, since it does not illuminate how and why an algorithm fails to reconstruct the secondary vertex. In this report some possible performance metrics are described, but the question of finding the most suitable metric is still open.

1.1 Motivation

The Higgs boson was observed in 2012 with the ATLAS and CMS experiments at the LHC [2, 3]. One of the major goals of the physics program at the LHC is to measure the properties of this particle, and compare its properties to those predicted by the SM. Deviations from those predictions could imply new physics beyond the SM.

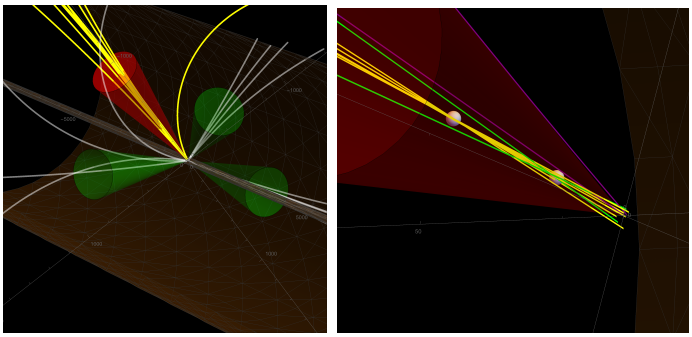
ATLAS published a first version of the VHcc analysis [4] Its result is an upper limit of 100 times the SM production rate for$ZH \to \ell\ell c\bar{c}$ at a 95% CL. CMS published a more comprehensive search for Higgs to $c\bar{c}$ [5] with an upper limit of $\mu < 70$, by including several additional channels in the analysis.

ATLAS has estimated the prospects for Higgs to charm coupling measurement senasitivity in the HL-LHC [6] at around $\mu < 6$, in absence of systematic uncertainties. This estimate does not take into account the challenges of flavour tagging in the HL-LHC, for which the algorithm presented in this report could provide a solution.

1.2 Flavor Tagging

The correct identification of jets originating from bottom quarks (b-tagging) and charm quarks (c-tagging) is a well established and fundamental part of the physics analysis performed at the LHC.

The signature by which heavy flavor jets are identified is the presence of secondary decay vertices inside the jet, the locations at which the b and c hadrons in the jet decayed into stable particles which were identified by the detector. This is illustrated in figure 1 - figure 1a shows a typical $t\bar{t}$ event, with one of the b jets highlighted in red, and figure 1b shows a close-up view of that b jet in the first 10 mm close to the primary vertex, with two secondary vertices, from the b and c hadrons.



(a) (b)

Figure 1: A typical $t\bar{t}$ event, with 4 jets. One of the b jets is highlighted in red (left) and a close up view of secondary decay vertices inside a b-jet is shown on the right.

The existing flavor tagging algorithms use two types of algorithms to identify this signature - impact parameter algorithms and vertex reconstruction algorithms. The impact parameter algorithms use only the impact parameter of the tracks associated to the jet, while the vertex reconstruction algorithms try to explicitly find the location of the secondary vertex, and to associate tracks to that vertex.

1.3 Structure of this Report

The structure of this report is as follows:

- A description of the jet dataset is given in section 2.
- Graph neural networks and how they are used for vertex finding are described in section 3
- Performance metrics for vertex finding and preliminary results are given in section 4
- The status of the VHcc analysis and related tasks are described in section 5
- The report is summarized in section 6, and the next stages of the PhD research are outlined in section 7

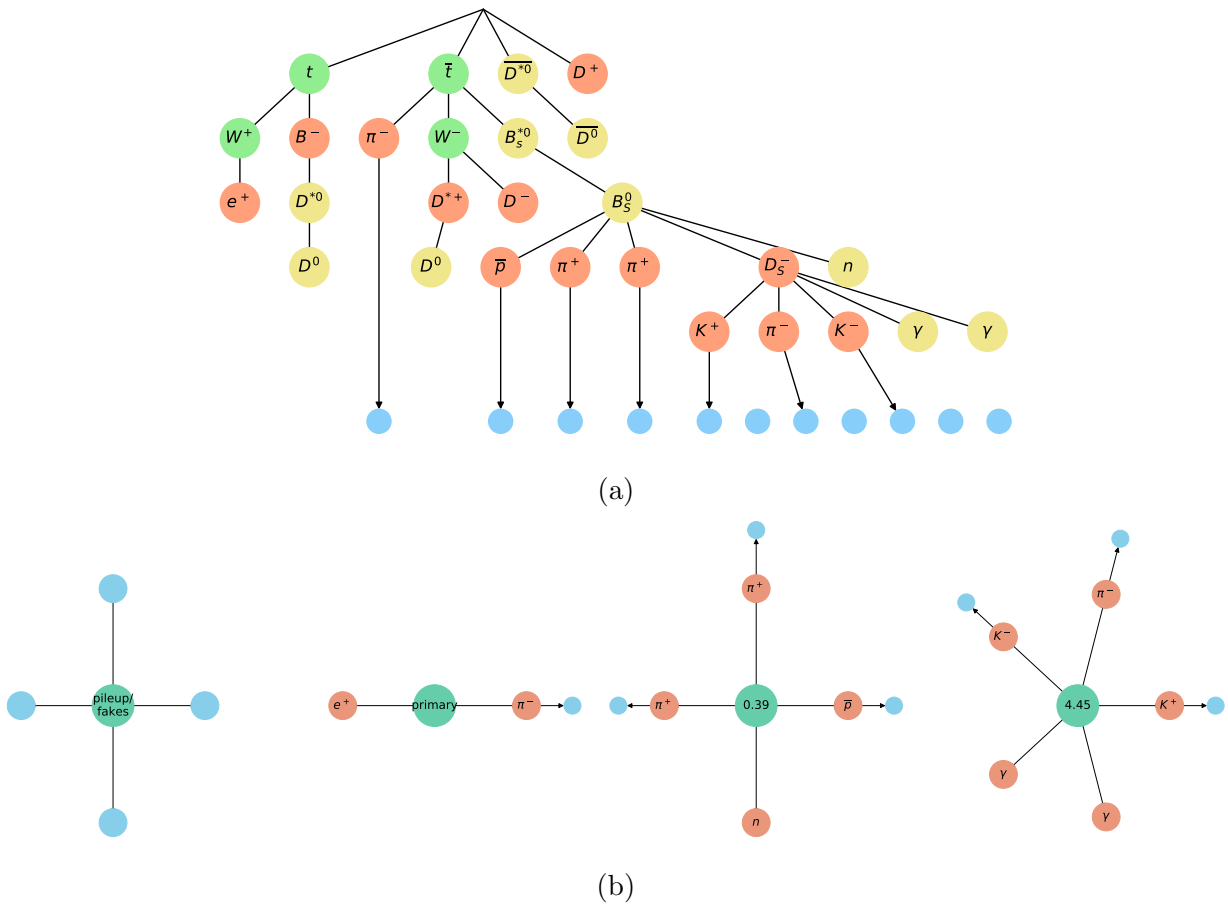
2 Secondary Vertex Finding

2.1 Decay topology for b and c jets

The data sample for this study consists of jets from $t\bar{t}$ events. The b-jets originate from the top quark decay, and the c and light jets originate either from the hadronic decay of the W boson, or from ISR/FSR. It is important to note that b jets typically include two secondary vertices, one from the decay of the b hadron, and a c hadron which originates from the b hadron decay, and c jets only include one decay vertex from a c hadron. All three flavors, b,c and light, can have additional secondary decays from V_0 decays and material interactions, and these are also targets of a secondary vertex finding algorithm.

The pt spectrum of b and c and light jets in this sample is very different. Equal numbers of jets from each class were selected in bins of pt/eta (1 GeV per pt bin and a 0.2 range in eta), since c-jets are the least numerous class, this effectively means that the b and light jets were selected to have the same pt/eta distribution as c-jets.

The simulation truth record of a b-jet decay chain is shown in figure 2a. It shows the decay $\bar{t} \to B_s^0 \to D_s^-$. Charged particles are shown in red, neutral particles in yellow. The reconstructed tracks associated to the jet are shown in blue at the bottom of the graph, with arrows indicating a match between the simulation truth particle and reconstructed track.



(a)

(b)

Figure 2: b-jets decay chain and reconstructed tracks association to truth particles. See text for details.

We can divide the reconstructed tracks into groups, based on their origin, this is illustrated in figure 2b. Each track can originate from the primary vertex (the interaction point), or one of the secondary decay locations. In this jet we have two, the B_s^0 decay at a distance of 0.39 mm from the primary vertex, and the D_s^- decay at a distance of 4.45 mm. Tracks can also be "fake", originating from pile-up or errors of the track finding algorithm. They will show up as blue dots with no connecting arrow in the graph of figure 2a.

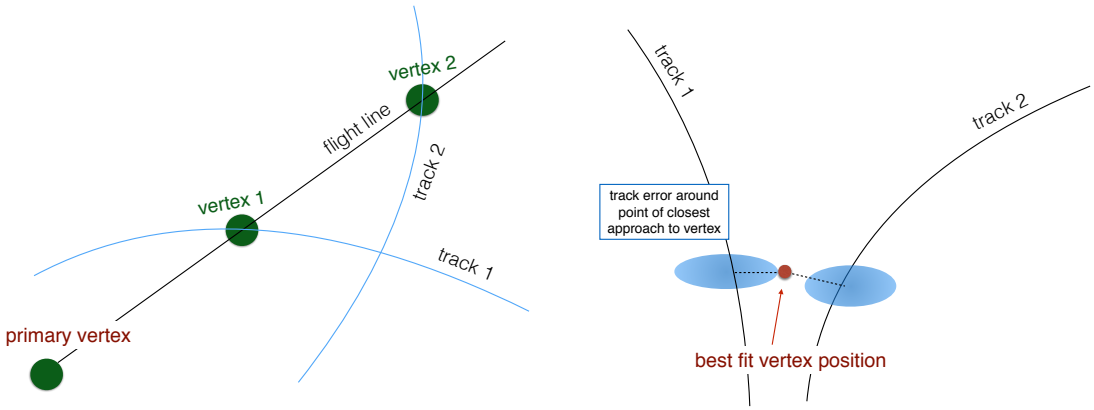
The task of secondary vertex reconstruction can be divided into two parts:

- Vertex Finding - Clustering together the tracks to reproduce the true division shown in figure 2b
- Vertex Fitting - Identifying the common origin point of a set of tracks group together in the vertex finding stage.

At this stage, we focus only on the first part, vertex finding. We can therefore formulate the task we want to perform as supervised clustering task - given a collection of tracks in a jet, find the correct partition of tracks to reproduce the truth-level based clustering.

2.2 The JetFitter algorithm

The standard ATLAS secondary vertex reconstruction algorithm is called JetFitter [7]. It performs both vertex finding and fitting. JetFitter attempts to find multiple secondary vertices inside the jet, corresponding to the b hadron vertex and the c hadron vertex. In order to constrain the fit and reduce fake vertices, the assumption is made that the vertices are on a straight line originating from the primary vertex. JetFitter starts by fitting all tracks as single track vertices to a flight line (figure 3a), and then iteratively tries to merge vertices based on vertex χ^2 quality criteria - figure 3b illustrates how the vertex χ^2 is computed . The JetFitter algorithm vertex finding is used as a baseline for checking the performance of the machine learning based algorithm described below.



(a) (b)

Figure 3: The JetFitter algorithm - see text for details

3 Graph Neural Networks for Vertex Finding

In section 2 the data and task of vertex finding were defined. When constructing a machine learning solution to particular task, once you have your input data and desired output, the next stage is to construct a neural network architecture that takes the data as input and outputs something that allows you to perform the target task.

In the following section we describe the neural network architecture that performs vertex finding. A more detailed description of the internal structure of this neural net architecture is given in appendix A.

3.1 Introduction to Graph Neural Networks

3.1.1 Fully Connected Neural Nets

The most basic version of a neural network is commonly referred to as a Fully Connected (**FC**) neural net [8], shown in figure 4. It is a model of a function $f_\theta(x) = y$ with fixed size input x and fixed size output y, and a set of parameters θ (commonly referred to as "weights"). Given a training dataset of (x, y) pairs, it is possible to train the model (by adjusting θ) to approximate the function which created the training dataset. In this context we think of this FC network as a building block used to construct a GNN.

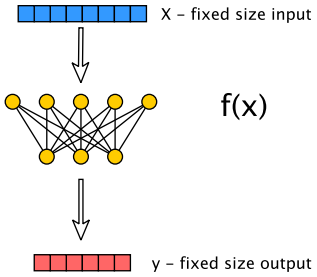


Figure 4: Fully Connected neural net.

3.1.2 The Graph Data Structure

In the context of GNNs, a graph is a data structure containing a set of objects, called "nodes", and connections between pairs of nodes called "edges". Nodes and edges can have information stored "on them" which is referred to as node (or edge) "features". The nature of the node/edge features depends on the specific data under consideration. The graph as a whole can also have some global features ("graph features").

3.1.3 GN Block

The general structure and common use cases of GNNs are described in [1]. Using a graph as input to a neural network is less straight forward since graphs have a variable number of nodes and edges, and there is no inherent order to the nodes or edges. The neural network architecture must be suited to those properties.

The basic building block of GNN is referred to in [1] as the "GN block". A GN block takes as input an arbitrary graph with node, edge and graph features, and outputs the same graph with updated node, edge and graph features. This is shown schematically in figure 5. Internally the GN block is composed from a few neural networks that work together in a sequence to perform this task. A key point is that the internal structure of the GN block includes "message passing" - propagation of information between different nodes in the graph, such that the graph structure is taken into account in the output graph, node and edge features. The graph can then be used to perform a number of tasks on graphs, commonly node, edge or graph classification.

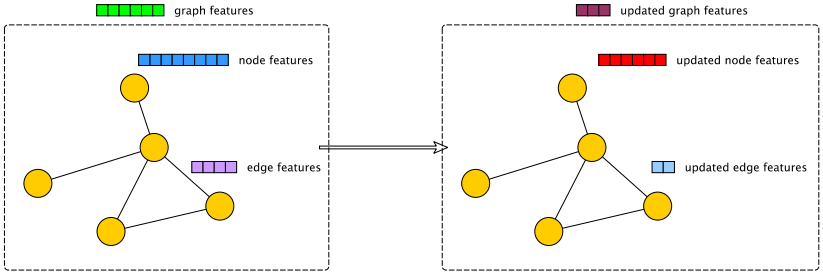


Figure 5: A GN block.

3.2 Defining Vertex Finding as a GNN task

A jet can be represented as a graph, where each nodes is one of the reconstructed tracks associated to the jet. Each node is connected to all other nodes by an edge. There is no physics information conveyed by the edges, they serve as "scaffolding" for the GNN (see appendinx A).

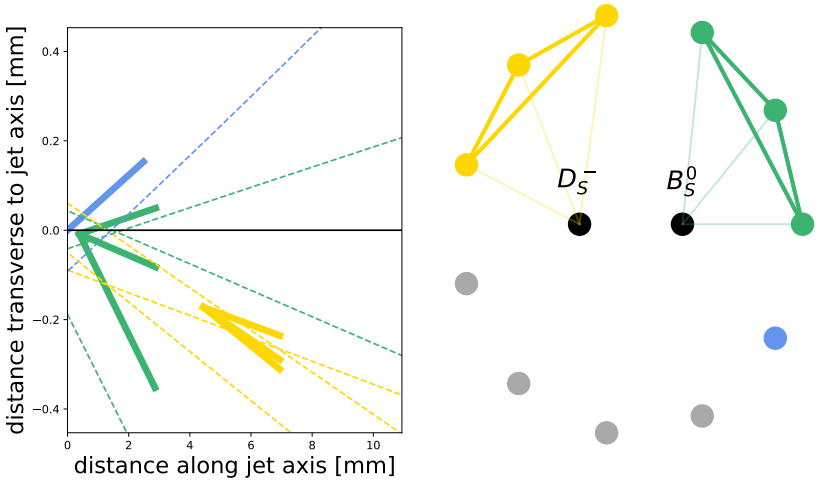
We attach labels to the edges - edges that connect two nodes from the same secondary decay have a label of "1" and all other edges a label of "0".

Nodes are also given labels. Nodes from secondary decays are labeled "1" and other nodes (primary vertex or pileup/fakes) are labeled "0".

In order to perform clustering, we ask the GNN to predict the node and edge labels given the input data (discussed in section 3.2.1 below).

An example jet is shown in figure 6a, with reconstructed tracks shown as dashed lines, and the true particle origins indicated by the solid lines. The matching "jet graph" is shown in figure 6b, it shows the tracks of the jet (the nodes of the graph) color coded based on their origin - green nodes from the b hadron decay, yellow nodes from the c hadron decay, gray nodes for pileup/fakes and blue nodes for tracks from the primary vertex.

The edges that connect the secondary nodes are shown, while all other edges are omitted. The b and c hadrons are also shown, to indicate the origin of each node.



(a) (b)

Figure 6: A jet in real space (left) and shown as a graph (right). The tracks are color coded according to their vertex origin. tracks from the primary are colored in blue, pile up tracks are shown in gray.

3.2.1 Node Features

The node features and graph features are the information provided to the GNN to perform it's task. Tracks are commonly represented by their perigee parameters, illustrated in figure 7. The perigee parameters describe the track position and momentum at the point of closest approach of the track to the origin.

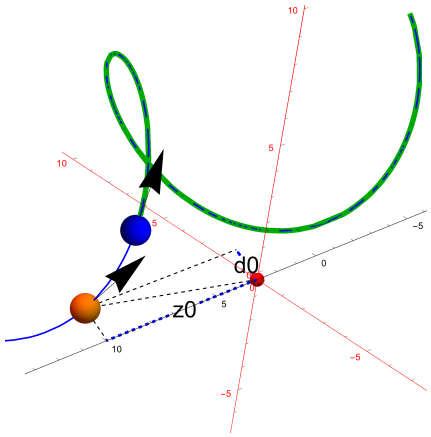


Figure 7: The perigee point is the point of closest approach of the track to the origin (orange point). The perigee parameters desribe the track position and momentum at that position. Given the perigee parameters, the position and momentum of the particle at any other point along the track (for example the blue point) can be computed.

Since the GNN operates on a single jet, a different representation was used, illustrated in figure 8. Given a jet direction, specified by the jet p_T, η and ϕ, the track representation is computed in the "jet

frame", defined by a rotation from the detector frame Z axis to the jet axis. Node features are the position of the track in the plane transverse to the jet axis, and the track momentum direction and size in the jet frame.

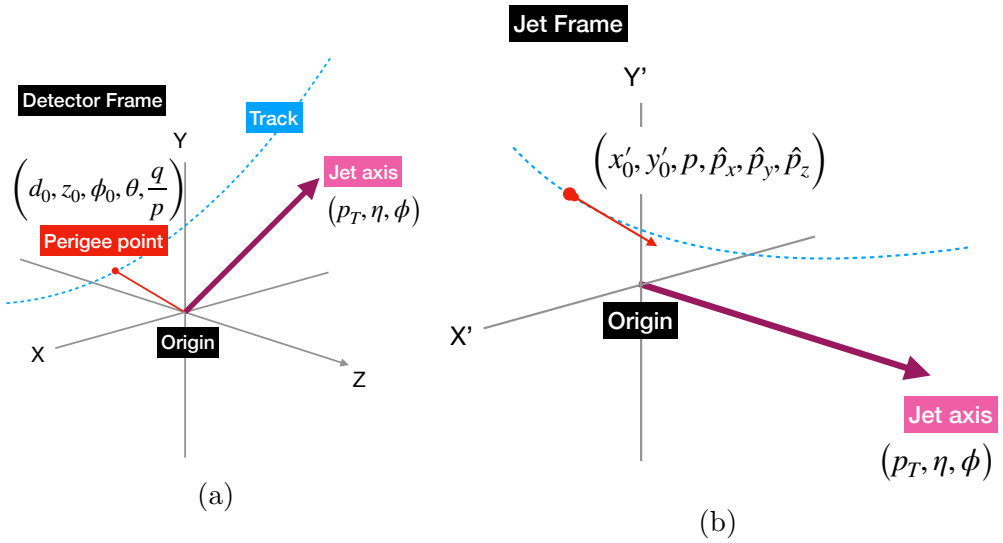


Figure 8: Given a jet direction and track in the detector frame (left), the track position and momentum can be computed in the plane transverse to the jet axis (right).

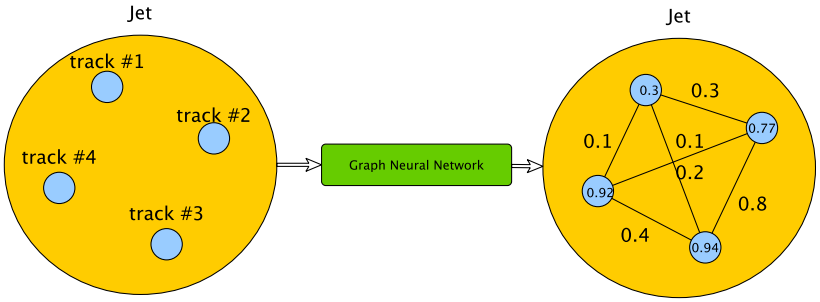
3.2.2 Converting the GNN Output to a Clustering Decision

The scheme for performing clustering with a GNN is illustrated in figure 9. In order to perform clustering, we ask the GNN to perform two operations on this graph -

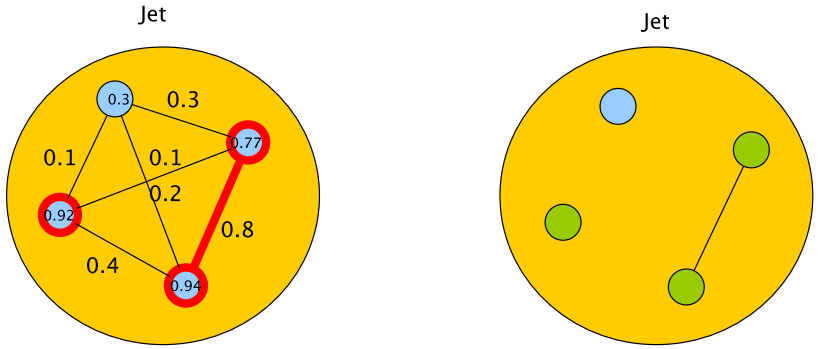
- Node classification - binary classification of each track as secondary/not secondary
- Edge classification - binary classification of each edge (a pair of tracks), to predict if they are in the same cluster.

The GNN outputs a score in the range $(0, 1)$ for each node (track) and each connection between pairs of nodes. Then a post-processing step is performed to convert the neural network output to a clustering assignment. Nodes with a score above some threshold are labeled as secondary nodes, and edges with a score above a particular threshold connect nodes to be in the same cluster.

The threshold for selecting nodes and edges can be tuned to either increase the efficiency or increase the purity of reconstructed vertices. The desired behavior of the algorithm depends on the downstream use in physics analysis.



(a) The GNN processes the input graph, and assigns a score to each node and edge.



(b) Nodes and edges that pass some user-selected threshold create the final vertex assignment of the algorithm.

Figure 9: Using a graph neural network to perform clustering with a combination of node and edge classification.

3.2.3 Using Vertexing GNN for Flavor Tagging

Once an algorithm has been trained for vertex finding, it can be combined in a flavor tagger in a number of different ways. One possibility is illustrated in figure 10. The vertex finding post-processing step can be neglected, and the output graph of the vertex finding step can be fed into a different GNN, which performs jet classification. This can be combined with the existing flavor tagging algorithm, or work as a stand alone flavor tagger.

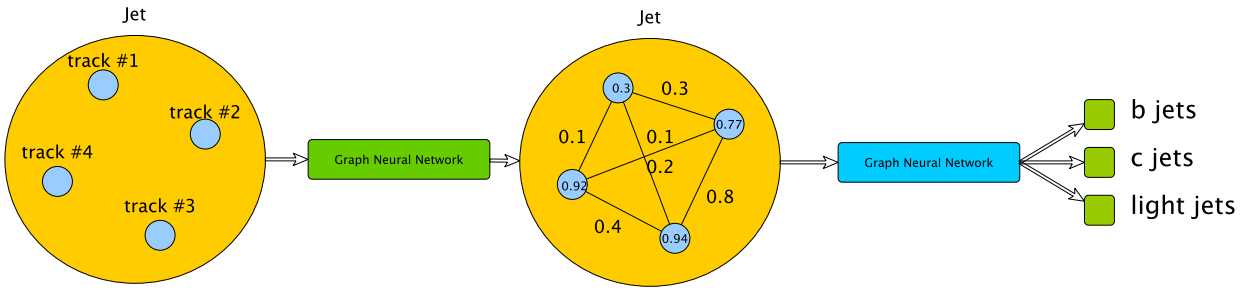


Figure 10: GNN based flavor tagging, with vertex finding as an intermediate step.

4 Definition of a Vertex Finding performance and Initial Results

The GNN described above was trained on a 25K sample, containing 10K b jets, 10K c jets and 5K light jets. The algorithm was trained for about 50 epochs (an epoch is a complete round of training on the entire training dataset), with performance reaching a plateau after about 10 epochs. The training time is approximately 3 hours.

When it comes to quantifying performance, there is no standard metric for clustering. The quality of clustering in usually only measured with respect to the downstream task (jet flavor classification in this case). But in this step of algorithm development we want to focus on having the most accurate picture of the underlying physics.

We quantify the vertex finding algorithm's performance from a few different perspectives:

1. True Vertex Reconstruction
2. Reconstructed Vertex Purity
3. Jet-Level Performance

Each of these performance metrics is explained below, and some performance plots from an independent test dataset of about 20K jets (both b, c and light) are presented.

4.1 True Vertex Reconstruction

A "true vertex" is a collection of tracks originating from the same physical location. Given a clustering algorithms output, a true vertex can belong to one of four categories (see figure 11):

1. **Failed** - the vertex was not identified by the algorithm
2. **Perfect** - the vertex was identified and all of its tracks were grouped into one vertex
3. **Partial** - the vertex was partially identified, all the tracks that were identified were grouped into one vertex
4. **Split** - the vertex was identified, but tracks were split into two predicted vertices.

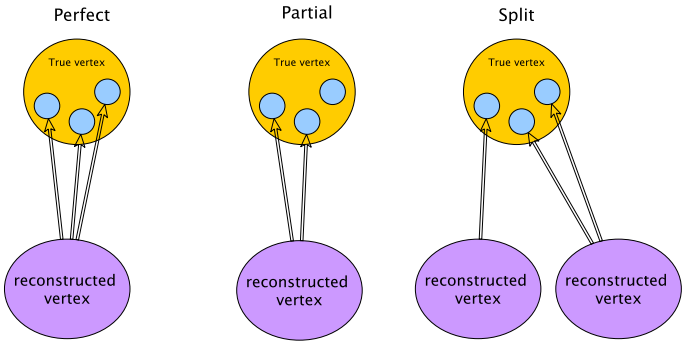


Figure 11: True vertex quality categories.

In figure 12 The distribution of true vertex categories is shown, for b,c and light jets (split by column) and for true vertex with only a single track (top) and two or more tracks (bottom). There is a small improvement in the number of "perfect" vertices in b and c jets, but the overall picture shows a performance consistent with JetFitter.

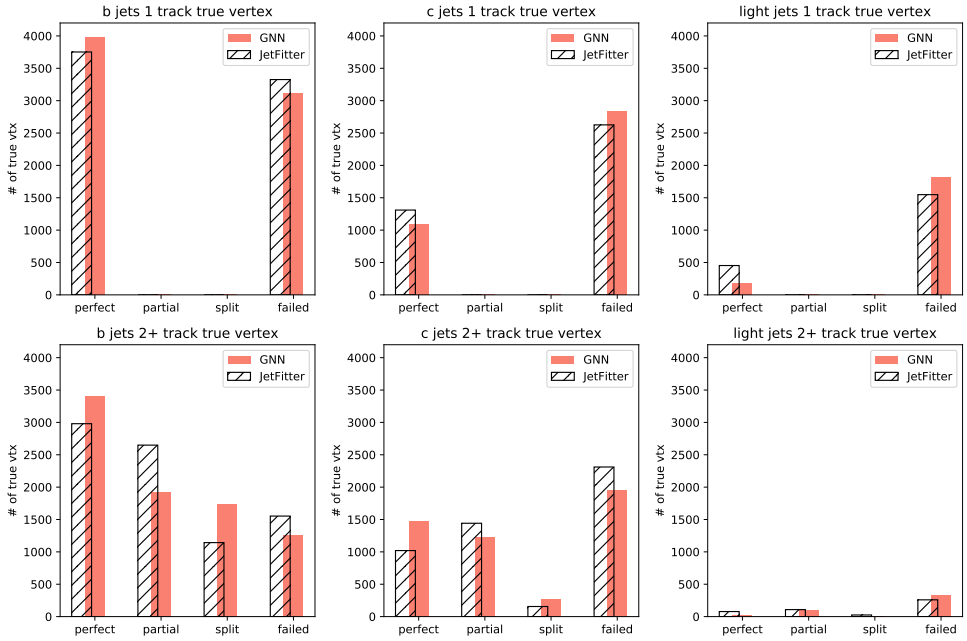


Figure 12: Distribution of true vertex categories ("Failed" category not shown)

4.2 Reconstructed Vertex Purity

Reconstructed vertices are groups of tracks associated to a secondary vertex by the algorithm (either the GNN after the post-processing step or JetFitter). A reconstructed vertex belongs to one of four categories (see figure 13):

1. **Perfect** - the vertex matches exactly with a true vertex
2. **Partial** - the vertex contains tracks from a single true vertex
3. **Merged** - the vertex contains tracks from two or more true vertices
4. **Fake** - the vertex contains only tracks not originating from a true vertex

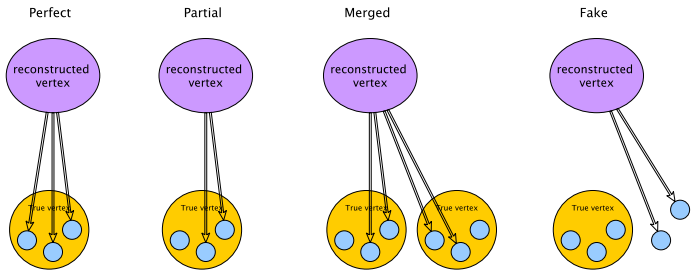


Figure 13: Reconstructed vertex categories.

Figure 14 shows the distribution of reconstructed vertex categories, Again we see a similar performance to JetFitter, with a slight decrease in fake vertices for b and c jets, but overall similar performance.

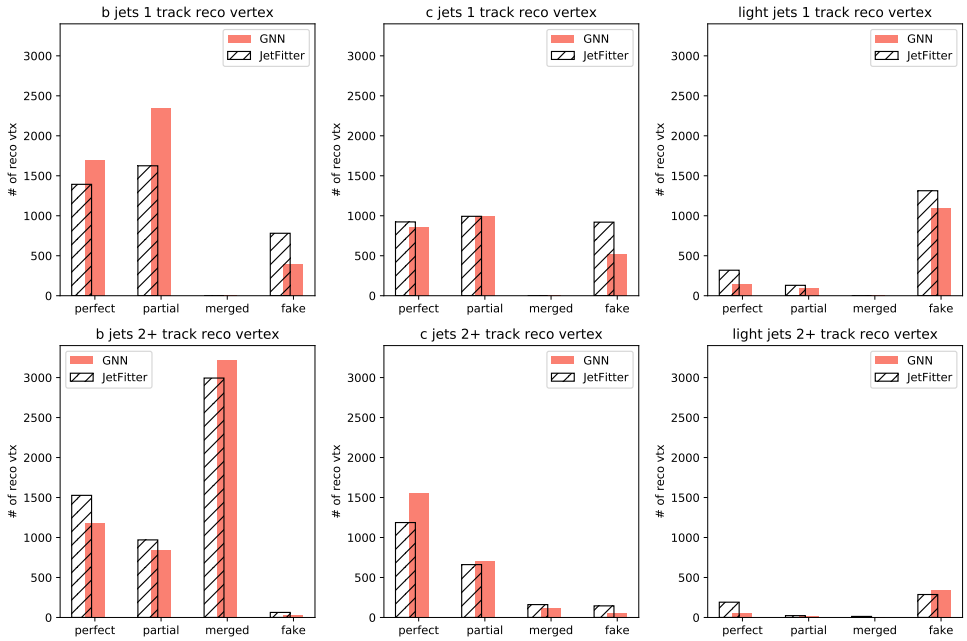


Figure 14

4.3 Jet-level vertexing performance

To define a performance measure on the jet level, we need to use a metric for the similarity of the clustering given by the algorithm and the ground truth. A commonly used measure for similarity between 2 clusterings (partitions of a set) is the the Adjusted Rand Index (ARI) [9].

The ARI serves as a "jet score" computed from the truth level clustering and the vertex assignment of the algorithm (either JetFitter or the GNN). Its values are between 1 and -1, with 1 showing perfect agreement, 0 showing performance similar to random clustering, and -1 shows the worst possible agreement.

Given two partitions of a set, the Rand Index (not adjusted) is defined by equation 1.

$$RI = \frac{\text{true positives} + \text{true negatives}}{\binom{n}{2}} \tag{1}$$

where "true positives" is the number of pairs of elements that are in the same cluster in both partitions, and true negatives are the number of pairs of elements that are not in the same cluster in both partitions. The normalization of $\binom{n}{2}$ is the number of element pairs in the set.

The Adjusted Rand Index (2) is a the same measure, but normalized by the expected RI value of random clustering.

$$ARI = \frac{\text{RI} - \text{E[RI]}}{\max(\text{RI}) - \text{E[RI]}} \tag{2}$$

Figure 15 shows a few different plots comparing the ARI of JetFitter and the GNN for the three different flavors.

The top row shows the distribution of ARI, it shows a similar trend for JF and GNN, with an advantage to the GNN for b and c jets.

The middle row shows the percentage of jets in the sample that have either equal JetFitter and GNN ARI, or a larger ARI for one of the algorithms. This shows that the GNN performs better for a large fraction of the jets, especially for b and c jets.

The bottom row shows the difference between ARI scores for those jets that have either better GNN ARI or JF ARI. This plot shows that the advantage of the GNN over JetFitter in jets where it performs better follows the same trend as the jets where JetFitter performs better.

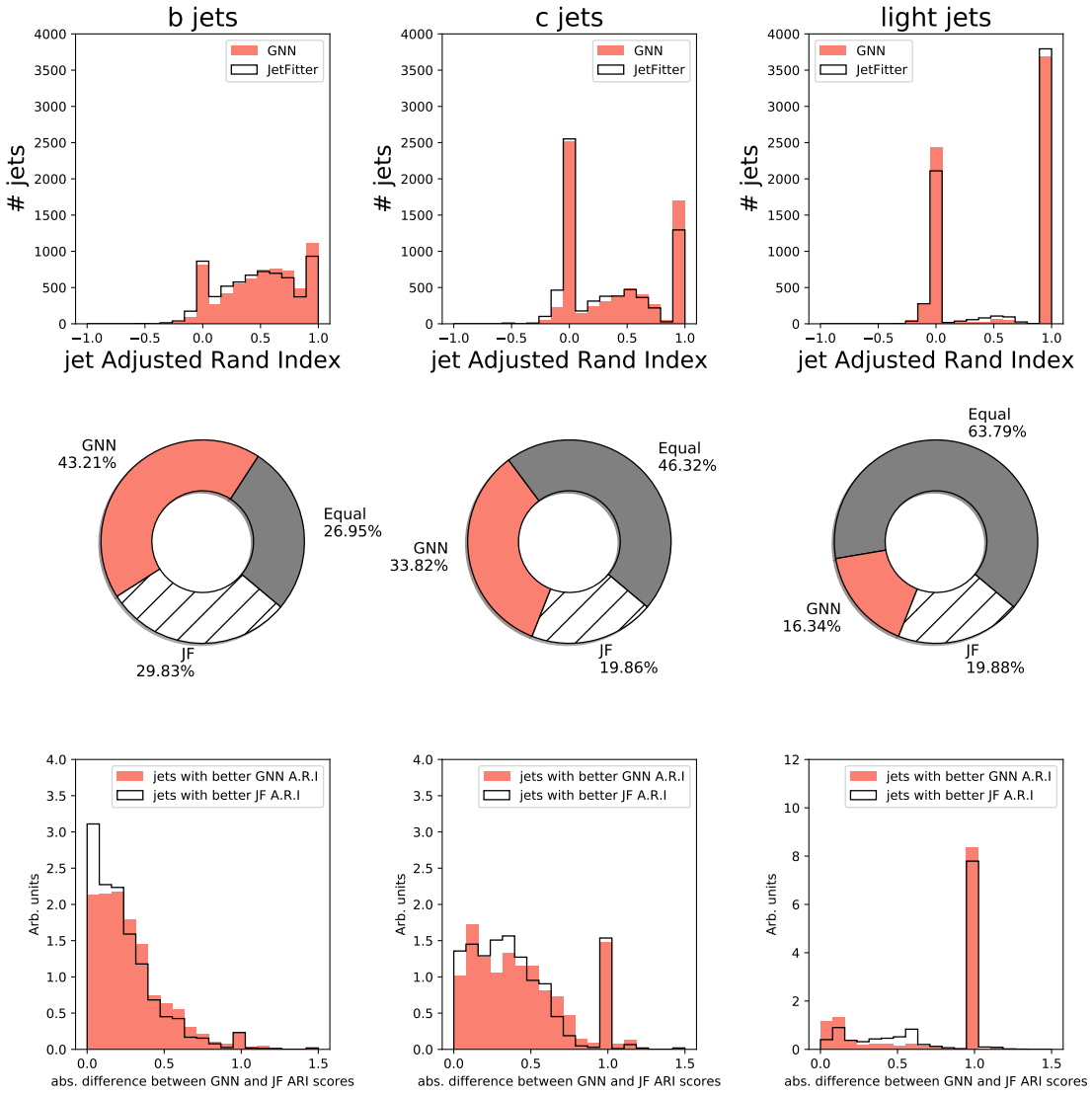
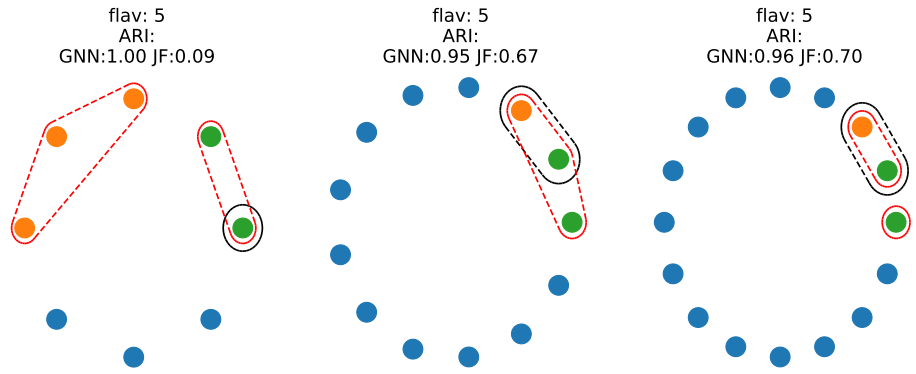


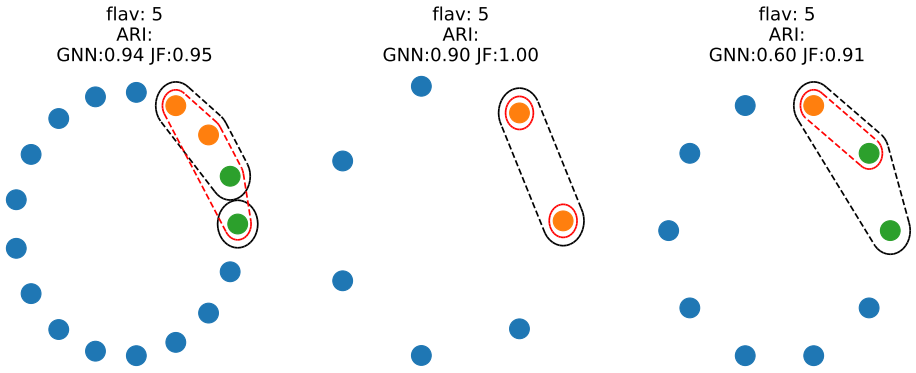
Figure 15: A comparison of jet ARI scores for JetFitter and GNN. See text for details.

4.3.1 Exploring the Performance

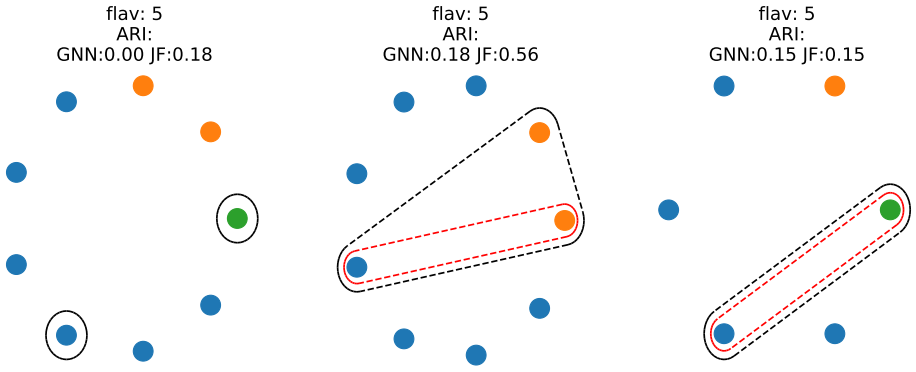
In order to understand this performance, we can explore the dataset by looking at specific examples. Some example b-jets (only b-jets for the moment, c and light jets will be the subject of a more extensive data exploration in the future) are shown in figure 16, with the JetFitter clustering shown as black dashed lines, and the GNN clustering as red dashed lines. The true clustering is indicated by the colors of the nodes - nodes that should be clustered together share the same color, primary vertex and fake vertices are colored in blue.



(a)



(b)



(c)

Figure 16: Examples of b-jets and the JetFitter and GNN cluster assignments. The different rows show jets where the GNN performed better (16a), jets where JetFitter performed better (16b) and jets where both performed poorly (16c).

5 VHcc Analysis

The end goal of this project is to deploy an improved secondary vertex reconstruction in flavor tagging and apply it to the VHcc analysis. At this point the analysis is using the standard ATLAS flavor tagging tools.

5.1 Analysis Status

An analysis utilizing the 3 decay channels used by the VHbb analysis is in preparation, aiming for publication in the next 6 months. After this publication the next stage of the VHcc analysis might be a combined effort with VHbb, for the LHC run-3 analysis.

5.2 Truth Tagging

A key feature of the VHcc analysis that is also related to flavour tagging is a procedure commonly called "truth tagging". The procedure is aimed at reducing the statistical uncertainty of the background samples enriched in light jets. Light jets have a low efficiency for being tagged as either b or c jets. Therefore, samples which are predominantly composed of light jets suffer from high statistical uncertainty after passing the flavor tagging requirements of the analysis.

When applying truth tagging - events that do not pass flavor tagging are not rejected. Instead, each sample is weighted according to the probability that it will pass the flavor tagging requirements. These probabilities are computed from flavor tagging efficiency maps, under the assumption that the probablity of an event to have 2 tagged jets is:

$$P(\text{event has 2 tagged jets}) = P(\text{jet 1 is tagged}) \cdot P(\text{jet 2 is tagged}) \tag{3}$$

and the probability of one jet to be tagged is its efficiency conditioned on its flavor, p_T and η.

One of the fundamental assumptions in flavor tagging is that the properties of the decay depend only on energy of the jet. that jets with a given p_T and η from any processes ($t\bar{t}$ or otherwise) have the same secondary decay properties. Therefore any flavor tagger performance can be parameterized in terms of p_T and η and the jet flavor.

In practice when performing truth tagging we observe that there are differences between the efficiency maps of different processes. This requires producing custom flavor tagging efficiency maps for every sample one wishes to truth tag. One of the key open questions in flavor tagging is the identifications of the exact factors that contribute to these differences. If these factors are identified and can be corrected for, a more general truth tagging procedure can be used.

5.3 Authors Contribution to the Analysis

The author is part of the 1-lepton channel analysis team, focusing on software development of the analysis framework, and flavor tagging related tasks such as developing truth tagging (described in the previous section). In the general ATLAS setting, the author works on building and maintaining all the flavor tagging software tools.

6 Summary and Conclusions

Secondary vertex finding is a crucial aspect of flavor tagging. In this preliminary stage we found that using graph neural networks to perform the task of vertex finding is a promising avenue for improving the quality of b jet vs. c jet identification by more accurately reconstructing the decay chain in jets.

There are two key aspects to explore in order to deploy this algorithm in ATLAS and in the VHcc analysis:

- The performance in real data, when used for flavour tagging. This requires an infrastructure capable of running the trained algorithm inside ATLAS software, and applying the established procedures for data vs. MC calibration of flavor tagging efficiency.
- Optimizing the algorithm in terms of model architecture, training procedure, and understanding it's performance in the MC dataset in terms of vertex reconstruction accuracy.

These aspects must be developed in parallel to allow the algorithm to be used in the next iteration of the VHcc analysis and in future work in the HL-LHC.

7 PhD Timeline

The two aspects of the project outlined in the previous section are broken up into a few key milestones that include publishing the method with a toy dataset as a proof of concept, and then implementing the method in ATLAS software and propagating it to the physics analysis. These milestones and their expected dates are detailed below.

- August 2019 - Interim report
- January 2020 - Proof of concept paper for secondary vertex finding with graph neural networks
- March 2020 - ATLAS software implementation of graph neural net for secondary vertex finding
- December 2020 - Final report
- January 2021 - Implementation of tagger into ATLAS software and VHcc Analysis
- March 2021 - ATLAS pub note on secondary vertex finding with ATLAS data
- June 2021 - PhD thesis

References

[1] P. Battaglia et al. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[2] ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett.*, B 716:1–29, 2012.

[3] CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett.*, B 716:30–61, 2012.

[4] Search for the decay of the Higgs boson to charm quarks with the ATLAS experiment. Technical Report ATLAS-CONF-2017-078, CERN, Geneva, Nov 2017.

[5] Search for the standard model Higgs boson decaying to charm quarks. Technical Report CMS-PAS-HIG-18-031, CERN, Geneva, 2019.

[6] Prospects for $H \to c\bar{c}$ using Charm Tagging with the ATLAS Experiment at the HL-LHC. Technical Report ATL-PHYS-PUB-2018-016, CERN, Geneva, Aug 2018.

[7] Giacinto Piacquadio and Christian Weiser. A new inclusive secondary vertex algorithm for b-jet tagging in ATLAS. *J. Phys. Conf. Ser.*, 119:032032, 2008.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, December 1985.

Appendix

A Graph Neural Net Architecture

The architecture for the GNN used to perform the vertex finding operation is shown in figure 17. Each one of the green boxes is a fully connected neural net which performs a specific operation on the graph.

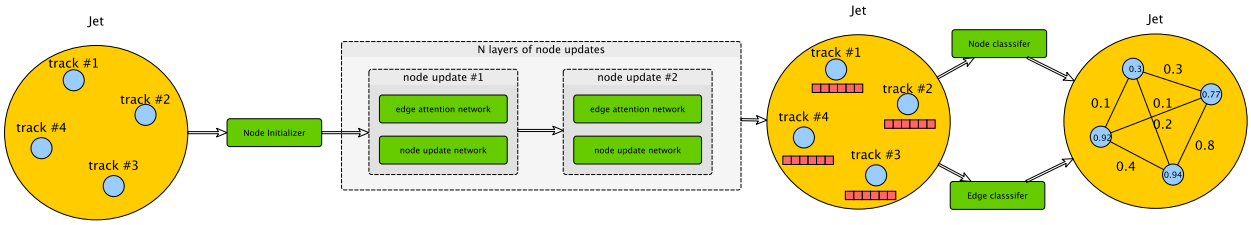


Figure 17: An overview of the GNN structure.

The internal neural networks are:

- Node Initializer
- Edge Attention
- Node Update
- Node Classifier
- Edge Classifier

They operate in a sequence, each step is explained below:

Node Initializer

The node initializer operates on each node in the graph independently, taking as input the node features and jet features, and outputting a vector of numbers. This is shown in figure 18a. The operation of the node initializer network on the graph is shown in figure 18b. The output vector for each node is added to the node features - this is referred to as a "hidden state". It is sometimes called "node embedding" or "node representation" depending on the context.

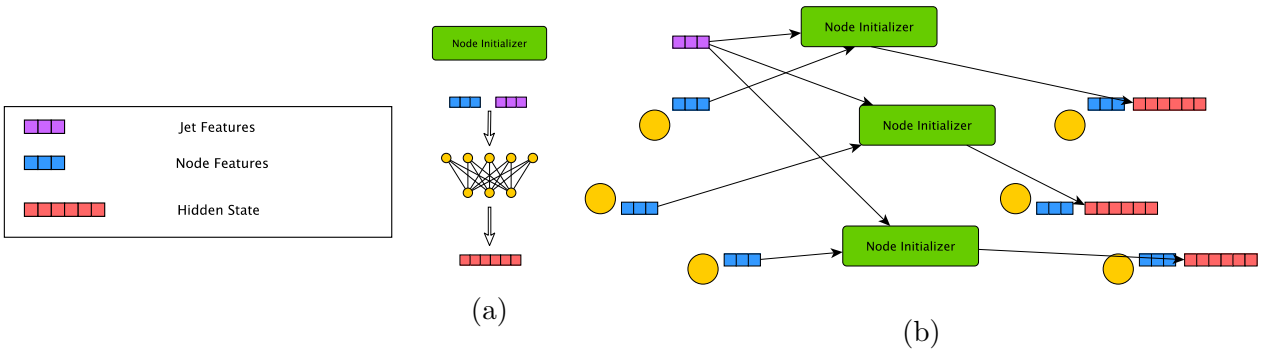


Figure 18: Node Initializer network.

Edge Attention

The structure of the FC edge attention network is shown in figure 19a, and the operation of the edge attention network on the graph is shown in figure 19b. This network gives each connection between pairs of nodes a weight, which is a single number. This weight is supposed to represent the "importance" of a particular node to its neighbor. It does not have a value we enforce on the network, it is left up to the neural network to "decide" during training how to compute these weights.

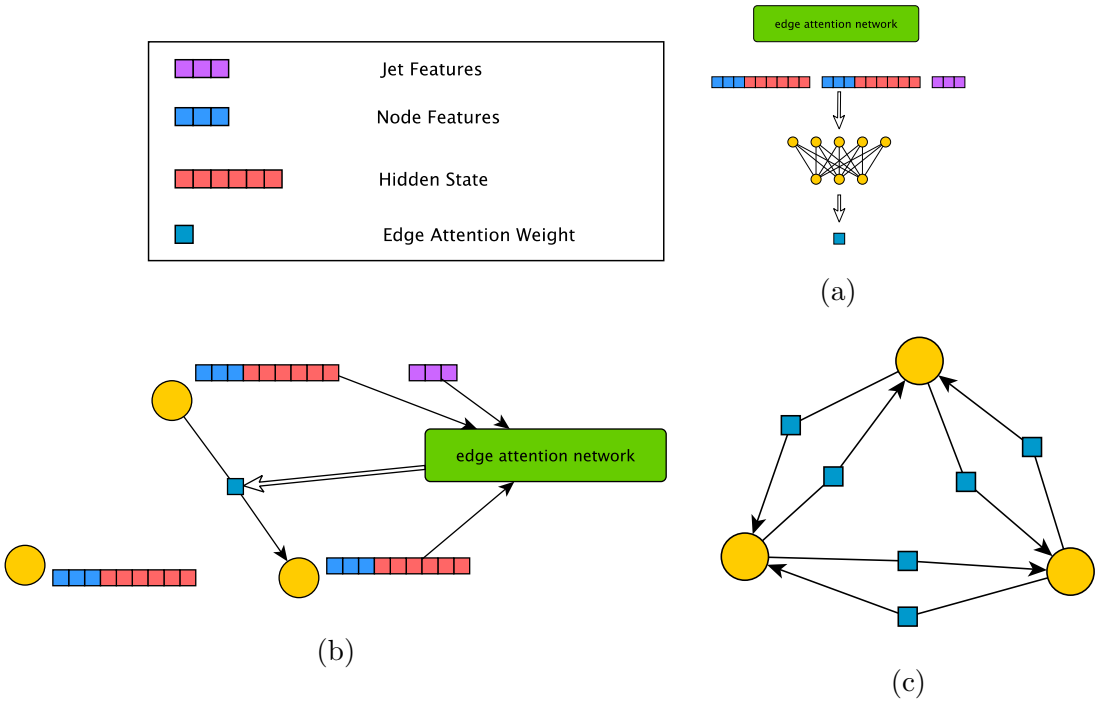


Figure 19: Edge attention network. The basic FC network structure of the edge network (a). The operation of the network on the graph (b). After the edge network has been applied to all edges, each edge has an "edge attention weight" stored on it.

Node Update

Once the edge weights have been computed, for each node a weighted sum of its neighbors is computed as shown in figure 20. This weighted sum, in combination with the nodes existing hidden state and features, is used to compute an updated hidden state for the node. This processes represents the neural network aggregating information from the rest of the graph and storing that information in the node hidden state. The process of edge attention and node update is repeated a few times. The decision of how many times to repeat this "message passing" processes is up to the designer of the network. In this particular application it was repeated twice.

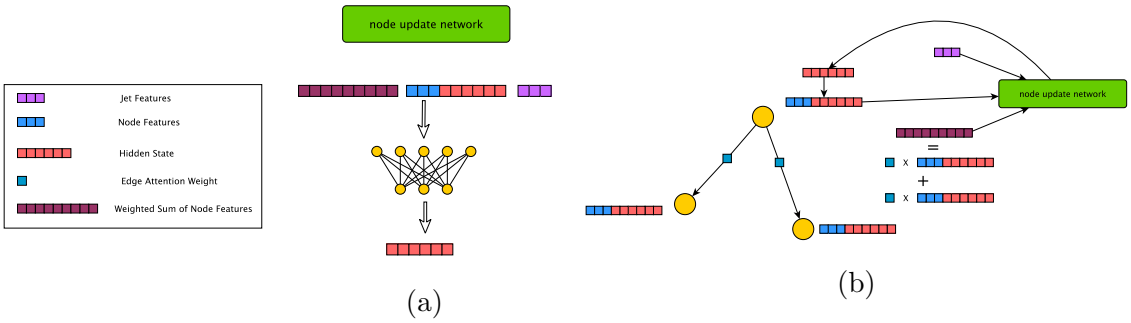


Figure 20: Node update network.

Node and Edge Classifier

After the node updates have concluded, each node should have in its hidden state information regarding the rest of the graph, and the relationship between the nodes. This allows dedicated networks to classify each node based only on its hidden state, or pair of nodes based on their hidden states. The node classifier is similar in its architechture to the node initializer. The edge classifier has a similar architecture to the edge attention network. These provide the outputs of the GNN.