A Quantum Computing Approach To Particle Track Reconstruction

Supervised by Dr Sarah Malik

Lucas Curtin

Physics Student

Word Count: 6935

UCL
Physics Department
UCL Physics Building, London
April 2023

Contents

1 Introduction **2**

2 Current Approach **2**
2.1 Kalman Filters . . . 3

3 Graphs **5**
3.1 Graph Data Example . . . 5
3.2 Graph Neural Networks . . . 6

4 Quantum Computation **7**
4.1 Logic Gates . . . 7
4.1.1 CNOT Gate . . . 7
4.1.2 Hadamard Gate . . . 8
4.2 Bell State . . . 9

5 Data **11**
5.1 Data Preparation . . . 11
5.2 Pre processing . . . 11
5.3 Graph Construction . . . 14
5.3.1 X Matrix . . . 14
5.3.2 R Matrix . . . 15
5.4 Example Case . . . 15
5.5 Data Flow . . . 16
5.5.1 Input Network . . . 16
5.5.2 Edge Network . . . 18
5.5.3 Node Network . . . 19

6 Training **20**
6.1 Classical Layers . . . 20
6.2 Quantum Layers . . . 21
6.2.1 MPS: Matrix Product States . . . 22
6.2.2 TTN: Tensor Tree Network . . . 23
6.2.3 L-VQE: Layer Variational Quantum Eigensolver . . . 24
6.3 Weighting . . . 25
6.4 Metrics . . . 25
6.5 F1 Score . . . 26
6.6 AUC-ROC Curve . . . 26

7 Results **27**
7.1 Further Investigation and Comparison . . . 29

8 Conclusion and Discussion **30**
8.1 Reducing Expressibility . . . 30
8.1.1 Entangling Dropout . . . 31

9 Acknowledgements **31**

A Events **36**

B Parameters and Performance **37**

1 Introduction

It is expected that at the start of 2029, the European Organisation for Nuclear Research (CERN) will be upgrading to the High Luminosity Large Hadron Collider (HL-LHC) [1]. This will greatly increase the rate of particle collisions, also known as luminosity. The High-Luminosity LHC will produce at least 15 million Higgs bosons per year, compared to around three million from the LHC in 2017. This poses a computational problem as modern methods might not be able to feasibly handle a projected ten-fold increase in data [2]. The emerging field of quantum computing is a possible solution to this as it offers a new approach for large combinatorial problems. This paper will investigate a quantum analogue to neural networks, parameterisable quantum circuits (PQCs), and if they can offer a quantum advantage over current classical approaches. Maintaining efficient and improving the reconstruction of data from CERN could uncover new physics and the development of quantum algorithms could have a wide field of meaningful applications such as in the climate crisis [3].

2 Current Approach

At the Large Hadron Collider (LHC) particles are accelerated to speeds close to the speed of light and are then collided. The higher the energy of the particles, the shorter their de Broglie wavelength thus providing greater detail to probe them. This wavelength is an example of the manifestation of wave-particle duality that occurs on the quantum scale and can be found in Eq. 1.

$$\lambda = \frac{h}{p} \tag{1}$$

where λ is the De Broglie wavelength, h is Planck's constant and p is the momentum of the particle. After the collision, the resultant particles deposit energy in the particle detector which can be seen in Figure 1. The detector has different segments for different classes of particles. This is due to their different physical qualities warranting different means of detection. For example, charged particles are bent by a magnetic field and the radius of the curvature of their movement can be used to determine the momentum using Eq. 2.

$$r = \frac{mV}{qB} \tag{2}$$

where m is the mass of the particle, V the velocity, q the charge, and B the strength of the magnetic field that causes the curvature. The paths of

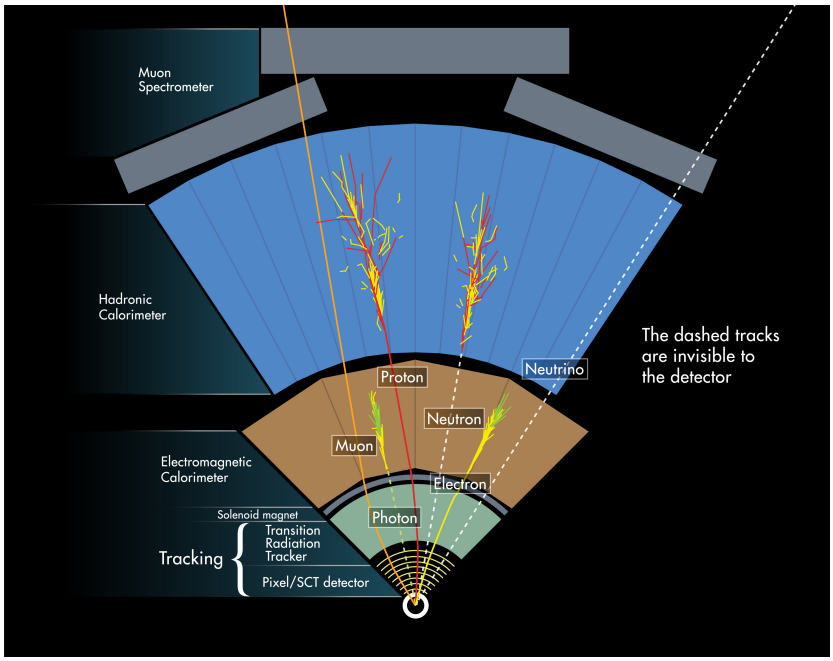


Figure 1: Diagram of particle paths in detector [4]. The different regions of colour represent different sections in the detector. For example the electromagnetic calorimeter (ECAL) is designed to slow down and detect muons and electrons but hadrons will be able to pass through due to their higher mass.

the particles have to be calculated to determine the radius of the curvature of their path in a process known as track reconstruction. Track reconstruction proves difficult as the tracks themselves are not shown as continuous data, but rather in discrete points as a particle passes through a layer of the detector. Particles depositing energy in these well-defined locations leads to the challenge in distinguishing which particle detections belong to which particle paths. The locations of particle detections are often referred to as hits and the paths they belong to as tracks. Track reconstruction can be viewed as a classification problem where, for example, each of tens of thousands of hits need to be placed into one of a thousand possible tracks.

2.1 Kalman Filters

The current method for ATLAS, one of the largest ongoing experiments at CERN, currently uses Kalman Filters [5]. Kalman filters operate by repeated use of the least squares estimator, a statistical tool that is used to fit to data [6]. The primary tracking process involves formatting seeds from typically 3 hits in the silicon detectors as seen in Figure 2. There are physical constraints placed on the

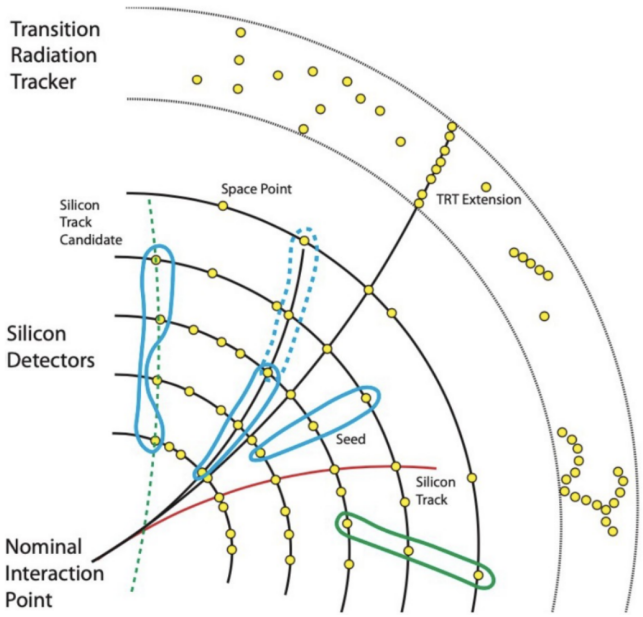


Figure 2: Track and Vertex reconstruction in ATLAS for LHC Run-3 and High-Luminosity phases [9]. The dashed lines show potential track candidates where as the dashed circles show potential seeds. These seeds are typically comprised of three hits known as triplets.

construction of seeds such as maximum transverse and longitudinal distances [7]. Any seeds that don't satisfy these constraints won't be considered in the later track candidate building to save on the computational cost. For example, a hit on the opposite side of the detector is very unlikely to be a compatible hit as it means the particle would have turned by a very large angle. The seeds that form multiple paths are then acted on by combinatorial Kalman filters to find track candidates. Kalman filters operate by predicting the particle's future position and velocity using a model of its motion. Then the distance is measured between this point and the location of hits on this detector. The hits that have the lowest χ^2 are then added together to form the track candidate. A χ^2 test is a measure of how far away a data point is from an expected value. The filter recursively operates due to the results being frequently imperfect as a result of detector resolution, multiple scattering, and noise. The filter has to consider lots of different variables and due to its recursive nature can be a highly computationally expensive operation. The current algorithms employed on ATLAS are highly novel and optimised but may not be able to withstand this high increase in data as they scale worse than quadratically $\mathcal{O}(n^6)$ [8].

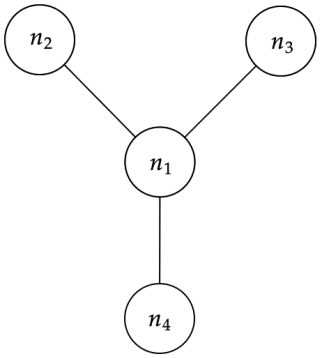


Figure 3: Example graph data consisting of 4 vertices and 3 edges. The edges in this example are undirected, where a directed edge would act from one node to another but not necessarily in reverse. For the hit data, the edges will be directed as it is expected that the particles will emanate from the centre of the detector to the outer layers.

3 Graphs

The hit and track data associated with this problem can be interpreted as graph data. A graph is described by a set of vertices and edges: $G = (V, E)$. Objects in data such as pixels in an image or people in a social network can be represented as vertices and the relationships between them can be represented by edges. This can be utilised in a wide range applications such as the study of molecules as the composition can be broken down into a graph data structure [10].

3.1 Graph Data Example

An example of graph data is shown in Figure 3. In this example, the information of node n_1 is condensed into what information it itself holds but also its connections with nearby nodes. These nearby nodes are often referred to as neighbours and also contain their own information. The information for each node can be represented in a feature vector. The hidden feature vector of a node at a layer k can be generalised in Eq. 3 [11].

$$h_v{}^{(k)} = \sum_{u \in N(v)} f(x_v, x^e{}_{(v,u)}, x_u, h_u{}^{(k-1)}) \tag{3}$$

The feature vector is described as hidden because it is within a hidden layer, generally inside a multi-layer perceptron - a common neural network. The sum of this function is over all the neighbours of the node $N(v)$. In this example this would be over the neighbours of n_1: n_2, n_3, and n_4. x_v is the feature

vector of the node n_1 and $x^e_{(v,u)}$ is the edge feature, E, between two nodes say the one shared with n_2. The function also considers the feature vector of this neighbouring node with x_u. Finally ${h_u}^{(k-1)}$ is the feature vector of the main node in the previous layer. The edges themselves can have deeper features. In the case of particle data, the edge will represent a physical trajectory and have spatial information associated with it, such as its angle. The nodes in this example can be understood as the hits, and the edges between the nodes can be understood as the tracks between two layers left by one particle in the detector.

3.2 Graph Neural Networks

Graph Neural Networks (GNNs) operating by passing information through the graph and modifying each node's representation based on its neighbours' representations. This is accomplished through a series of message passing phases in which each node accumulates messages from its neighbours in order to update its representation after sending messages to them depending on its current representation [12]. The subsequent layer of the GNN receives the updated representations as input, and the cycle is repeated until the desired output is attained. This recursive nature will be implemented in the form of repeated executions of Edge Networks and Node Networks which train on and update these graph features. GNNs can be employed to approach generally three problems when it comes to predicting graph data. A graph-level task consists of classifying between graphs as a whole. This can involve differentiating between two molecules represented in graph format. The remaining two tasks focus on classifying nodes and edges within graphs. This paper will focus on the latter to try and determine which group edges are generated by the same particle. GNNs operate in a similar fashion to more common neural networks but by allowing for the flow of data to include the relationships between data points. This will be further discussed in Section 5.5 and how the relationships between nodes are trained on.

4 Quantum Computation

Despite being able to encode the problem in graph data, the magnitude of the task is still at hand. Quantum computation is a possible solution to this, as it can exploit quantum phenomena. Classical computation is orientated around bits, which can take a value of 0 or 1. Quantum computation however is orientated around qubits instead. Qubits have the unique ability to exist in a state of superposition where they can simultaneously represent both 0 and 1. These states can be represented in Dirac notation as seen in Eq. 4 which shows a superposition of states for a single qubit.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{4}$$

When this quantum state is measured, a classical value will be received - either 0 or 1. The coefficients α and β are known as probability amplitudes and their squares are equal to the probability of measuring that state. For example, $|\alpha|^2$ would be the probability of measuring this in the $|0\rangle$ state. Much more memory can be stored on qubits than regular bits, as the amount of memory held by a quantum computer scales exponentially with the number of qubits [13]. Measuring the quantum state however will still collapse it to a classical value. The benefit of this setup is that it allows for all the states to be acted on simultaneously. An example of the power of this is seen in Grover's search algorithm [14] where all the states are operated on to rotate them towards the correct basis to increase the probability of measuring the desired state. This search algorithm provides a quadratic temporal speedup $\mathcal{O}(\sqrt{n})$ over the best classical search algorithm.

4.1 Logic Gates

Analogous to classical computation, there are quantum logic gates which allow for the construction of quantum circuits and the flow of quantum encoded information. Quantum gates are unitary operators which are, unlike classical logic gates, always reversible. Quantum logic gates are represented as matrices and some of the more commonly used gates are defined in the following section.

4.1.1 CNOT Gate

Quantum gates often have classical analogues, the CNOT gate for example is closest to the XOR digital logic gate. The XOR gate returns a bit of 1 if only one of its inputs are 1. The CNOT gate for a 2 qubit system is shown in the matrix below. This logic gate, for a two qubit system, will flip the second qubit if the first qubit is equal to 1. For example, the operations on two possible states in a two qubit system can be seen in Eq. 5. These two qubits can be thought of as a target qubit and a control qubit where if the control qubit has a value of 1 then it'll act on the target qubit. This can be seen in Figure 4 where the $|x\rangle$ qubit would be the control qubit in this example and the $|y\rangle$ qubit would be the target qubit.

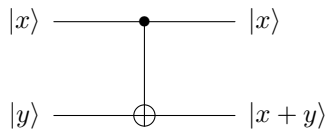


Figure 4: CNOT Gate for a two bit system where the qubits can take a value of either 0 or 1. $|x+y\rangle$ represents modular addition of the two values.

$$\mathbf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$|01\rangle \xrightarrow{\text{CNOT}} |01\rangle \qquad |10\rangle \xrightarrow{\text{CNOT}} |11\rangle \tag{5}$$

4.1.2 Hadamard Gate

Another quantum advantage over classical computing arises from the use of the Hadamard gate. It creates an equal superposition of a given state as seen in Eq. 6. An equal superposition occurs when all states are equally likely of being measured.

$$|0\rangle \xrightarrow{\text{Hadamard}} \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{6}$$

Using the Hadamard gate (hereafter a H Gate) is the first step in the aforementioned Grover's search algorithm as it prepares all the qubits in an equal superposition. A measurement (typically a z-measurement which refers to measuring along the z axis) will return either a classical bit 0 or 1 exactly 50% of the time in the case of Eq. 6 when there is no noise. The action of the H gate can be better understood if a Bloch sphere, a geometric representation of the qubit state, is considered. $|0\rangle$ and $|1\rangle$ can be considered the 'up' and 'down' states respectively as seen in Figure 5. These states after being acted on by the H gate can be understood analogously as the 'left' and 'right' states again respectively. The H gate can then be thought of as a tool to change between the X and Z bases. As previously mentioned, the measurement of a qubit will collapse it into one of its possible states and a classical value is received. This restricts the information that can be gained about a qubit as it can not be measured in multiple directions. For example, if the qubit is prepared in the up or $|0\rangle$ state, a z measurement will return the same value every single time (assuming no experimental noise). However, if the qubit is prepared in the left state, a Z measurement will return a classical bit of either 0 or 1 each 50% of the time, the same result we had discussed earlier. Likewise, an X measurement on a left state will return the same value every time, again assuming no experimental noise. This phenomenon is used in the BB84 protocol [15] which allows for two parties to share a 'key' to decrypt future information sent between them

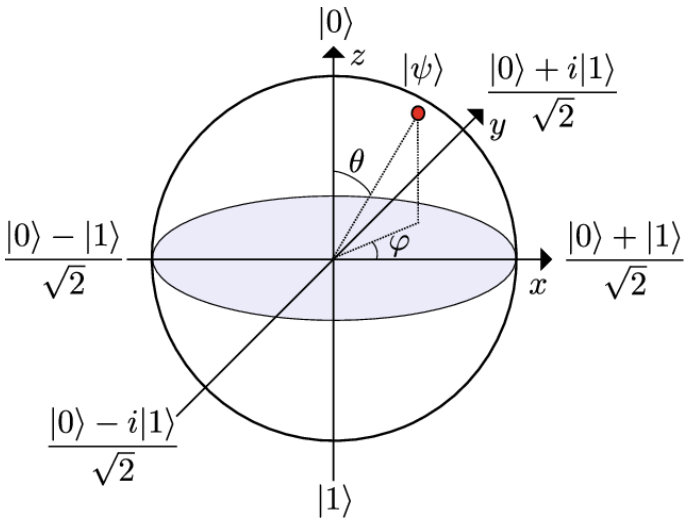


Figure 5: Bloch Sphere of primary qubit states [16]. The surface of the sphere, which is centred at the coordinate system's origin, represents every possible state of the qubit. The sphere's north pole represents the state $|0\rangle$ while its south pole represents the state $|1\rangle$. On the equator of the sphere, states have an equal chance of being in either state $|0\rangle$ or $|1\rangle$. States that are more likely to be in state $|0\rangle$ are located on the upper hemisphere, whereas states that are more likely to be in state $|1\rangle$ are located on the lower hemisphere.

securely. Nobody could 'eavesdrop' on the message without disturbing the system itself. To eavesdrop, a user must measure the qubits and if they chose the wrong axis they would get the wrong answer 50% of the time. They would then have to replace the signal in the message. However, if the two original communicators publicly transferred some of the information shared, discrepancies can be found to discover if the communication was compromised. If not, the rest of the original message can be used as a key to encrypt future messages.

4.2 Bell State

The Hadamard gate and CNOT gate can be used in conjunction with each other to create an entangled state. Prior to the measurement, the H gate turns qubit 0 into an equal superposition as discussed in Section 4.1.2 previously. The CNOT gate operates on this state, however it is only activated when the qubit is in the $|1\rangle$ state. The quantum circuit describing this can be found in Figure 6. This creates the final quantum state seen in Eq. 7 known as a Bell state which is a very common example of an entangled state.

$$\frac{|0\rangle|0\rangle + |1\rangle|1\rangle}{\sqrt{2}} \tag{7}$$

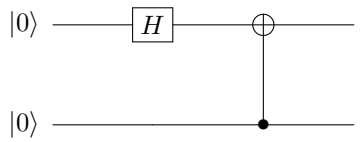


Figure 6: Quantum circuit that generates a bell state with the use of a CNOT and a H gate.

An entangled state is one that cannot be constructed from generic states i.e. $|\psi_1\rangle \otimes |\psi_2\rangle$ where $\otimes$ represents a tensor product. The state itself has a 50% chance of measuring either $|0\rangle|0\rangle$ or $|1\rangle|1\rangle$. However, if only one qubit was measured it would tell us the state of the other and collapse the superposition. For example, if a measurement of the first qubit returned $|1\rangle$, the state as a whole would collapse to the $|1\rangle|1\rangle$. If these two qubits were established in a lab and then separated by a very large distance, this behaviour would still immediately occur if one qubit was measured. This is another quantum phenomenon that can be exploited for a quantum advantage. Processing one qubit can reveal information about multiple qubits. This offers a far greater potential power for computations in superposition on either a polynomial or exponential scale [17].

5 Data

The approach to the track reconstruction problem will focus on previous work which sought to find a way to combine quantum circuits and the classical GNN model [8] originally proposed by the HEP TrkX team [18]. This section will discuss the origins of the data, the pre-processing, the construction of the graphs, and the data flow through the GNN.

5.1 Data Preparation

The available dataset is from the TrackML competition [19] which is comprised of multiple independent events, where each event contains simulated measurements of particles generated in a proton collision at the LHC. The detector is built as a cylinder with silicon slabs which measure the hits of the particles that pass through them. The detector modules are separated into different volumes and each volume there are regular layers where the detectors lie in various modules. This is illustrated Figure 7 where each rectangle corresponds to a volume and the strips within them are the layers labelled from 2 upwards. All simulated detector modules are semiconductor sensors that are built from silicon sensor chips. An example event prior to pre-processing can be found in Figure 8 where the cylindrical structure is more apparent.

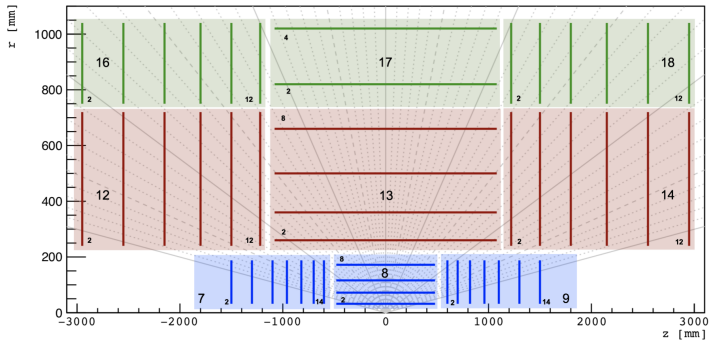


Figure 7: Detector layout with volumes specified in large font and layers specified in smaller font [20]. The barrel region of the detector consists of volumes 8, 13, and 17 and the centre of the detector corresponds to the coordinate $r = 0, z = 0$.

5.2 Pre processing

Only 100 events of the available dataset were utilised because of the time required to train the models. Due to the cylindrical symmetry of this problem, cylindrical coordinates are employed and a reference can be found in Figure 9.

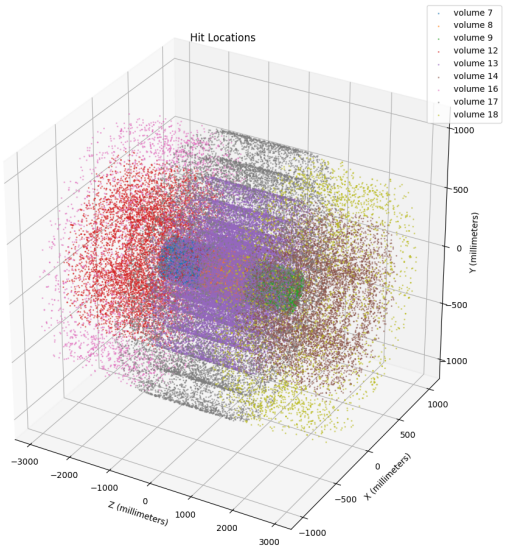


Figure 8: Particle hits from event 1000 in cylindrical detector. Plotting with code adapted from [21].

Only the hits in the barrel region of the detector were considered to further reduce the size of the data. The barrel region refers to volumes: 8, 13, and 17 as seen in Figure 7. There were subsequent cuts made on the magnitude of the particles $|p_t|$. In reality, the detector won't be able to record the magnitude of the momentum of the particle. However as this is simulated data, the particles that exceed this momentum can be readily removed. There is also some activity in the detector that does not arise from a particle passing through it from the collision but rather simulated noise. These hits were also removed. Edge construction could now be completed since the nodes were cleaned. This is completed by taking a hit in a certain layer and comparing it to hits in the next consecutive layer as potential edge candidates. Restrictions are placed on the edges themselves to avoid highly oblique and nonphysical edges. This is done with restrictions on the z intercept, z_0, and the ratio of difference in ϕ and r: $\frac{\Delta\phi}{\Delta r}$. This can be seen in Figure 10. There is also a restriction on the psuedorapidity of the edge. Pseudorapidity [22] is a widely used parameter in particle physics due to its differences being Lorentz invariant and is defined as $\eta = -\ln\tan\left(\frac{\theta}{2}\right)$. Where θ is the angle between the particle's trajectory and the beam axis. The limiting values of these constraints can be found in Table 1. The result of the pre-processing from the same event seen in Figure 8 can be found in Figure 11.

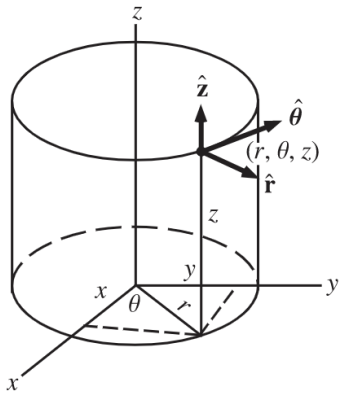


Figure 9: Cylindrical coordinates used in this problem [23]. Note that instead of θ, the term ϕ is used hereafter.

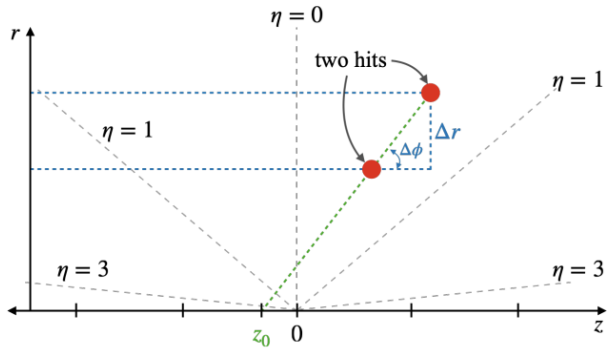


Figure 10: A sketch of the cylindrical coordinate system for particle collisions [24]. The r axis is the projection of the transverse xy plane. η refers to the psuedorapidity.

$\lvert p_t \rvert$	≥ 0.75 GeV
$\frac{\Delta\phi}{\Delta r}$	$\leq 6 \times 10^4 [\frac{rad}{mm}]$
z_0	$\leq 100\ mm$
η	[-5,5]

Table 1: Pre-processing cuts made on the dataset and associated units.

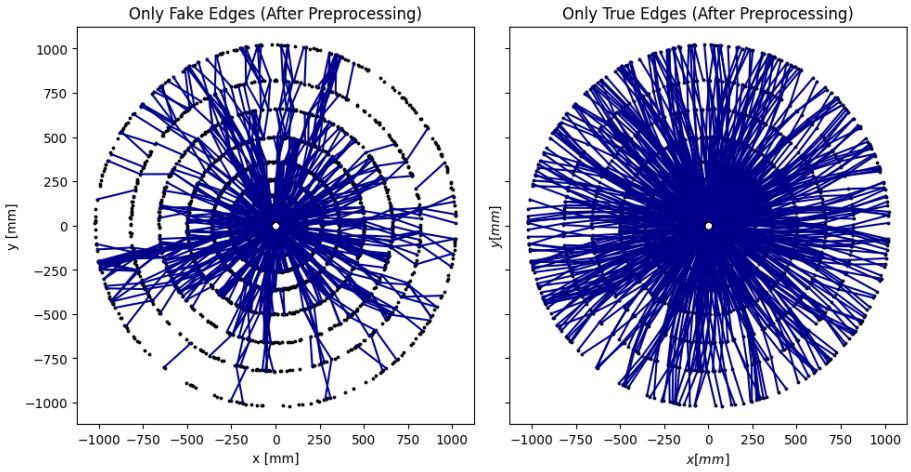


Figure 11: Result of the pre-processing on event 1000 shown on the x-y plane. Fake edges refer to edges that pass the cuts made in Table 1 but don't connect two hits that are made by the path of the same particle. True edges however do refer to the edges that correspond to the path of the same particle. For this event, there were 5162 true edges and 5511 fake edges for 10673 total edges.

5.3 Graph Construction

The data was then condensed into 4 matrices for each event: X, R_i, R_o, and y. The X matrix contains the cylindrical spatial coordinates of each hit: r, ϕ, and z. The R_i and R_o matrices store which coordinates corresponded to which edge and whether they are an input or an output node. The y matrix contains the labels for each edge and whether they are true or fake with a value of 1 or 0 respectively.

5.3.1 X Matrix

For any edge data point there are two spatial coordinates associated with it: the locations of the input and output node. If the X matrix contained duplicates of these coordinates it would lead to double counting which could in turn lead to a model later overfitting as it could be tested on data it has already seen before. To avoid this, the coordinates of all the input and output coordinates were concatenated and then searched through for all unique values. The X matrix has a shape of $N_v \times 3$ where the 3 is due to the r, ϕ, z spatial coordinates and N_v corresponds to the number of hits. The r and z coordinates are normalised for training by dividing them by 1000 whilst ϕ is divided by π as the range of it's values are $-\pi \leq \phi \leq \pi$.

5.3.2 R Matrix

The R_i matrix will first be considered as the process to construct R_o is analogous. For N_v hits and N_e edges, the R_i matrix will have a shape of $N_v \times N_e$. Each edge corresponds to a column in the R_i matrix and each row corresponds to a spatial coordinate of a hit. To create the R_i matrix, one method is to first create an empty matrix of the specified dimensions. Then the matrix can be populated by taking a value from the X matrix and seeing where it comes up in the input part of the edge data. For example, if the 3rd row of the X matrix contains a coordinate that appears as the input node for the 9th edge - there would be value of 1 in the R_i matrix at the 3rd row and 9th column. For R_o, the same is done but with the output coordinates.

5.4 Example Case

An example of the construction of these matrices can be seen in Figure 12. a, b, c, d, e, f, g all represent nodes and the lines represent edges between the nodes. The matrices are shown in Eqs. 8 and 9 and the initial data in Table 2.

Table 2: Example Edge Data.

Input Node	Output Node	y
a	d	1
a	f	0
b	e	1
c	f	1
c	g	0

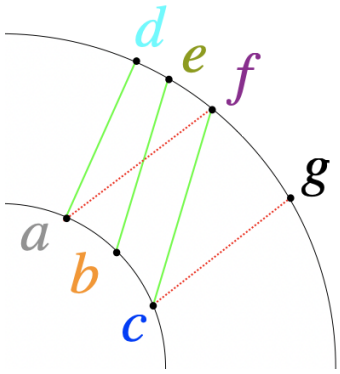


Figure 12: Example edge and node data. The green full lines represent true edges whilst the red dotted lines represent fake edges.

$$X = \begin{bmatrix} a_r & a_\phi & a_z \\ b_r & b_\phi & b_z \\ c_r & c_\phi & c_z \\ d_r & d_\phi & d_z \\ e_r & e_\phi & e_z \\ f_r & f_\phi & f_z \\ g_r & g_\phi & g_z \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{8}$$

$$R_i = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad R_o = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

The X matrix has each node split up into its 3 coordinates. For example a_r, a_ϕ, a_z correspond to the r, ϕ, and z coordinates of node a. Each row in each R matrix relates to the corresponding row in the X matrix. The R_i matrix shows that node c is an input node in the fourth and fifth edge in the edge data found in Table 2 as there are 1 values in the fourth and fifth elements. The sum of either R matrix should return the length of the edge data, which in this case is 5. The labels of the edge data y are represented in the y matrix and determines whether the edge is a true or a fake edge. For example, their are two entries in the 6th row of R_o in the second and fourth columns. This corresponds to the f node being the input node for the second and fourth edge, however only one of these edges can be true. As seen in the y matrix, the second edge is a fake edge and the fourth is a true edge given by the 0 and 1 values respectively.

5.5 Data Flow

There are two different approaches to this problem, one with a quantum graph neural network and a classical graph neural network. The data flow for these two GNNs are the same but differ with how they calculate their predictions on whether an edge is true or fake. The data flow will first be considered and it revolves around the structure seen in Figure 13.

5.5.1 Input Network

The InputNet is a fully connected neural network that operates on the X matrix, producing hidden dimensions for each node. The number of hidden dimensions, denoted by N_h, can be adjusted before training the model. The InputNet's structure is illustrated in Figure 14. Following the InputNet, the hidden values of each node are appended to the node's original coordinates, producing the H matrix shown in Eq. 10. For instance, the first row of X corresponds to node a, so the first row of the H matrix includes the InputNet's output, a_h, along

with the node's initial three-dimensional coordinates $\vec{\mathbf{a}}$. The dimensions of a_h are equal to N_h. Overall due to the InputNet, the model can learn a higher-dimensional representation of the input data. This is due to the model now being able to account for both local and global information about the graph due to the H matrix combining the hidden dimensions with the original coordinates of each node.

$$H = \begin{bmatrix} a_h & \vec{\mathbf{a}} \\ b_h & \vec{\mathbf{b}} \\ c_h & \vec{\mathbf{c}} \\ d_h & \vec{\mathbf{d}} \\ e_h & \vec{\mathbf{e}} \\ f_h & \vec{\mathbf{f}} \\ g_h & \vec{\mathbf{g}} \end{bmatrix} \tag{10}$$

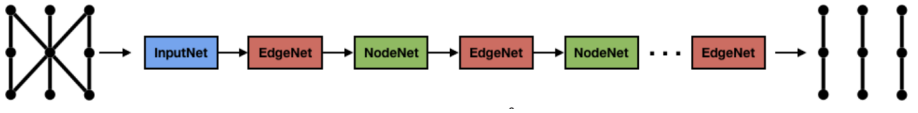


Figure 13: Network structure from the HepTrkX group [25]. There is initially an input transformation layer (InputNet) that prepares the data and then recurrent applications of the Edge Network (EdgeNet) and Node Network (NodeNet) which trains on the edge and node features respectively. The number of iterations specified in training dictates how many consecutive applications of the Edge and Node Networks there are. There will always be a final edge network to return the final predictions of the edges.

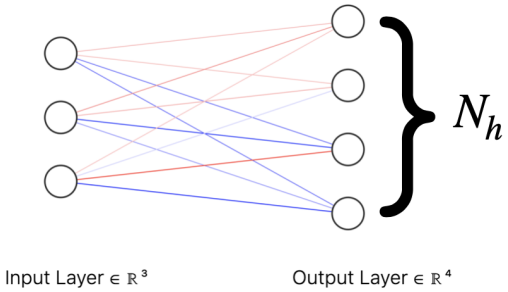


Figure 14: Neural network architecture of InputNet created with Alexander LenNail's software [26]. The activation function for the neurons is ReLU [27]. N_h refers to the number of hidden dimensions of the model. There are three initial neurons to act on the spatial coordinates of each node which corresponds to a row of the X matrix.

5.5.2 Edge Network

The information from the InputNet is then fed into the EdgeNet and NodeNet which are shown in Figure 15. The EdgeNet computes the weights of the edges in the graph data, repeatedly strengthening the most likely ones. The EdgeNet uses the H matrix from the InputNet and the transposes of the R matrices to construct the B_i and B_o matrices as seen in Eqs. 11 and 12. The transpose of a matrix involves an operation that swaps the rows and the columns. This transposing allows for matrix multiplication between a certain row in the R matrix and the corresponding row in the H matrix that both correspond to the same node. This process as a whole allows for the original coordinates and their hidden dimensions to pass through the model, increasing expressibility without the loss of information.

$$R_i^\intercal \times H = B_i \tag{11}$$

$$R_o^\intercal \times H = B_o \tag{12}$$

The B matrix can then be created from combining B_i and B_o as shown in Eq. 15. The final results of these matrices can be interpreted as the initial edge data found in Table 2 but with the additional hidden dimensions appended to each node. This B matrix is then trained on by the EdgeNet. This varies based on whether a CGNN is used or a QGNN. The EdgeNet returns an array of weights of the form $[0,1]^{N_e}$ with each entry corresponding to an edge. A weighting of 0 denotes a fake edge and a weighting of 1 a true edge.

$$R_i^\intercal = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad R_o^\intercal = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

$$B_i = \begin{bmatrix} a_h & \vec{\mathbf{a}} \\ a_h & \vec{\mathbf{a}} \\ b_h & \vec{\mathbf{b}} \\ c_h & \vec{\mathbf{c}} \\ c_h & \vec{\mathbf{c}} \end{bmatrix} \quad B_o = \begin{bmatrix} d_h & \vec{\mathbf{d}} \\ f_h & \vec{\mathbf{f}} \\ e_h & \vec{\mathbf{e}} \\ f_h & \vec{\mathbf{f}} \\ g_h & \vec{\mathbf{g}} \end{bmatrix} \tag{14}$$

$$B = \begin{bmatrix} a_h & \vec{\mathbf{a}} & d_h & \vec{\mathbf{d}} \\ a_h & \vec{\mathbf{a}} & f_h & \vec{\mathbf{f}} \\ b_h & \vec{\mathbf{b}} & e_h & \vec{\mathbf{e}} \\ c_h & \vec{\mathbf{c}} & f_h & \vec{\mathbf{f}} \\ c_h & \vec{\mathbf{c}} & g_h & \vec{\mathbf{g}} \end{bmatrix} \tag{15}$$

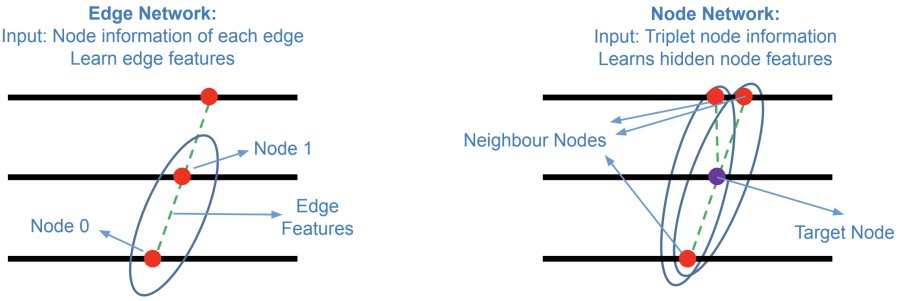


Figure 15: Representation of the Edge Network and the Node Network in the formation of seeds that are comprised of triplets [28].

5.5.3 Node Network

The NodeNet updates the features for every node based on the calculated edge weights. The edge weights from the EdgeNet are fed through the NodeNet to create the weighted R matrices. If the edges in Table 2 were initially weighted at [0.8, 0.3, 0.7, 0.6, 0.4] by the EdgeNet then the initial R matrices can be weighted based on these results. These weighted matrices can be found in Eq. 16.

$$R_{wi} = \begin{bmatrix} 0.8 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0.4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad R_{wo} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0 \\ 0 & 0.3 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0.4 \end{bmatrix} \tag{16}$$

The task of the NodeNet is to train on the hidden node features, and this is done by creating weighted averages of possible input and output nodes. This is analogous to finding the weights for the edges but now instead with the nodes. To find the weighted averages of the nodes, a new set of matrices are defined. These are the M_i and M_o matrices shown in Eq. 19. The results of the network are now written more concisely so [**a**] refers to the array containing [a_h $\vec{\mathbf{a}}$]. Therefore both of the matrices will have a dimension of $N_v \times (N_h + 3)$.

$$R_{wi} \times B_i = M_i \tag{17}$$

$$R_{wo} \times B_o = M_o \tag{18}$$

$$M_i = \begin{bmatrix} 0.8[\mathbf{a}] + 0.3[\mathbf{a}] \\ 0.7[\mathbf{b}] \\ 0.6[\mathbf{c}] + 0.4[\mathbf{c}] \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad M_o = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ 0.8[\mathbf{d}] \\ 0.7[\mathbf{e}] \\ 0.3[\mathbf{f}] + 0.6[\mathbf{f}] \\ 0.4[\mathbf{g}] \end{bmatrix} \tag{19}$$

$$M = \begin{bmatrix} 1.1[\mathbf{a}], \mathbf{0}, [\mathbf{a}] \\ 0.7[\mathbf{b}], \mathbf{0}, [\mathbf{b}] \\ [\mathbf{c}], \mathbf{0}, [\mathbf{c}] \\ \mathbf{0}, 0.8[\mathbf{d}], [\mathbf{d}] \\ \mathbf{0}, 0.7[\mathbf{e}], [\mathbf{e}] \\ \mathbf{0}, 0.9[\mathbf{f}], [\mathbf{f}] \\ \mathbf{0}, 0.4[\mathbf{g}], [\mathbf{g}] \end{bmatrix} \tag{20}$$

It should be noted that the **0** terms, written in bold, in the M_i and M_o matrices will have the same dimensions as the other terms. For example if [**a** has 7 elements in it (4 from N_h and 3 from the spatial coordinates) then a **0** entry will equal [0,0,0,0,0,0,0]. Typically there wouldn't be this many **0** elements but this arises due to the simplicity of the example. These two matrices are combined along with the original H matrix to form the M matrix as seen in Eq. 20. The M matrix contains the original hidden dimensions for the nodes and their new weights as a result of the edge network. This matrix is then passed through the NodeNet layer which again will vary depending on whether a classical or quantum approach is taken. This will return a new H matrix with the same dimensions as the one seen in 10 but with updated hidden dimensions. This updated H matrix is then passed back through the model if more iterations in the training procedure are desired. After all iterations have been completed, there is a final run of the EdgeNet to calculate the edge weights, which constitute the predictions of the model.

6 Training

This section will discuss how the training takes place during the data flow described in Section 5.5. This includes the weighting given to the edges to deal with the class imbalance, how the EdgeNet and NodeNet train on the matrices, and the parameters for training. The B and M matrices are trained on by the EdgeNet and NodeNet respectively in what will be referred to as the EdgeNet layer and the NodeNet layer. The classical approach trains a neural network on these matrices whereas a quantum approach trains a PQC on them.

6.1 Classical Layers

In the classical example these layers are neural networks which can be found in Figures 16 and 17. The output of the NodeNet layer has the same dimensions

as N_h, the hidden dimensions. This allows for the updating of the H matrix which can then be fed back into the EdgeNet and NodeNet. The output of the EdgeNet layer is one value that relates to the edge weighting for the row of the inputted matrix.

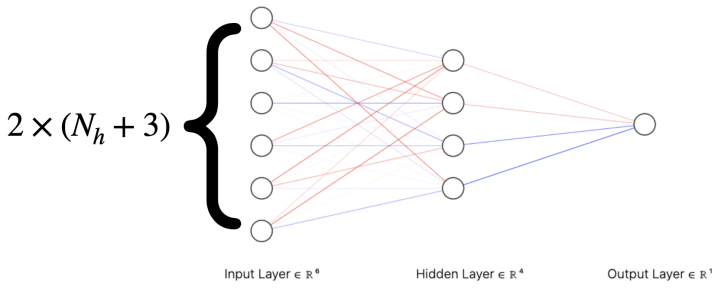


Figure 16: Dense Neural Network which operates on the B matrix in the EdgeNet. The middle layer has the same dimensions as the number of hidden dimensions, N_h with the final output being one number which is the weight of the edge. The activation function functions for the middle and final layer are the hyperbolic tanh function and sigmoid respectively [29].

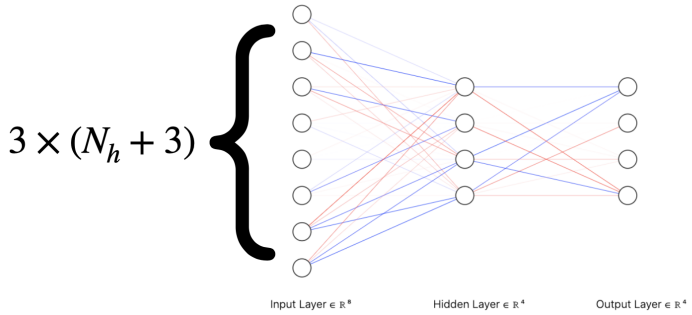


Figure 17: Dense Neural Network which operates on the M matrix in the NodeNet. The middle and last layer has the same dimensions as the number of hidden dimensions N_h and the same activation functions as the EdgeNet. The number of input nodes shown in the diagram is not representative of the actual amount.

6.2 Quantum Layers

For the quantum EdgeNet and NodeNet, PQCs are used instead of conventional neural networks. An example circuit can be found in Figure 18, which would be

used for both the EdgeNet and the NodeNet. The circuit is first initialised with all the $R_y(\theta)$ gates set to random rotations between $[0, 2\pi]$. The matrices would then be passed through an input layer with a ReLU activation function to obtain the values $[0, 1]^{N_Q}$, where N_Q is the number of qubits in the circuit. These values are then normalised using min-max normalisation [30] and multiplied by π so the output is $[0, \pi]^{N_Q}$. This scale is a preference and can be changed, for example the range $[0, 2\pi]$ could also be used. These values are then encoded into the quantum circuit with the rotation gates specified by $R_y(x)$. For example if $N_Q = 4$ and the initial values were $[0, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{\pi}{4}]$, then the rotations of the 4 $R_y(x)$ gates would be set to $[0, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{\pi}{4}]$. There are different approaches to measuring the circuit, but typically a z measurement is applied on each qubit at the end of the circuit. It should be noted that this is a simulated quantum circuit on a classical computer, and the noise associated with the measurement is set to zero artificially. The expectation value of the circuit can then be calculated which leads to 4 classical values in the case of Figure 18 with $N_Q = 4$. These 4 classical values are then passed through a classical dense layer with a sigmoid activation function. For the EdgeNet, this layer will have an output of 1 and for the NodeNet the output will be equal to N_h, as to update the H matrix. The weights and biases of neurons in classical neural network layers are changed analogously to how the quantum circuit is updated. Instead of altering weights and biases, the $R_y(\theta)$ gates' rotation angles are used to update the quantum circuit. The performance of the model is generalised in a loss function. The gradients of the loss function with respect to the circuit parameters are determined to inform these updates to the gates using methods like back propagation. Different circuits will have different numbers of trainable parameters based on their architecture, N_q, and N_L. For example, the number of parameters N_p for the Circuit 10 model can be calculated with $N_p = N_l \times N_q$. Note that both the EdgeNet and the NodeNet can have a certain amount of layers so generally the total number of parameters will be $2 \times N_p$ if they have the same number of layers. The benefit of these repeatable layers is that their parameters can be easily increased without increasing N_q. The drawback of these circuits is that there is typically measurements made on each qubit at the end which can introduce noise into the results. Other circuits based around matrix product states and tree tensor networks seek to condense the information about the quantum state so only one measurement is required.

6.2.1 MPS: Matrix Product States

Another quantum circuit architecture is based on a MPS quantum state classifier which has previously been used to classify quantum data efficiently [32]. An example of a MPS for a 4 qubit system can be found in Figure 19. Matrix product states are useful as they allow for the efficient representation of lowly entangled states. This includes ground states of one-dimensional quantum systems with short-range interactions, like spin chains or Hubbard models [33]. This is because there is often little entanglement among remote locations in these systems and the MPS can capture the state with only a few parameters.

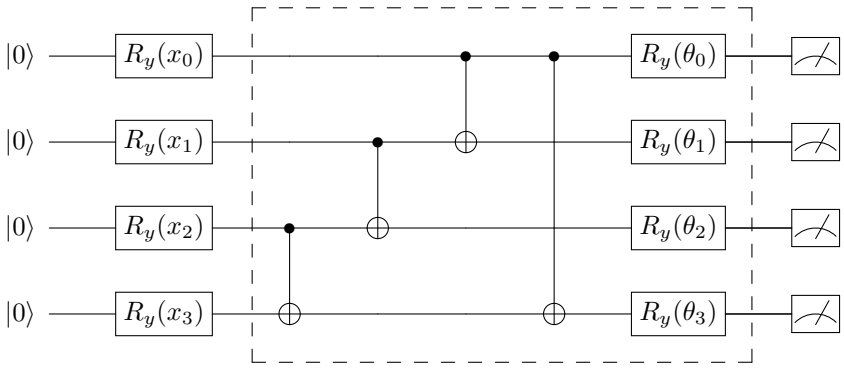


Figure 18: Example PQC labelled as Circuit 10 [31]. The grid section represents one layer, $N_l = 1$, but can be repeated for a more complex circuit with more parametrisable gates.

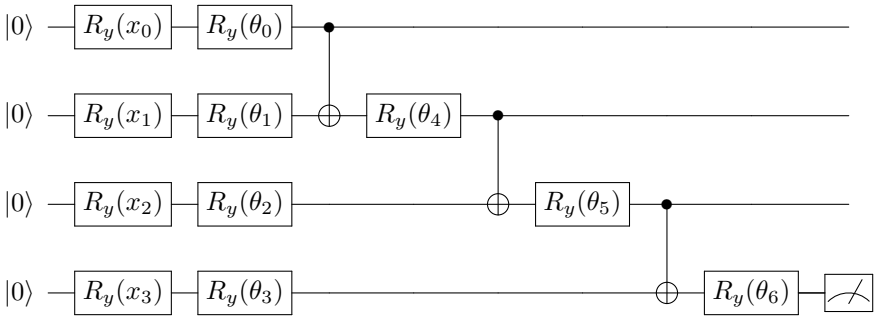


Figure 19: Quantum circuit architecture for a matrix product state. Unlike the circuit in Figure 18, this circuit does not have the structure to be regularly repeated with uniform layers. Note also the measurement only taking place on the last qubit rather than all qubits.

6.2.2 TTN: Tensor Tree Network

Matrix product states can be expanded to higher dimensions and broader categories of networks with tree tensor networks (TTNs). TTNs can represent the state of more sophisticated structures, such as two-dimensional lattices or even more general graphs, in contrast to matrix product states. They have been successfully used for quantum many-body systems [34] where a MPS would perform poorly. TTNs have been used in the past for quantum assisted machine learning (QAML) on classical data such as the MNIST database which consists of handwritten numbers [35]. A similar architecture has been employed for a 4 qubit system as seen in Figure 20. Each qubit in the system can be represented as a tensor, and these tensors can then be arranged in a hierarchical tree structure to create TTNs. Each qubit can be represented by the 'leaves' of the root tensor, which represents the complete system. Both the TTN and

MPS architectures seek to condense information down to reduce the amount of measurements required on the circuit. However, this means the structures themselves are not inherently repeatable like Circuit 10.

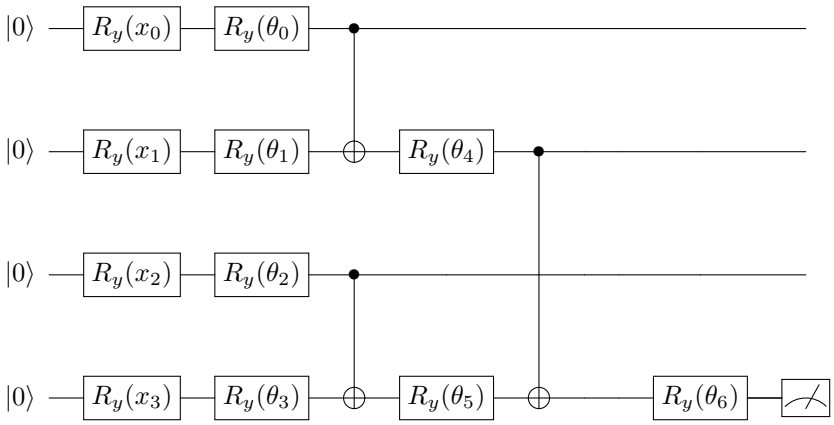


Figure 20: Quantum circuit architecture for a tree tensor network. This circuit also does not have regular repeatable layers like the MPS seen previously. There is also only one measurement on the last qubit as the tensor network is designed to try and condense the information into this final state.

6.2.3 L-VQE: Layer Variational Quantum Eigensolver

A variational quantum eigensolver (VQE) is a quantum algorithm that is typically used to estimate a molecule's ground state energy [36]. In VQE, a classical computer is used to optimise the characteristics of the quantum state in order to reduce energy. This is unlike other methods such as perturbation theory which determines the ground state by considering small shifts in energy from a known system. To employ a VQE, a typically simple 'ansatz' quantum state is created on a quantum computer. The ansatz state is then varied by changing its parameters, and the energy of each variation is calculated using the quantum computer. The classical computer then uses an optimization algorithm, such as gradient descent, to find the set of parameters that minimize the energy. This is a similar idea behind how neural networks operate with the minimisation of the loss function that allows for the optimisation of the parameters in the network. This idea can be implemented in the form a L-VQE which intends to increase the precision and effectiveness of the VQE algorithm by adding additional layers until the desired level of accuracy is achieved [37]. This allows for the construction of a new quantum circuit repeatable architecture as seen in Figure 21.

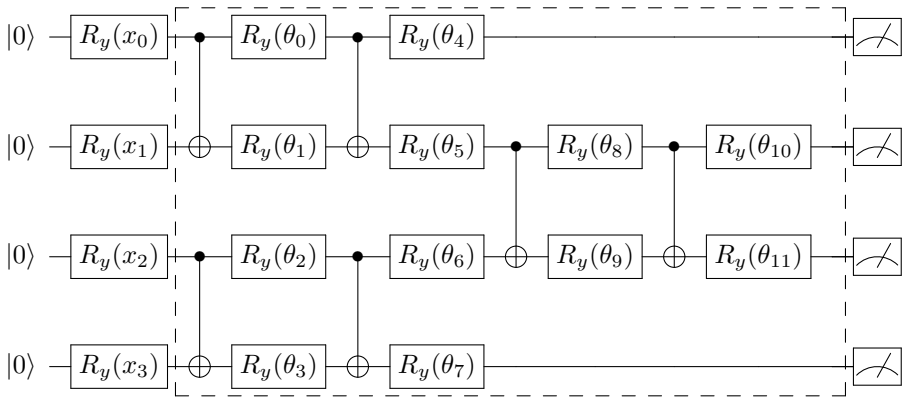


Figure 21: Initial ansatz for L-VQE method for $N_Q = 4$ and $N_L = 1$. Section inside the grid corresponds to repeatable layer [38].

6.3 Weighting

For the dataset, there's an imbalance between the true and fake edges so there is a weighting algorithim deployed during the training so the model doesn't develop a bias towards predicting the larger class. The weight given to each data point from a certain class can be calculated using Eq. 21.

$$w_j = \frac{n_s}{n_c \times n_{sj}} \tag{21}$$

Where w_j is the weight of class j, n_s is the total number of samples, n_c is the number of unique classes, and n_{sj} is the number of samples of class j. The weights used in the training can be found in 3.

Table 3: Results for 100 prepared events

Label	**Count**	**Weighting**
Fake	428346	1.025
True	449887	0.976

6.4 Metrics

In the case of binary classification, the model can predict whether the data belongs to the positive or the negative class whilst the data is only actually part of either the positive or negative class. This leads to 4 possible outcomes of any one prediction by the model. The quantities of these outcomes are represented by a confusion matrix, as seen in Figure 22. This matrix can give valuable information about the competency of the model for different tasks whereas other metrics such as accuracy can sometimes be misleading such as when there is a skew in the size of the data classes.

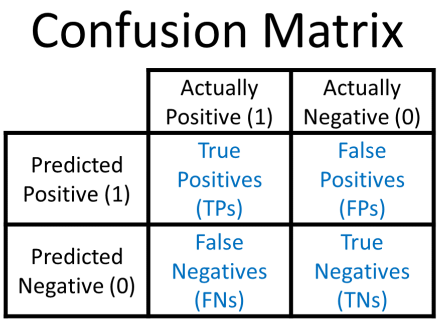


Figure 22: Confusion matrix for predictions of a model on a data set [39]. For example, the model predicting a fake edge when in reality it was a true edge would count as a false negative (FN).

6.5 F1 Score

F1 score is the weighted average of precision and recall. Precision is the ratio of correctly predicted positives to the total number of positive predictions and can be calculated by Eq. 22.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{22}$$

The definitions of the terms in this equation can be found in Figure 22. Recall, also known as sensitivity, is the ratio of correctly predicted positives to the total number of observations that actually belong to the positive class and can be calculated by using Eq. 23.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{23}$$

F1 score, which can be calculated using Eq. 24 takes false positives and negatives into account so is a better measure when there's an imbalance of data between the classes.

$$\text{F1 Score} = \frac{2 \times (\text{Recall} \times \text{Precision})}{\text{Recall} + \text{Precision}} \tag{24}$$

6.6 AUC-ROC Curve

A 'Receiver Operator Characteristic' (ROC) curve is a probability curve that plots how the model performs at different decision thresholds i.e. at what probability a prediction is classed as a positive or a negative classification. To calculate the ROC curve, the true positive rate (TPR) and the false negative rate (FNR) are required. The true positive rate is synonymous with recall as defined in

Eq. 23. The true negative rate, also known as specificity, is the proportion of the negative class that got correctly classified as defined in Eq. 25. The false positive rate (FPR), which is equal to $(1 - \text{Specificity})$, is the proportion of the negative class that was incorrectly classified by the model. The ROC curve plots the TPR against the FPR at a range of threshold values. The 'Area Under The Curve' (AUC) score is a summary of the ROC curve and values range from [0.5, 1]. A value of 0.5 would represent a model randomly classifying the data and 1 would represent a model perfectly classifying the data.

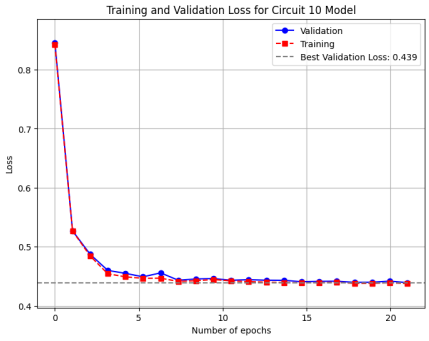
$$\text{Specificity} = \frac{TN}{TN + FP} \tag{25}$$

7 Results

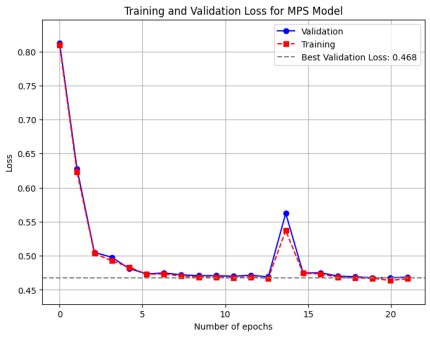
Due to the long training times and time constraints of this investigation, the network architectures were not fully optimised. For example, the optimum number of layers for a certain number of iterations was not determined, but rather the performance of different architectures for the same hyper parameters. Full plots of the recall, precision, and F1 scores for each model can be found in the appendix. Each model was trained on 80 events and validated on 20 for a 100 total. The precise event numbers are also included in the appendix for repeatability of these results. The classical model outperforms the quantum models as seen in Table 4. The quantum models don't offer a speed up advantage currently either due to the large computational cost required in simulating a quantum circuit. For example, the classical model in Figure 23(b) took approximately 58 hours to train and the circuit 10 model Figure 23(a) took 60 hours. All models however quickly approach each of their minimum losses, possibly suggesting there's not enough trainable parameters. To further investigate this, the circuit 10 model was trained again with a more complex architecture. This involves increasing the number of hidden dimensions N_h and the number of layers N_L.

Model	Loss	Accuracy	Precision	Recall	F1	ROC AUC
Circuit 10	0.439	0.775	0.881	0.651	0.748	0.847
MPS	0.469	0.756	0.870	0.617	0.722	0.835
TTN	0.441	0.775	0.880	0.649	0.747	0.854
Classical	0.411	0.786	0.869	0.687	0.767	0.868
LVQE	0.465	0.745	0.803	0.667	0.729	0.830

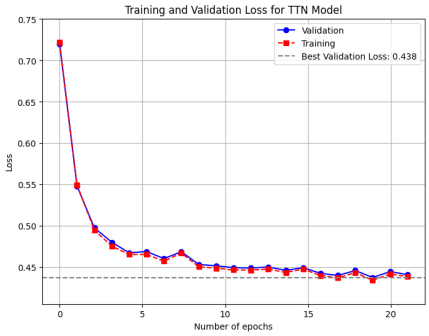
Table 4: Performance results for the five models. The threshold for these metrics was 0.5, which means if the model gave a weighting of 0.6 to an edge it would be classified as a true edge. If the threshold was 0.7 however this 0.6 value would be classified as a fake edge as it is below the threshold.



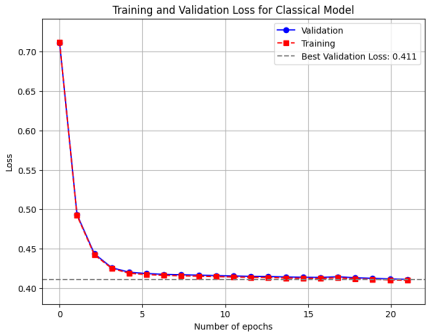
(a) Circuit 10 model from Figure 18.



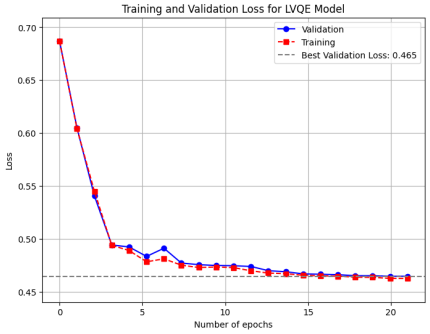
(b) MPS model as seen in Figure 19.



(c) TTN model as seen in Figure 20.



(d) Classical model from Section 6.1.



(e) LVQE model from Figure 21.

Figure 23: Training and validation loss of five separate models. The loss is calculated after each epoch. Training was done over 20 epochs, but there is an initial test of the model before training for 21 data points in total. The Circuit 10 and LVQE models (a,e) each have 3 layers $N_L = 3$ with $N_Q = 4$. The other quantum models seen in (b) and (c) also have $N_Q = 4$ but don't have a repeatable layer structure.

7.1 Further Investigation and Comparison

The more complex Circuit 10 model was compared to another approach that used quantum enhanced support vector machines (SVMs) [40]. This approach utilises a SVM where a kernel function is calculated either on a (simulated) quantum or a classical computer. The quantum and classical results are shown in Tables 5 and 6 respectively. The SVM process doesn't involve a loss that dictates the training, so other metrics are used to compare the performance. The results of the Circuit 10 model training on the smaller dataset with $N_L = 5$ and $N_h = 10$ can be found in Table 7. Across all the metrics, the PQC method for the Circuit 10 model under performed compared to the SVM method. However, a more complex architecture and a smaller data set led to improvements in the loss, accuracy and ROC AUC score of the model as compared to the QGNNs in Table 4. This suggests the quantum circuits didn't have enough expressibility to perform well with the size of the problem. The complex Circuit 10 model still had a worse loss and ROC AUC score compared to the CGNN despite having more trainable parameters. To further explore a possible quantum advantage, the quantum and classical graph neural networks should both be trained again with far more complex architectures. For example, some quantum circuits such as the TTN may perform better in these environments with higher entanglement.

Table 5: Quantum Kernel Performance Metrics. The data was split into 16 separate regions which were then each individually performed on. The error arises from one standard deviation of the corresponding metric.

Metric	**Value**
ROC AUC	0.915 ± 0.006
Accuracy	0.875 ± 0.008
Precision	0.81 ± 0.02
F1 Score	0.89 ± 0.01

Table 6: Classical Kernel Performance Metrics. The uncertainty arises as the same procedure was used as in Table 5.

Metric	**Value**
ROC AUC	0.913 ± 0.006
Accuracy	0.867 ± 0.008
Precision	0.81 ± 0.01
F1 Score	0.88 ± 0.01

Table 7: Circuit 10 model performance on the reduced data set with both the NodeNet and EdgeNet having 20 trainable parameters each. More expressibility was added to the model by setting $N_h = 10$. Further parameters can be found in the appendix.

Metric	**Value**
ROC AUC	0.859
Accuracy	0.783
Precision	0.838
F1 Score	0.772
Loss	0.429

8 Conclusion and Discussion

The validation and training scores for the classical and quantum models under perform when compared to the SVM method. To improve the scores, a higher data sample could be used. For this experiment, only 100 events were used from the TrackML data set [19], whereas 8850 events were available. This restriction was put into place due to the times associated with training the models. Another method would involve increasing the complexity of the models, both in terms of the number of layers and the number of qubits. Due to the classical layers in the training procedure, the dimensions of the information can be controlled to allow for a higher number of qubits. The increase in layers would increase the expressibility of the model which however could in turn lead to overfitting.

8.1 Reducing Expressibility

If future work entails increasing the complexity of the model, the possibility of overfitting needs to be considered and avoided. A regularisation method called dropout is employed in classical neural networks to avoid overfitting. Dropout causes a portion of neurons in a layer to have their values set to 0. This stops the network from being overly dependent on any one characteristic and forces it to learn new reliable ones. This process is also referred to as reducing expressibility, in terms of the model's ability to represent a wide range of complex operations. This is especially necessary in networks with a large number of parameters as they can quickly begin to overfit to the data. By reducing the expressibility of the model, it can become more resistant to input noise and less sensitive to the precise values of individual neurons. This has been seen to significantly reduce a model's classification error [41]. For the quantum circuits used in this experiment, there were not as many parameters seen in typical classical neural networks. However, with the increasing of layers and number of qubits this could quickly become an issue. There are still classical layers embedded in the quantum training process that give the hidden dimensions in the B and M matrices as seen in Section 5.1. However, these consist of generat-

ing hidden dimensions from the quantum circuit and dropout layers here would more likely cause a loss of information. There are different methods to reduce the expressibility of a quantum network analogous to a dropout layer that takes place inside the circuit itself.

8.1.1 Entangling Dropout

The quantity of entangling gates over qubits has a significant impact on the expressibility of a parameterized quantum circuit. By directly reducing the power of entanglement, the circuit's reachable set in the feature Hilbert space is also reduced which can improve the model's performance [42]. Hilbert space being the mathematical space in which the quantum states are represented. Entangling dropout is a possible technique to lessen expressibility by randomly eliminating some entangling gates at certain parameter updates from the circuit during the training phase, similar to the dropout approach for neural networks. Classical dropout layers have a ratio of neurons to drop out. The quantum equivalent could involve two ratios: one for the gate removal rate, and the other for the layer rate. These ratios should be carefully chosen for the problem at hand and often the model performs better when some layers are not included in the drop out procedure. Another method, closer to the classical analogue, would be to set the rotation of gates to 0 at certain parameter updates.

9 Acknowledgements

Thank you to my supervisor, Dr Sarah Malik, for providing guidance and feedback constantly throughout this project. Thanks also to Marcin Jastrzebski for the results acquired with a quantum-enhanced support vector machine and Mohammad Hassanshahi for the aid in using the DIAS cluster.

References

[1] B. Schmidt, "The High-Luminosity upgrade of the LHC: Physics and Technology Challenges for the Accelerator and the Experiments," *J. Phys. Conf. Ser.*, vol. 706, no. 2, p. 022002, Apr. 2016. doi: 10.1088/1742-6596/706/2/022002

[2] S. Amrouche and et Al, "Track reconstruction at LHC as a collaborative data challenge use case with RAMP," *EPJ Web Conf.*, vol. 150, p. 00015, 2017. doi: 10.1051/epjconf/201715000015

[3] P. Cooper, P. Ernst, D. Kiewell, and D. Pinner, "Quantum computing just might save the planet," *McKinsey & Company*, May 2022. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/quantum-computing-just-might-save-the-planet

[4] J. Pequenao and P. Schaffner, "How ATLAS detects particles: diagram of particle paths in the detector," *CERN Document Server*, Jan. 2013. [Online]. Available: http://cds.cern.ch/record/1505342/export/hx?ln=en

[5] P. Astier, A. Cardini, R. D. Cousins, A. Letessier-Selvon, B. A. Popov, and T. Vinogradova, "Kalman filter track fits and track breakpoint analysis," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 450, no. 1, pp. 138–154, Aug. 2000. doi: 10.1016/S0168-9002(00)00154-6

[6] H. Gjersdal, A. Strandlie, and O. Røhne, "Straight line track reconstruction for the ATLAS IBL testbeam with the EUDET telescope." CERN, Geneva, Tech. Rep., 2014. [Online]. Available: https://cds.cern.ch/record/1708349

[7] D. Magano, A. Kumar, M. Kądis, A. Locąns, A. Glos, S. Pratapsi, G. Quinta, M. Dimitrijevs, A. Rivošs, P. Bargassa, J. Seixas, A. Ambainis, and Y. Omar, "Quantum speedup for track reconstruction in particle accelerators," *Phys. Rev. D*, vol. 105, no. 7, p. 076012, Apr. 2022. doi: 10.1103/PhysRevD.105.076012

[8] C. Tüysüz, B. Demirköz, F. Carminati, K. Novotny, F. Fracas, S. Vallecorsa, D. Dobos, J.-R. Vlimant, and K. Potamianos, "arXiv : A Quantum Graph Neural Network Approach to Particle Track Reconstruction," *CERN Document Server*, Jul. 2020. doi: 10.5281/zenodo.4088474

[9] M. Elsing, "ATLAS Track and Vertex Reconstruction for Run-3 and High-Luminosity LHC," *CERN Document Server*, Sep. 2021. [Online]. Available: https://cds.cern.ch/record/2780986

[10] K. Singh, J. Münchmeyer, L. Weber, U. Leser, and A. Bande, "Graph Neural Networks for Learning Molecular Excitation Spectra," *J. Chem. Theory Comput.*, vol. 18, no. 7, pp. 4408–4417, Jul. 2022. doi: 10.1021/acs.jctc.2c00255

[11] J. Hui, "Graph Neural Networks (GNN, GAE, STGNN) - Jonathan Hui - Medium," *Medium*, Jan. 2022. [Online]. Available: https://jonathan-hui.medium.com/graph-neural-networks-gnn-gae-stgnn-1ac0b5c99550

[12] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *arXiv*, Dec. 2018. doi: 10.48550/arXiv.1812.08434

[13] D. Ventura and T. Martinez, "Quantum associative memory," 1998. [Online]. Available: https://arxiv.org/abs/quant-ph/9807053

[14] L. K. Grover, "Quantum Mechanics Helps in Searching for a Needle in a Haystack," *Phys. Rev. Lett.*, vol. 79, no. 2, pp. 325–328, Jul. 1997. doi: 10.1103/PhysRevLett.79.325

[15] S.-K. Chong and T. Hwang, "Quantum key agreement protocol based on bb84," *Optics Communications*, vol. 283, no. 6, pp. 1192–1195, 2010. doi: https://doi.org/10.1016/j.optcom.2009.11.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030401809011316

[16] A. F. Kockum and F. Nori, "Quantum bits with josephson junctions," in *Fundamentals and Frontiers of the Josephson Effect*. Springer International Publishing, 2019, pp. 703–741. doi: 10.1007/978-3-030-20726-7_17. [Online]. Available: https://doi.org/10.1007%2F978-3-030-20726-7_17

[17] R. Jozsa and N. Linden, "On the role of entanglement in quantum computational speed-up," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 459, no. 2036, Jan. 2002. doi: 10.1098/rspa.2002.1097

[18] A. Tsaris, D. Anderson, J. Bendavid, P. Calafiura, and D. Zurawski, "The HEP.TrkX Project: Deep Learning for Particle Tracking," *J. Phys. Conf. Ser.*, vol. 1085, p. 042023, Sep. 2018. doi: 10.1088/1742-6596/1085/4/042023

[19] P. Calafiura, S. Farrell, H. Gray, J.-R. Vlimant, V. Innocente, A. Salzburger, S. Amrouche, T. Golling, M. Kiehn, V. Estrade, C. Germaint, I. Guyon, E. Moyse, D. Rousseau, Y. Yilmaz, V. V. Gligorov, M. Hushchyn, and A. Ustyuzhanin, "TrackML: A High Energy Physics Particle Tracking Challenge," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, Oct. 2018, p. 344. doi: 10.1109/eScience.2018.00088

[20] C. Tüysüz, C. Rieger, K. Novotny, B. Demirköz, D. Dobos, K. Potamianos, S. Vallecorsa, J.-R. Vlimant, and R. Forster, "Hybrid quantum classical graph neural networks for particle track reconstruction," *Quantum Mach. Intell.*, vol. 3, no. 2, pp. 1–20, Dec. 2021. doi: 10.1007/s42484-021-00055-9

[21] wesamelshamy, "TrackML Problem Explanation and Data Exploration," *Kaggle*, Jun. 2018. [Online]. Available: https://www.kaggle.com/code/wesamelshamy/trackml-problem-explanation-and-data-exploration

[22] F.-H. Liu, "Formulation of particle pseudo-rapidity (rapidity) distribution in high-energy pp collision and e+e- annihilation," *Canadian Journal of Physics - CAN J PHYS*, vol. 80, no. 5, pp. 533–540, May 2002. doi: 10.1139/p01-141

[23] E. W. Weisstein, "Cylindrical Coordinates," *Wolfram Research, Inc.*, Oct. 2005. [Online]. Available: https://mathworld.wolfram.com/CylindricalCoordinates.html

[24] C. Tüysüz, C. Rieger, K. Novotny, B. Demirköz, D. Dobos, K. Potamianos, S. Vallecorsa, J.-R. Vlimant, and R. Forster, "Hybrid quantum classical graph neural networks for particle track reconstruction," *Quantum Mach. Intell.*, vol. 3, no. 2, pp. 1–20, Dec. 2021. doi: 10.1007/s42484-021-00055-9

[25] S. Farrell, P. Calafiura, M. Mudigonda, Prabhat, and A. Tsaris, "Novel deep learning methods for track reconstruction," *ResearchGate*, Oct. 2018. [Online]. Available: https://www.researchgate.net/publication/328303900_Novel_deep_learning_methods_for_track_reconstruction

[26] A. LeNail, "NN-SVG: Publication-Ready Neural Network Architecture Schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, Jan. 2019. doi: 10.21105/joss.00747

[27] J. Schmidt-Hieber, "Rejoinder: "Nonparametric regression using deep neural networks with ReLU activation function"," *Ann. Stat.*, vol. 48, no. 4, pp. 1916–1921, Aug. 2020. doi: 10.1214/19-AOS1931

[28] "4th Inter-experiment Machine Learning Workshop," Oct. 2022, [Online; accessed 24. Oct. 2022]. [Online]. Available: https://indico.cern.ch/event/852553/contributions/4057625

[29] S. Sharma, "Activation Functions in Neural Networks - Towards Data Science," *Medium*, Nov. 2022. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[30] J. Han, M. Kamber, and J. Pei, "3 - Data Preprocessing," in *Data Mining (Third Edition)*. Morgan Kaufmann, Jan. 2012, pp. 83–124. ISBN 978-0-12-381479-1. doi: 10.1016/B978-0-12-381479-1.00003-4

[31] C. Tüysüz, C. Rieger, K. Novotny, B. Demirköz, D. Dobos, K. Potamianos, S. Vallecorsa, J.-R. Vlimant, and R. Forster, "Hybrid quantum classical graph neural networks for particle track reconstruction," *Quantum Mach. Intell.*, vol. 3, no. 2, pp. 1–20, Dec. 2021. doi: 10.1007/s42484-021-00055-9

[32] A. S. Bhatia, M. K. Saggi, A. Kumar, and S. Jain, "Matrix Product State Based Quantum Classifier," *arXiv*, May 2019. doi: 10.48550/arXiv.1905.01426

[33] Y. Shimizu, K. Matsuura, and H. Yahagi, "Application of matrix product states to the Hubbard model in one spatial dimension," *arXiv*, Oct. 2013. doi: 10.48550/arXiv.1310.3340

[34] K. Okunishi, H. Ueda, and T. Nishino, "Entanglement bipartitioning and tree tensor networks," *arXiv*, Oct. 2022. doi: 10.1093/ptep/ptad018

[35] M. L. Wall and G. D'Aguanno, "Tree tensor network classifiers for machine learning: from quantum-inspired to quantum-assisted," *arXiv*, Apr. 2021. doi: 10.1103/PhysRevA.104.042408

[36] O. Higgott, D. Wang, and S. Brierley, "Variational Quantum Computation of Excited States," *Quantum*, vol. 3, p. 156, Jul. 2019. doi: 10.22331/q-2019-07-01-156

[37] T. Schwägerl, C. Issever, K. Jansen, T. J. Khoo, S. Kühn, C. Tüysüz, and H. Weber, "Particle track reconstruction with noisy intermediate-scale quantum computers," *arXiv*, Mar. 2023. doi: 10.48550/arXiv.2303.13249

[38] X. Liu, A. Angone, R. Shaydulin, I. Safro, Y. Alexeev, and L. Cincio, "Layer vqe: A variational approach for combinatorial optimization on noisy quantum computers," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022. doi: 10.1109/TQE.2021.3140190

[39] R. Draelos, "Measuring Performance: The Confusion Matrix - Towards Data Science," *Medium*, December 2021. [Online]. Available: https://towardsdatascience.com/measuring-performance-the-confusion-matrix-25c17b78e516

[40] P. Duckett, G. Facini, M. Jastrzebski, S. Malik, S. Rettie, and T. Scanlon, "Reconstructing charged particle track segments with a quantum-enhanced support vector machine," *arXiv*, Dec. 2022. doi: 10.48550/arXiv.2212.07279

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. doi: 10.5555/2627435.2670313

[42] M. Kobayashi, K. Nakaji, and N. Yamamoto, "Overfitting in quantum machine learning and entangling dropout," *ResearchGate*, May 2022. doi: 10.48550/arXiv.2205.11446

A Events

Training Events: ['1066', '1067', '1083', '1082', '1011', '1010', '1058', '1059', '1052', '1053', '1089', '1088', '1071', '1070', '1038', '1039', '1006', '1007', '1094', '1095', '1045', '1044', '1076', '1077', '1048', '1049', '1093', '1092', '1001', '1000', '1099', '1098', '1042', '1043', '1028', '1029', '1061', '1060', '1016', '1017', '1084', '1085', '1055', '1054', '1078', '1079', '1046', '1047', '1072', '1073', '1005', '1004', '1097', '1096', '1051', '1050', '1065', '1064', '1080', '1081', '1012', '1013', '1068', '1069', '1056', '1057', '1062', '1063', '1015', '1014', '1087', '1086', '1041', '1040', '1075', '1074', '1090', '1091', '1002', '1003']

Validation Events: ['1025', '1024', '1032', '1033', '1035', '1034', '1022', '1023', '1031', '1030', '1026', '1027', '1018', '1019', '1021', '1020', '1036', '1037', '1008', '1009']

Training Events for SVM: ['1066', '1067', '1083', '1082', '1011', '1010', '1058', '1059', '1052', '1053', '1089', '1088', '1071', '1070', '1038', '1039', '1006', '1007', '1094', '1095', '1045', '1044', '1076', '1077', '1048', '1049', '1093', '1092', '1001', '1000', '1099', '1098', '1042', '1043', '1028', '1029', '1061', '1060', '1016', '1017', '1084', '1085', '1055', '1054', '1078', '1079', '1046', '1047', '1072', '1073']

Validation Events for SVM: ['1025', '1024', '1032', '1033', '1035', '1034', '1022', '1023', '1031', '1030', '1026', '1027', '1018', '1019', '1021', '1020', '1036', '1037', '1008', '1009']

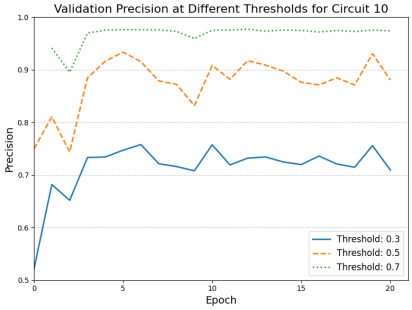
B Parameters and Performance

Table 8: Training parameters for Quantum and Classical GNNs.

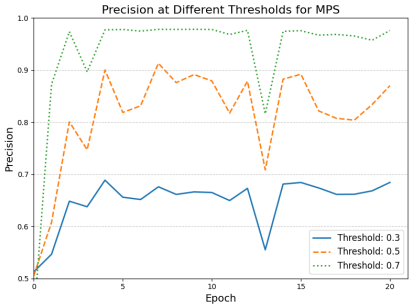
Parameter	**Value**
Batch size	1
Learning rate	0.01
Number of Training Events	80
Number of Validation Events	20
Number of Iterations	3
Number of Epochs	20
Test Every	50
Hidden Dimensions	4
Network	QGNN
Optimizer	Adam
Loss Function	BinaryCrossentropy

Table 9: Training parameters for Circuit 10 in the SVM comparison.

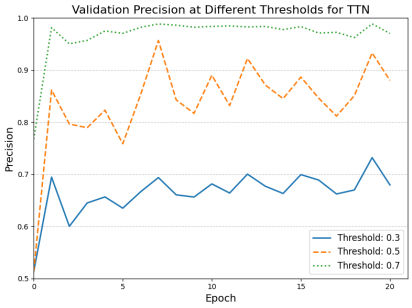
Parameter	**Value**
Batch size	1
Learning rate	0.01
Number of Training Events	50
Number of Validation Events	20
Number of Iterations	3
Number of Epochs	20
Test Every	50
Hidden Dimensions	10
Network	QGNN
Optimizer	Adam
Loss Function	BinaryCrossentropy



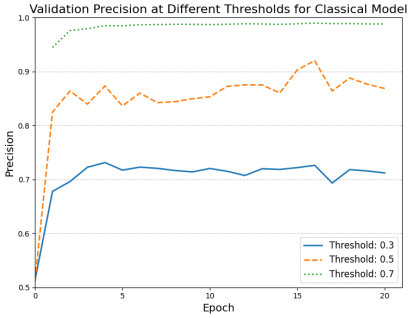
(a) Precision for Circuit 10



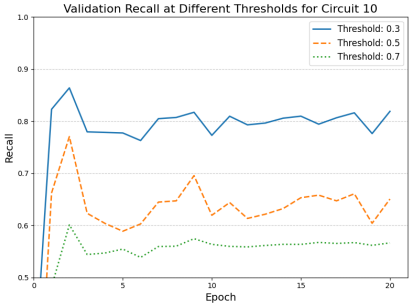
(b) Precision for MPS



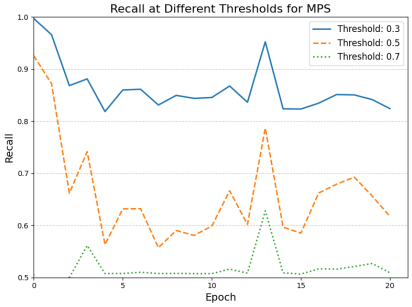
(c) Precision for TTN



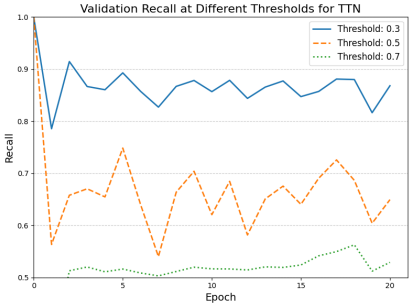
(d) Precision for Classical Model



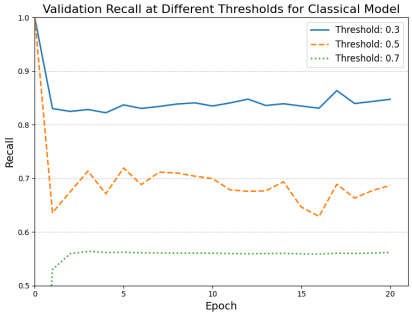
(e) Recall for Circuit 10



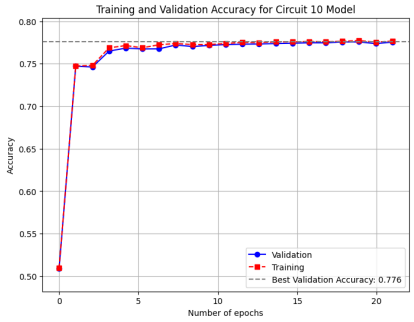
(f) Recall for MPS



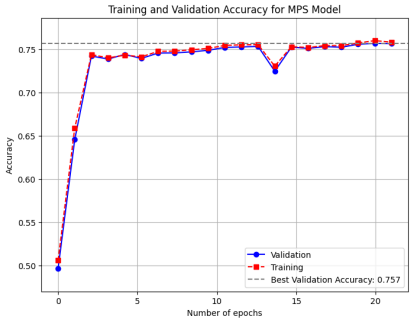
(g) Recall for TTN



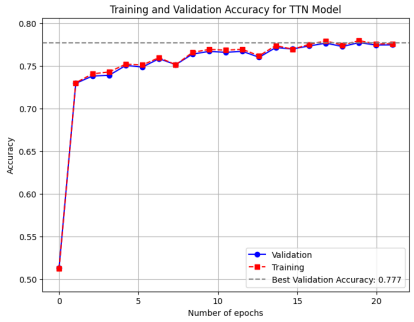
(h) Recall for Classical Model



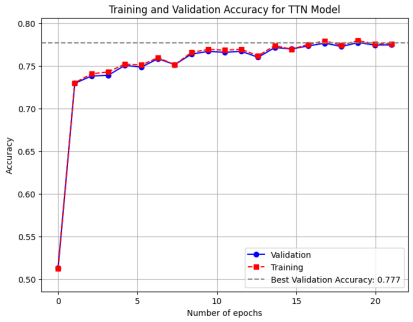
(a) Accuracy for Circuit 10



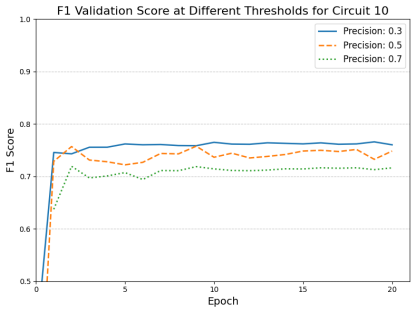
(b) Accuracy for MPS



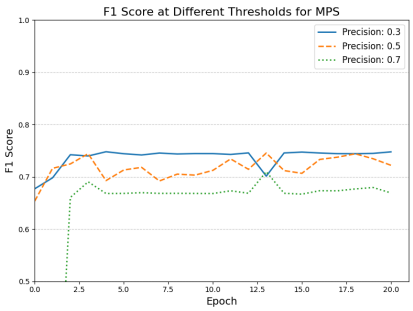
(c) Accuracy for TTN



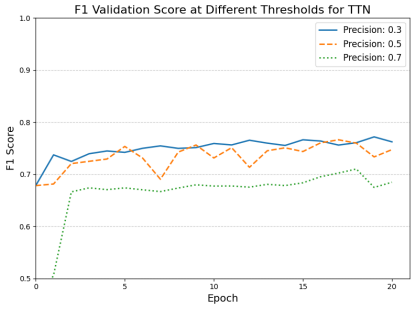
(d) Accuracy for Classical Model



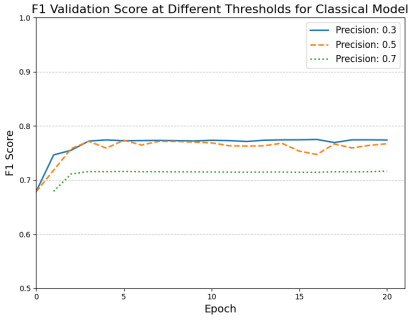
(e) F1 score for Circuit 10



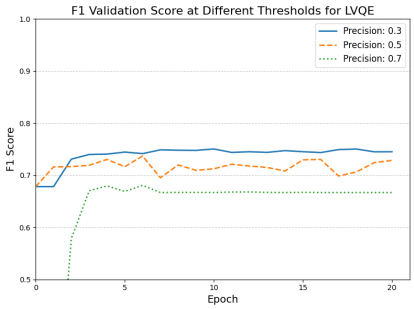
(f) F1 Score for MPS



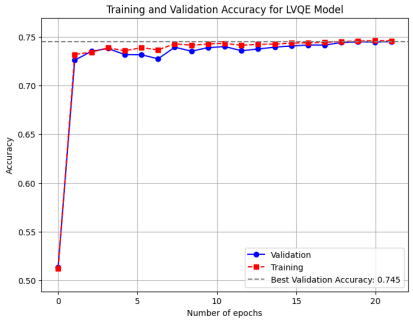
(g) F1 Score for TTN



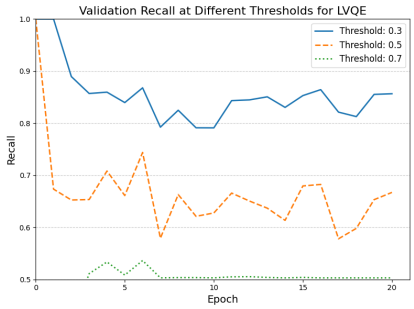
(h) F1 Score for Classical Model



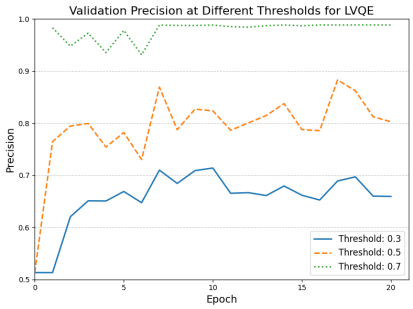
(a) F1 score for LVQE Model



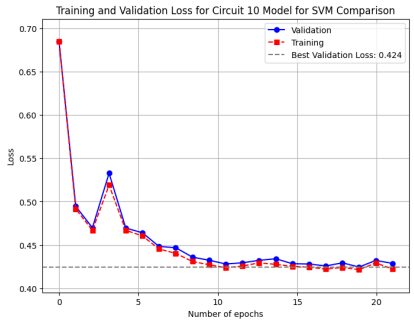
(b) Accuracy for LVQE Model



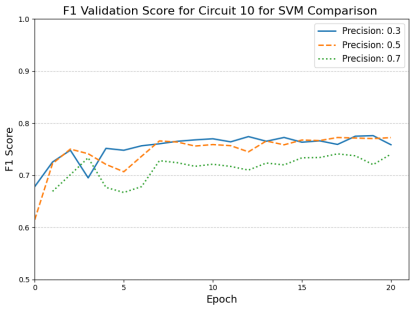
(c) Recall for LVQE Model



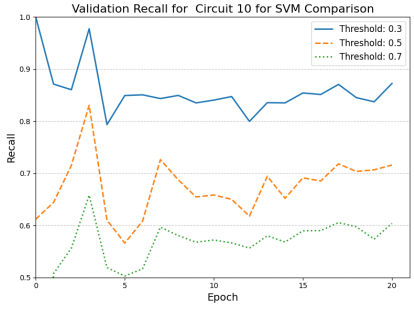
(d) Precision for LVQE Model



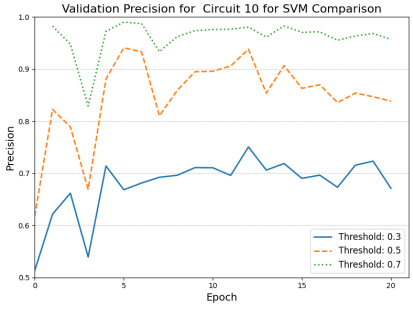
(e) Loss for Circuit 10 Model For SVM Comparison



(f) F1 Score for Circuit 10 Model For SVM Comparison



(g) Recall for Circuit 10 Model For SVM Comparison



(h) Precision for Circuit 10 Model For SVM Comparison