HYBRID QUANTUM-CLASSICAL GRAPH NEURAL NETWORKS FOR PARTICLE TRACK RECONSTRUCTION AT THE LARGE HADRON COLLIDER

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CENK TÜYSÜZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
PHYSICS

AUGUST 2021

Approval of the thesis:

HYBRID QUANTUM-CLASSICAL GRAPH NEURAL NETWORKS FOR PARTICLE TRACK RECONSTRUCTION AT THE LARGE HADRON COLLIDER

submitted by **CENK TÜYSÜZ** in partial fulfillment of the requirements for the degree of **Master of Science in Physics Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Seçkin Kürkcüoğlu
Head of Department, **Physics** _____

Prof. Dr. M. Bilge Demirköz
Supervisor, **Physics, METU** _____

Examining Committee Members:

Prof. Dr. Sadi Turgut
Physics, METU _____

Prof. Dr. M. Bilge Demirköz
Physics, METU _____

Prof. Dr. M. Altan Çakır
Physics Engineering, Istanbul Technical University _____

Assist. Prof. Dr. Osman Barış Malcıoğlu
Physics, METU _____

Assist. Prof. Dr. Heather Gray
Physics, University of California, Berkeley _____

Date: 05.08.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Cenk Tüysüz

Signature :

ABSTRACT

HYBRID QUANTUM-CLASSICAL GRAPH NEURAL NETWORKS FOR PARTICLE TRACK RECONSTRUCTION AT THE LARGE HADRON COLLIDER

Tüysüz, Cenk
M.S., Department of Physics
Supervisor: Prof. Dr. M. Bilge Demirköz

August 2021, 107 pages

Particle collider experiments aim to understand Nature at small scales. Particle accelerators, such as the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN), collide particles at high rates (MHz) and high energies (TeV) in order to probe such small scales. High collision rates may bring many computational challenges. One of these challenges is particle track reconstruction, which is the task of distinguishing the trajectories of charged particles passing through the detector. The upcoming High Luminosity upgrade of the LHC is going to increase the collision rates and require more computational resources. Particle track reconstruction algorithms will also be under much more stress, as the current algorithms are scaling worse than quadratically. This work presents a hybrid Quantum-Classical model to solve the particle track reconstruction problem by combining novel Graph Neural Networks with Quantum Neural Networks that are compatible with Noisy Intermediate Scale Quantum (NISQ) computers. Results indicate that the hybrid model can match the performance of the classical model within the limits of 16 qubits and 16 hidden dimensions.

Keywords: particle track reconstruction, quantum variational algorithms, machine learning

ÖZ

BÜYÜK HADRON ÇARPIŞTIRICISI'NDA PARÇACIK İZİ YAPILANDIRMASI İÇİN HİBRİT KUANTUM-KLASİK ÇİZGE SİNİR AĞLARI

Tüysüz, Cenk
Yüksek Lisans, Fizik Bölümü
Tez Yöneticisi: Prof. Dr. M. Bilge Demirköz

Ağustos 2021 , 107 sayfa

Parçacık çarpıştırma deneyleri doğayı küçük boyutlarda anlamayı hedefler. Avrupa Nükleer Araştırma Merkezi'nde (CERN) bulunan Büyük Hadron Çarpıştırıcısı (BHÇ) gibi parçacık hızlandırıcıları küçük boyutları incelemek için parçacıkları yüksek frekans (MHz) ve enerjide (TeV) çarpıştırır. Yüksek çarpışma frekansları pek çok bilgisayımsal zorluğu beraberinde getirmektedir. Bu zorluklardan bir tanesi ise çarpışma sonucu açığa çıkan parçacıkların rotasının belirlenmeye çalışıldığı parçacık izi yapılandırma problemidir. Şu anda BHÇ üzerinde çalışmaları sürmekte olan yüksek ışınım geliştirmeleri ise parçacık çarpışma oranlarını artıracak ve bilgisayım zorluklarını daha da büyüyecektir. Şu anda kullanılmakta olan parçacık izi yapılandırma algoritmaları ikinci dereceden daha kötü bir ölçeklenmeye sahip oldukları için de bu durumdan fazlasıyla etkilenecektir. Bu çalışma parçacık izi tekrar oluşturma problemini çözmek için yenilikçi Çizge Sinir Ağları'nı Kuantum Sinir Ağları ile birleştiren, Gürültülü Orta Ölçekli Kuantum (NISQ) bilgisayarla uyumlu, bir hibrit Kuantum-Klasik model sunmaktadır. Sonuçlar 16 kübit ve 16 gizli uzay boyutuna sahip hibrit modelin klasik

model ile benzer sonuçlar verebildiğini ortaya koymaktadır.

Anahtar Kelimeler: parçacık izi yapılandırma, kuantum değişimsel algoritmalar, yapay zeka

to infinity and beyond

ACKNOWLEDGMENTS

I would like to start by thanking my supervisor Prof. Dr. M. Bilge Demirköz, who thought me a lot about both science and life. My academic journey started with her in 2016 and now looking back, I can see all the valuable experiences I had in her group. She believed in me to switch to a new field during my master's and I hope I did not let her down, for this I can't express my gratitude enough.

I would like to thank Carla Rieger, Dr. Daniel Dobos, Dr. Karolos Potamianos, Dr. Sofia Vallecorsa, Dr. Jean-Roch Vlimant, Dr. Federico Carminati, Dr. Fabio Fracas and Dr. Alessandro Roggero for valuable discussions. I would like to thank Dr. Kristiane Novotny separately for all the support, guidance and friendship, which helped me tremendously throughout the thesis. I would like to thank my colleagues and friends from METU IVMER and especially Selen Akçelik, Raheem Hashmani, Uğur Kılıç and Erinç Kılıç for their valuable discussions and support.

I would like to thank my friends Çağan Yüksel and Feyza Nur Çalışkan for their friendship and support. I would like to thank Ezgi Taş for the joy she brought to the boring routine of my life and being there whenever I need. Last but not least, I would like to thank my family who supported me no matter what and never questioned my decisions.

I would like to express my gratitude for all the open science supporters from open-source developers to researchers who submit their work to arXiv. I believe in the importance of breaking the barriers in science and include every person equally regardless of their gender, religion or origin.

I would like to acknowledge support of TÜBİTAK through the 2210-E scholarship. This work was supported by Presidency of Strategy and Budget and travels to CERN were supported by Turkish Atomic Energy Authority (TAEK) (Grant No: 2017TAEKCERN-A5.H6.F2.15 and 2020TAEK(CERN)A5.H1.F5-26). Part of this work was conducted at "*iBanks*", the AI GPU cluster at Caltech. I acknowledge

NVIDIA, SuperMicro and the Kavli Foundation for their support of "*iBanks*". I acknowledge the use of IBM Quantum services for this work. The views expressed are those of the author, and do not reflect the official policy or position of IBM or the IBM Quantum team.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Networks
ATLAS	A Toroidal LHC Apparatus
AUC	Area Under the ROC Curve
BP	Barren Plateau
CERN	European Organization for Nuclear Research
CKF	Combinatorial Kalman Filter
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
FC	Fully Connected
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GNN	Graph Neural Network
GPU	Graphical Processing Unit
HEP	High Energy Physics
HHL	Harrow-Hassidim-Lloyd
HL-LHC	High Luminosity Large Hadron Collider
HNN	Hybrid Neural Network
IEC	Information Encoding Circuit
LHC	Large Hadron Collider
LSTM	Long Short Term Memory
ML	Machine Learning

MLPF Machine Learned Particle Flow

MPS Matrix Product State

NISQ Noisy Intermediate Scale Quantum

NN Neural Network

PQC Parametrized Quantum Circuit

QA Quantum Annealing

QC Quantum Computer

QCD Quantum Chromodynamics

QGNN Hybrid Quantum-Classical Graph Neural Network

QNN Quantum Neural Network

QPU Quantum Processing Unit

QUBO Quadratic Unconstrained Binary Optimization

QV Quantum Volume

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic

TN True Negative

TP True Positive

TPR True Positive Rate

TPU Tensor Processing Unit

TTN Tree Tensor Network

VQA Variational Quantum Algorithm

VQC Variational Quantum Classifier

VQE Variational Quantum Eigensolver

CHAPTER 1

INTRODUCTION

Particle accelerator experiments aim to understand the nature of particles by colliding groups of particles with each other (or with fixed targets) at high energies and try to observe the creation of particles and their decays. The Large Hadron Collider (LHC) [1] at the European Organization for Nuclear Research (CERN) hosts four major experiments at interaction points that each focus on different problems in High Energy Physics (HEP). Both the LHC and the experiments come with many hardware and software challenges.

Particle track reconstruction is one of the problems that is computationally challenging for most HEP experiments due to its combinatorial nature. In this problem, the aim is to identify unique trajectories of each outgoing particle that appear after particle collisions. Although traditional approaches can handle the current rate of particles, they are struggling with the transition to High Luminosity Large Hadron Collider (HL-LHC) [2]. HL-LHC is the name used for the next era of the current LHC, and is expected to have more particles in the beam by 2027 [3]. Therefore, there is a need for new algorithms that can handle increased luminosity and also have better scaling to support future applications.

Graph Neural Network (GNN) [4] approaches bring a novel machine learning perspective to solve the particle track reconstruction problem [5, 6]. Their use has already gained lots of interest in recent years and continues to extend to similar problems [7]. Quantum Computer (QC) approaches have also gained popularity, as one can translate parts of the tracking problem to be solved by quantum search or annealing methods [2, 8–10]. In this work, we investigated the possibility of combining both paradigms and implemented a Hybrid Quantum-Classical Graph Neural Network (QGNN).

The proposed QGNN model is trained and evaluated using the publicly available TrackML dataset [11], which has been widely used by the particle track reconstruction community in recent years [12] and compared against a classical model [5] to investigate possible advantages. Results of this work are important for several reasons. First, this work implements a flexible hybrid QGNN algorithm that can be applied to particle track reconstruction and many other graph learning tasks. Second, it presents results that would help improve the current understanding of Quantum Neural Networks (QNNs), which is very limited. Finally, it provides valuable insight for future Hybrid Neural Network (HNN) implementations.

The outline of the thesis is as follows; In Chapter 2, we start by presenting a background for HEP experiments at LHC and continue by defining the particle track reconstruction problems. Then, we give the current status and recent approaches to solve this problem. In Chapter 3, we introduce QC along with Variational Quantum Algorithms (VQAs). In Chapter 4, we present details of the proposed model from how it is built to how its performance is evaluated. In Chapter 5, we present our results along with an extended discussion of our findings. Finally, in Chapter 6 we give an overview and conclusion of the thesis.

CHAPTER 2

PARTICLE TRACK RECONSTRUCTION

2.1 The Large Hadron Collider

European Organization for Nuclear Research (CERN) hosts the Large Hadron Collider (LHC), which is world's largest and highest energy particle collider, in Geneva at the border of Switzerland and France [1]. The circular particle accelerator has a 27 km circumference and is designed to accelerate both protons and lead-ions to a center-of-mass energy of 7 TeV and 13 TeV, respectively. The purpose of the LHC is to answer fundamental questions in physics through high energy particle collisions. The LHC hosts four major experiments, each focusing on different properties of particle collisions. While the A Toroidal LHC Apparatus (ATLAS) and Compact Muon Solenoid (CMS) collaborations study the Higgs boson and investigate new physics, the ALICE experiment investigates heavy-ion collisions and tries to understand the state of matter and properties of quark-gluon plasma just after the Big Bang. The LHCb collaboration investigates the difference in the amount of matter and antimatter. These experiments are placed at four points across the LHC beam and operate independently.

Accelerating particles to such high energies is not a simple task. Particles go through several stages of acceleration, before being fed to the LHC to reach the desired energies. If we take the proton acceleration as an example; protons start their journey in a bottle of Hydrogen gas. An electric field is used to separate electrons from H_2 molecules. After the separation, H^+ ions or protons are ready to be accelerated. First, they go through LINAC 2 which is a linear accelerator that accelerates protons to 50 MeV. Then, they are fed to the Proton Synchrotron Booster to reach 1.4 GeV. The acceleration continues with Proton Synchrotron to reach 25 GeV and Super Proton

Synchrotron to reach 450 GeV before the final stage, LHC, that accelerates protons to 6.5 GeV. All of these stages, along with the four experiments of the LHC can be seen in the the general layout of the CERN accelerator complex in Fig. 2.1.

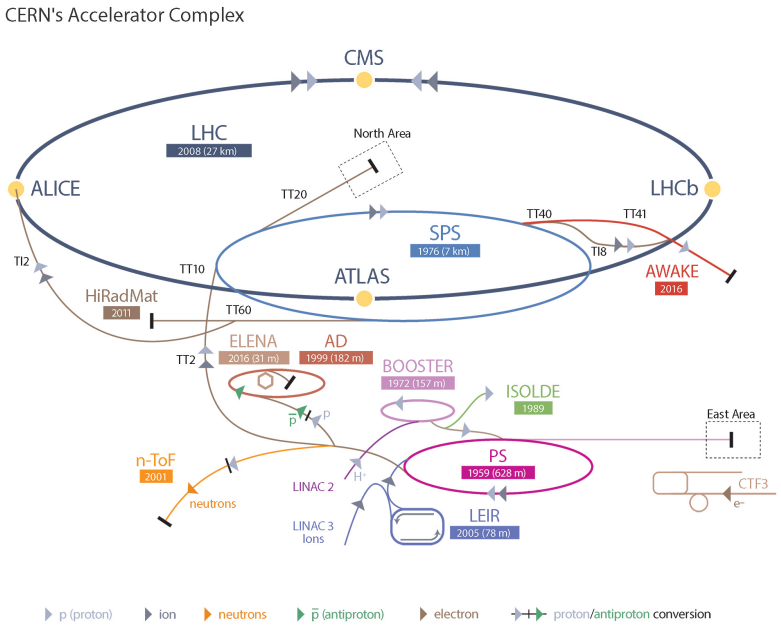


Figure 2.1: CERN's accelerator complex. Several stages of the LHC is given with different colors. The main LHC beam line is shown with the dark blue circle and the four major experiments are marked with yellow circles. Figure is retrieved from [13].

The LHC accelerates particles to high energies in groups referred to as bunches that are separated in time. Two beams contain bunches and travel in the opposite direction. LHC was designed to have 2808 bunches in a proton beam, each containing 1.1×10^{11} protons corresponding to a luminosity of $10^{34} cm^{-2} s^{-1}$. Luminosity is the unit of number of particles per unit area per second. All bunches are separated in 25 ns intervals which corresponds to a 40 MHz collision frequency, which is also called bunch crossing [1].

At the centers of the four experiments the two beams are brought together. These points that are referred to as interaction points. Head-on collisions with a minimal angle are produce at the interaction points. This angle is called crossing angle (θ_C) and it has a small range between 150-200 µrad [1]. The majority of the particles pass through these points, that allows the bunches to be reused to produce new collisions.

An illustration of a bunch crossing is displayed in Fig. 2.2.

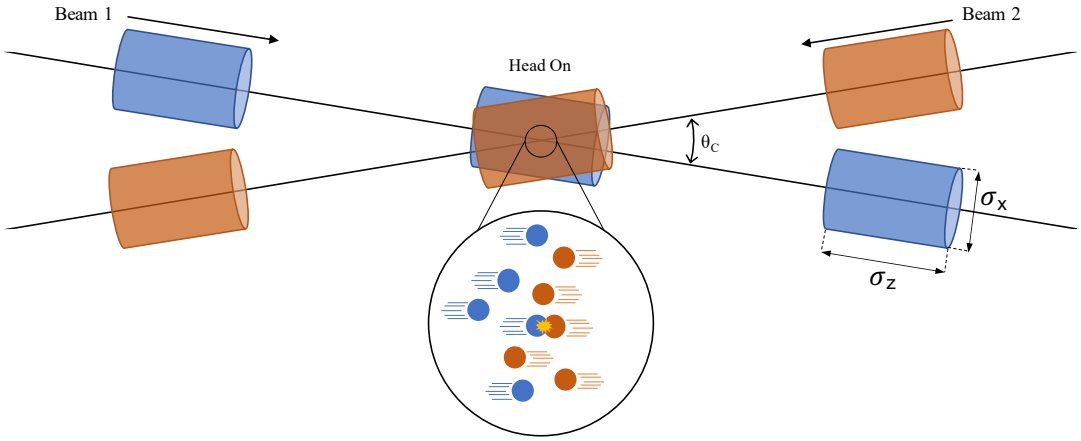


Figure 2.2: An illustration of a bunch crossing at the LHC. Cylinders represent bunches. The crossing angle (θ_C) is exaggerated for visual purposes. The bunch length (σ_z) is ~7.5 cm and the bunch width (σ_x) is ~16 μm at the interaction points. Figure is adapted from [14].

A bunch crossing is called an event that typically includes multiple proton-proton collisions. There are 3 types of reconstructed spatial locations, called vertices, at these experiments. The first is the primary vertex, which results from rare high energy or high momentum transfer interactions. The second is called a pile-up vertex and is a result of frequent soft Quantum Chromodynamics (QCD) interactions (low energy inelastic interactions). Only few hundred events out of the possible 40 million collisions/sec have a high enough energy primary vertex that makes it interesting to analyze. The abundance of pile-up vertices make distinguishing primary vertices in an event harder. The average amount of pile-up vertices in a bunch crossing is referred to as the pile-up ($\langle\mu\rangle$) at LHC. The last type is called secondary vertices as they are the result of a decay process of long-lived particles produced by a primary or pile-up interaction. An illustration of these vertices is given in Fig. 2.3.

Particle collider experiments consist of several parts, each focusing on a different property of particles to measure. They are called to have an onion shape due to their layered structure. Silicon tracking detectors are generally located near the collision center and focus on measuring the trajectory of particles. Outside the tracking detectors, there are several types of calorimeters to measure the energies of particles. Muon

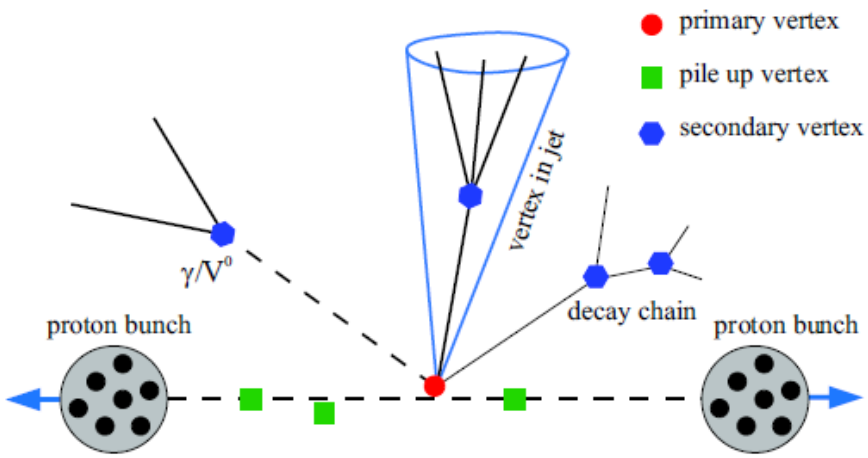


Figure 2.3: An illustration of primary (red circles), pile-up (green squares) and secondary (blue hexagons) vertices at the LHC. Figure is retrieved from [14].

detectors are placed at the outmost layers, since almost all muons travel through the inner layers without disintegrating. In addition to these detectors, collider experiments contain magnets that produce magnetic fields on the order of several Teslas. The magnetic field bends the charged particles, and the bending radius is used to estimate the momentum of particles. As an example to these experiments, an overview of the CMS experiment and its components can be seen in Fig. 2.4.

These experiments have been the ultimate tools to test particle theories, most importantly the Standard Model. Although the discovery of the Higgs boson by the ATLAS [16] and the CMS [17] experiments is mostly known in the media, they showed and still continue to show success in many occasions, discovering and precisely measuring many types of particles and their decays [18]. However, a discovery is never an end point in physics, there are still more theories to test and more to discover in particle physics. Therefore, there is always a need for particle accelerators that reach to higher energies as well as luminosities to increase the statistics.

Currently, the LHC is going through an upgrade phase to have increased luminosity and will be named High Luminosity Large Hadron Collider (HL-LHC) [3]. The major goal of HL-LHC is to increase the intensity of the beam by reducing its size in order to have more collisions. This upgrade involves major changes to LHC subsystems from more powerful magnets to new superconducting links that can carry

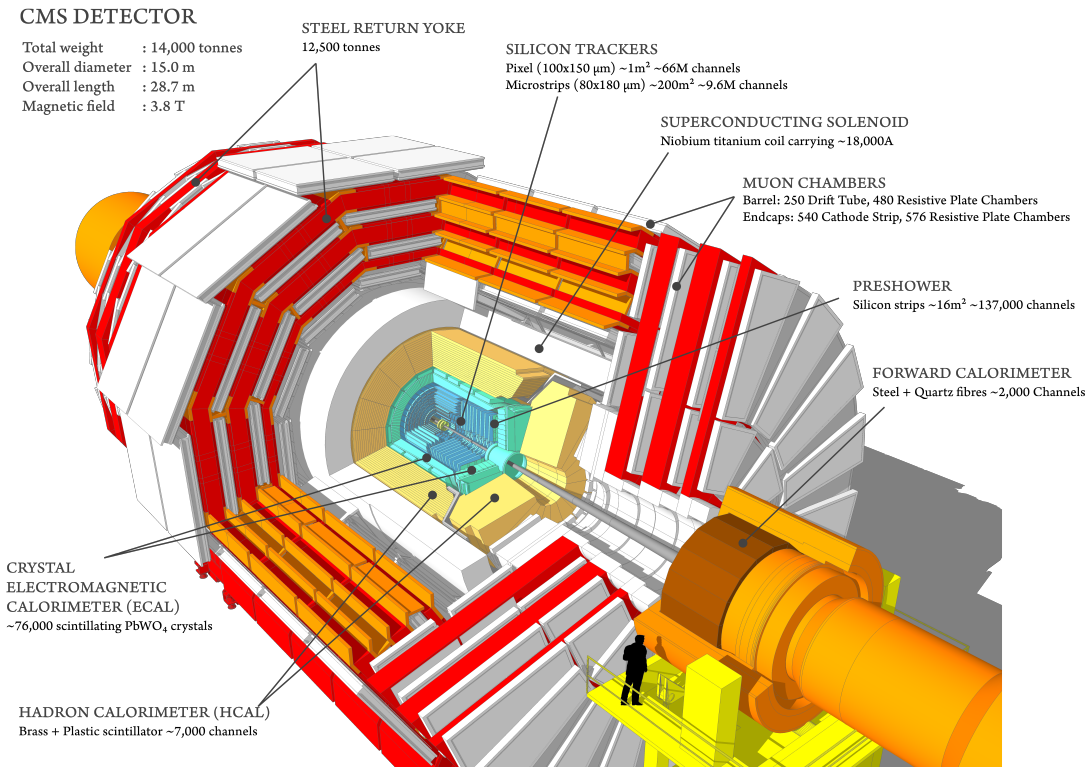


Figure 2.4: Subsystems of the CMS detector [15]. Subsystems are placed such that the granularity and precision of the detectors decrease, while the density increases from inside to outside, because the track density falls as $1/r^2$.

more current through cables. Another key component is the upgrade to the injectors, which increases the bunch population while maintaining the beam size. HL-LHC is expected to be completed by 2027, resulting in a factor of 10 increase in the luminosity compared to design luminosity of the LHC, which was $10^{34} cm^{-2} s^{-1}$. With the increased luminosity, total number of events will be increased and statistical errors in all analyses will decrease significantly. However, this comes with challenges in both hardware and software. In this work, we investigate one of the computational challenges, which is called particle track reconstruction.

2.2 Problem Definition

Each component of particle collider experiments are complex and very large teams are needed to design, manufacture and also operate them. The silicon tracking detector

that we consider in this thesis is no exception. Generally, the overall layout of the tracking detectors are similar. The ATLAS inner detector, shown in Fig. 2.5, can be viewed as an example layout of tracking detectors. Pixel detectors are located in the innermost section to be as close as possible to collision center. On the outside, there are trackers that wraps the pixel detectors. These are referred to as barrel trackers due to their shape resembling a barrel. There are also disk-shaped trackers on the outer layers located along the beam axis. These trackers are called end-cap trackers due to their location.

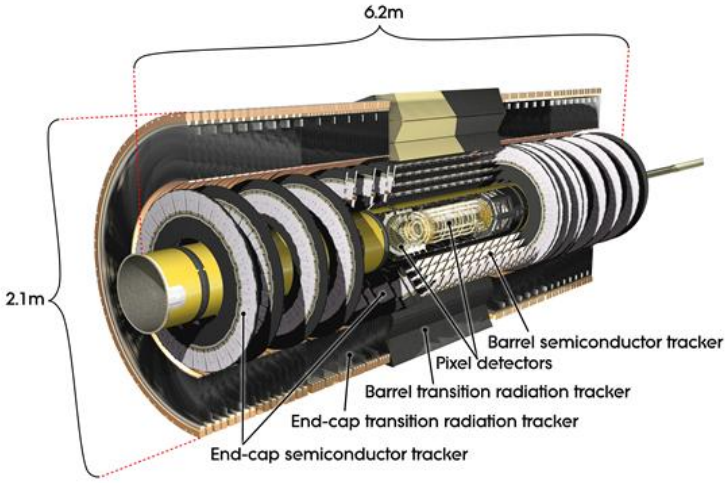


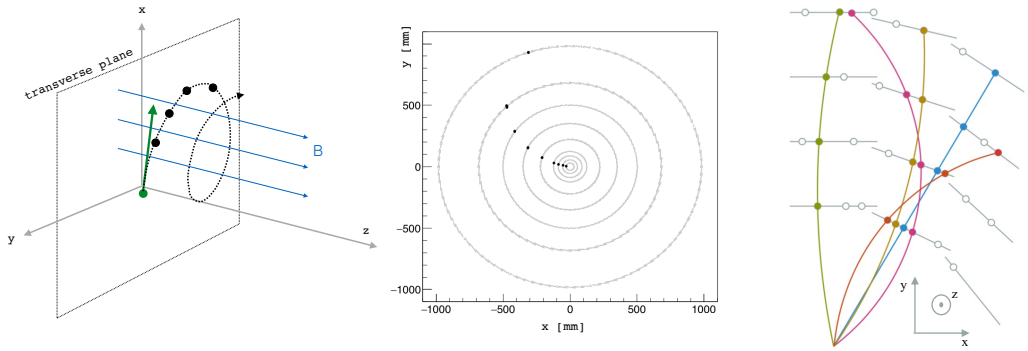
Figure 2.5: Subsystems of the ATLAS inner detector. 4 layers of pixel detectors are located at the center and 4 layers of semiconductor trackers surround them on the outside. The transition radiation tracker wraps the semiconductor tracker. These 3 types of detector form the barrel region due to their geometry. There are 9, disk-shaped, semiconductor tracker layers on each side and another transition radiation tracker located away from the center along the beam axis (end-cap region). Figure is retrieved from [19].

Charged particles bend under a magnetic field, if the magnetic field has a component orthogonal to the momentum vector of the particle. This is one of the reasons for the need of a sophisticated tracking detector geometry. Under a static magnetic field B(r), a particle with momentum p and signed charged q follows the trajectory governed by Eq. 2.1 [20]. The bending causes particles to follow a helix-shaped trajectory. A

particle's propagation under a magnetic field can be visualized as in Fig. 2.6.

$$\frac{d^2\mathbf{r}}{ds^2} = \frac{q}{p}\frac{d\mathbf{r}}{ds}\mathbf{B}(\mathbf{r}) \tag{2.1}$$

When charged particles pass through a silicon tracker detector, they create signals that are called *hits*. The reconstructed trajectories of particles are referred to as *tracks*. Fig. 2.6 (c) shows examples of hits and their associated tracks with different colors for each particle. The particle track reconstruction is the task of determining the trajectories of particles that propagate outwards from the collision center, using the measurements (hits) of the silicon tracker detectors.



(a) (b) (c)

Figure 2.6: Example of particle hits and tracks in a magnetic field. Particles travelling in a magnetic field away from the primary vertex (a), leaves hits in the detectors as shown in black points (b). These hits are reconstructed with track finding algorithms to particle tracks (c). Figure is adapted from [11].

2.3 Approaches to Particle Track Reconstruction Problem

Particle track reconstruction is a very challenging and computationally expensive problem. It is the most computationally expensive task of the CMS event reconstruction and it is reported to take the same amount of time with the rest of the computations [21]. The LHC experiments currently employ Kalman filter and Hough transform based approaches. Despite the Kalman filter based approaches' high accuracy tracking performance, they scale quadratically or worse in general (e.g. $\mathcal{O}(n^6)$ [2]). This lead

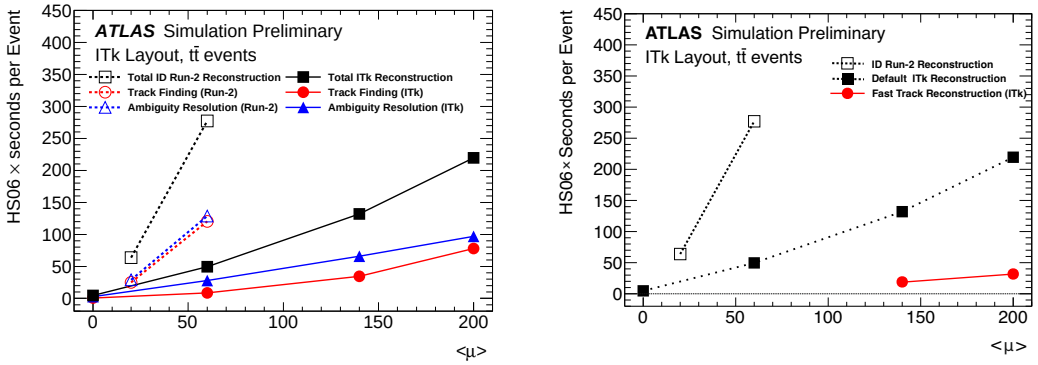
researchers to start investigating new methods for particle track reconstruction as a result of this situation in recent years. Global methods based on pattern recognition are started to be considered in contrast to local methods (i.e. Combinatorial Kalman Filter (CKF) based methods [20, 21]), which were used at Run-1 and Run-2, where each track is reconstructed one by one. On top of these methods, developments in Deep Learning (DL) and Quantum Computer (QC) technologies opened the door to new possibilities in particle track reconstruction. In the next sections, a more detailed overview of these approaches is given.

2.3.1 Traditional Approach

The particle track reconstruction algorithms at the LHC generally share a similar work-flow [20]. Track reconstruction starts with merging the charge deposition measurements on pixel detectors to single hits. Then, three hits from different layers of detectors are selected to form triplets that are called seeds. This stage is called track seeding. At least 3 hits are required to estimate a track's curvature, which will later be used to calculate its parameters such as the charge, momentum and point of closest approach (perigee) to the interaction region. These triplets are chosen to be good seeds according to requirements set by geometry, momentum and perigee. Good triplet seeds are then fed to a CKF algorithm [22], which uses each seed to estimate hits in the next layer that might belong to the same particle and builds track candidates after going through all detector layers. At the last stage, a set of quality criteria are applied to track candidates to get rid of ambiguities and track candidates with shared hits [23].

Track seeding and building stages are the most computational expensive stages of particle track reconstruction at the LHC. Both scale worse than quadratic in number of hits [24]. This behaviour can be seen in Fig. 2.7, where previous Run-2 results are plotted in empty black boxes. Fig. 2.7a compares the total inner detector reconstruction time of the Run-2 with the simulated results of the new inner tracker hardware [25], at respective pile-up values of $\langle\mu\rangle$ = 20 and 60 for Run-2 and values up to 200 (to cover HL-LHC conditions) for the new hardware. Fig. 2.7a further compares the default track reconstruction software with the proposed Fast Track Reconstruction algorithm [26]. The times are given in HS06 benchmark [27], which is a benchmark

used to quantify performance of CPUs in the HEP community. Both figures show that significant improvements are expected with the deployment of new hardware and software.



(a) (b)

Figure 2.7: Computation time required for the reconstruction of particle tracks at the ATLAS experiment with respect to pile-up. Times are given in HS06 benchmark [27]. Run-2 results with $\langle\mu\rangle$ = 20 and 60 is plotted with dashed lines and empty markers. The expected reconstruction performance of the upgraded ATLAS inner tracker (ITk) is plotted with straight lines and filled markers. Computation time of different track reconstruction stages are given with different colors (a). The expected performance of the Fast Track Reconstruction algorithm is plotted with red circles (b). Figures are retrieved from [26].

2.3.2 Deep Learning Based Approaches

2.3.2.1 Neural Networks

Artificial Neural Networks (ANN or NN in short) are machine learning models that are inspired from the working principles of neurons. The first conceptual use of NNs date back to 1940s. McCulloch and Pitts proposed a mathematical model to explain the nervous activity [28] and A.M. Turing considered the human cortex as an unorganized machine and theorized on educating this machine [29]. Later, first practical algorithm was implemented by Frank Rosenblatt, named the perceptron [30].

A perceptron is a supervised binary classification algorithm. Perceptrons have parameters that require sensitive tuning in order to give the desired output. This is usually achieved through an iterative process called training (or optimization). During training, a set of pre-selected data is fed to the algorithm and the parameters of the algorithm is tuned until the model gives the desired output. Then, the model is also tested on unseen data. Details of the training process will be covered throughout the thesis.

The parameters of a perceptron are called weight and bias. Weight is a matrix such that $W \in \mathbb{R}^n$ and bias is $b \in \mathbb{R}$ for an input $\hat{x} \in \mathbb{R}^n$. The perceptron multiplies the input with the weight matrix and adds the bias. Finally, this value is fed to an activation function g, and the output $y \in \mathbb{R}$ is obtained as given in Eq. 2.2.

$$y = g\left(b + \sum_i W_i x_i\right) \tag{2.2}$$

The perceptron can also be expressed graphically as in Fig. 2.8.

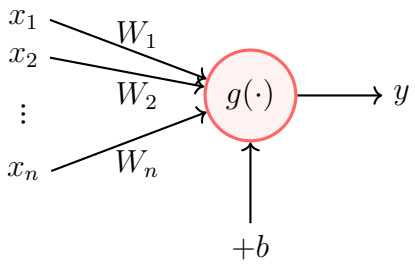


Figure 2.8: Graphical representation of a perceptron. $\hat{x}$ is the input, y is the output, g is the activation function, W is the weight matrix and b is the bias.

The activation function (g) is used to introduce non-linearity to the system. The system is linear until the activation function is applied. This allows the perceptron to learn non-linear distributions. There are two perceptron conventions. First is a continuous model, where the output of the activation function is a continuous value. Sigmoid and Tanh functions are generally used to limit the output value of the perceptron, while ReLU behaves linearly above a certain threshold. The second is a discrete model, where a threshold function, such as a step function, is applied after the activation function to produce the outputs as a 0 or a 1. A mathematical model of a perceptron can be seen in Eq. 2.2. Above mentioned activation functions are summarised in

Table 2.1 and their graphical representations are given in Fig. 2.9.

Table 2.1: Some of the popular activation functions used in NNs.

Sigmoid	Tanh	Step	ReLU
$\frac{1}{1+e^{-x}}$	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$= \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$	$= \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$

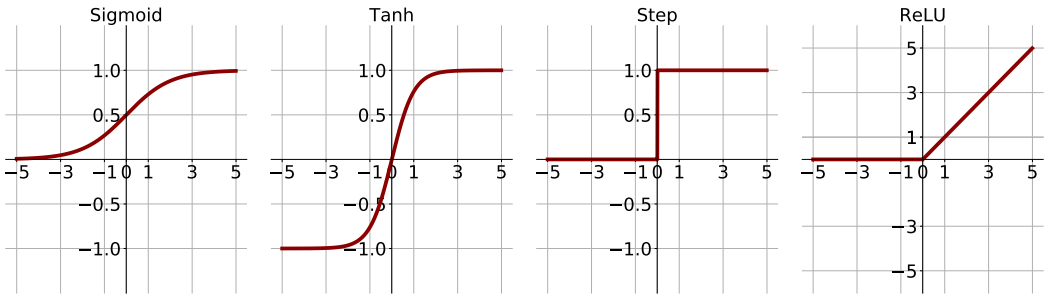


Figure 2.9: Some of the popular activation functions used in NNs. Their mathematical descriptions are given in Table 2.1.

In the early years of NNs, researchers implemented multi-perceptron learning schemes and tried to train them to learn mathematical functions. Their attempts mostly resulted in failure due to the lack of learning capacity and not using activation functions in their architectures, combined with inefficient algorithms that required excessive computational resources. This situation lasted until the 1980s. Since then, backpropagation is used to train multi-layer perceptrons. Backpropagation is a gradient descent based algorithm to train NNs. Although the name backpropagation was first used by Rumelhart et al. in 1986 [31], similar algorithms had been already proposed by many researchers in different contexts since the 1960s [32, 33].

The invention of backprogation has led to many breakthroughs. Researchers started to train multi-layer perceptron models with larger learning capacities. Due to the multi-layered structure and their relevance to neurons, perceptron model-based machine learning became known as Deep Learning (DL), and these models became popular to a wider audience as Deep Neural Networks (DNNs). DNNs can be represented visually. Vertically stacked perceptrons form a layer and different layers are stacked horizontally.

Layers can have different number of perceptrons depending on the design of the model. A generic DNN architecture is displayed in Fig. 2.10. One of the most significant developments of the early days of DL was the application of DNNs to recognize hand-written images by LeCun et al. in 1989 [34]. Many more models rapidly emerged later and showed promising results for many pattern recognition tasks such as classifying steady vowels from speech spectra of a person [35]. However, the second development phase also suffered from the lack of large amounts of computational power required to train large models.

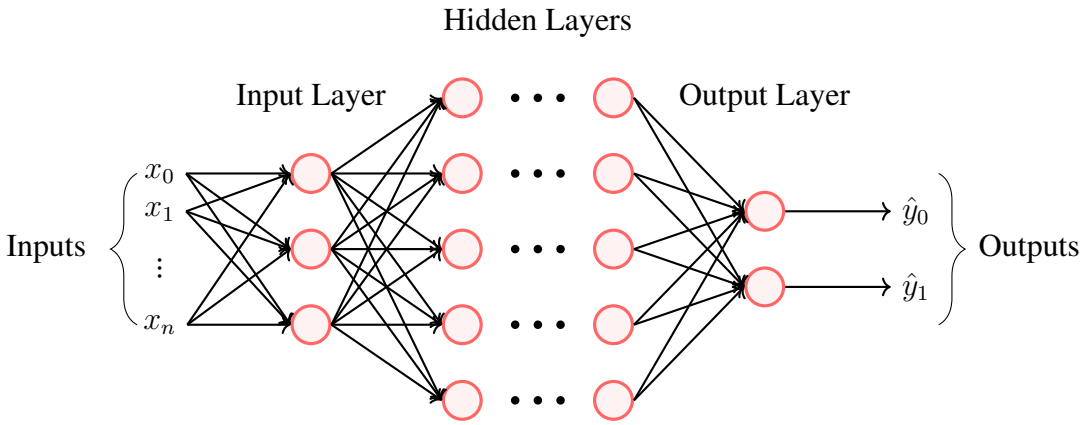


Figure 2.10: Graphical representation of a generic Deep Neural Network model. Input ($\hat{x}$) is fed to an input layer to increase the dimension size. Then, its output is passed through a set of hidden layers. Finally, an output layer reduces the dimension to the desired value to obtain predictions ($\hat{y}$).

Advances in Central Processing Unit (CPU) technology did not yet provide enough computational resources to meet the extensive requirements of training large DNN models in the 1990s. A solution for this challenge was presented from a different processing unit, called Graphical Processing Unit (GPU) in 2009. GPUs contain thousands of cores that enable tremendous amounts of parallelization in computational tasks based on matrix multiplication. DNN models, which need a lot of large matrix multiplications benefited from the power of parallelization provided by the GPUs [36]. The speed-up gained by using GPUs to train DNNs enabled the training of large scale models and started the biggest era of DL [37, 38]. It is important to note that GPUs are used more broadly in computing not just for ML applications.

The HEP.TrkX project showed an initial Recurrent Neural Network (RNN) and Graph

Neural Network (GNN) approach to the particle track reconstruction problem using a limited dataset [5]. They extended their results in the Exa.TrkX project showing promising accuracy in tracking [6] with the help of an end-to-end architecture and using a larger dataset. Recently, a GNN based particle flow reconstruction model, Machine Learned Particle Flow (MLPF) [39], successfully showed a linear scaling in the pile-up (as opposed to quadratic scaling of the traditional methods), which was considered as a breakthrough in the HEP community. This increased expectations in the potential of GNNs in particle tracking in dense environments.

2.3.3 Recurrent Neural Networks

In the RNN model, the output of a NN layer is connected to other NN layers in a recurrent structure. The iterative number of layers caused the gradients of the NN layers to decrease and resulting in a decrease in trainability of these types of NNs [40]. This problem is named the vanishing gradients problem due to the decrease of gradients [41]. Their performance suffered from the vanishing gradients problem in the early days. The introduction of the LSTM [42] blocks that are used besides the RNN layers [43] solved this problem by limiting the effect of previous data by the help of a structure called the "forget gate". The HEP.TrkX project proposed using LSTMs similarly to Kalman Filters to overcome this problem [23]. In this approach, each layer's measurements are fed to LSTM blocks which combine this information with the output of the previous blocks in the RNN structure. Outputs are then fed to Fully Connected (FC) NNs to give the final prediction. The pipeline of the model can be seen in Fig. 2.11.

2.3.4 Graph Neural Networks

GNNs are NN models that use the data in the form of graphs. A graph can be thought as an extended form of images, where each data point has a connection to some other data points with a certain weight. A graph has 3 attributes; **u** represents global (environmental) features that might be shared across different graphs. $V = \{\mathbf{v}_i\}_{i=1:N^v}$ represents features that belongs to nodes. Last but not least, $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$

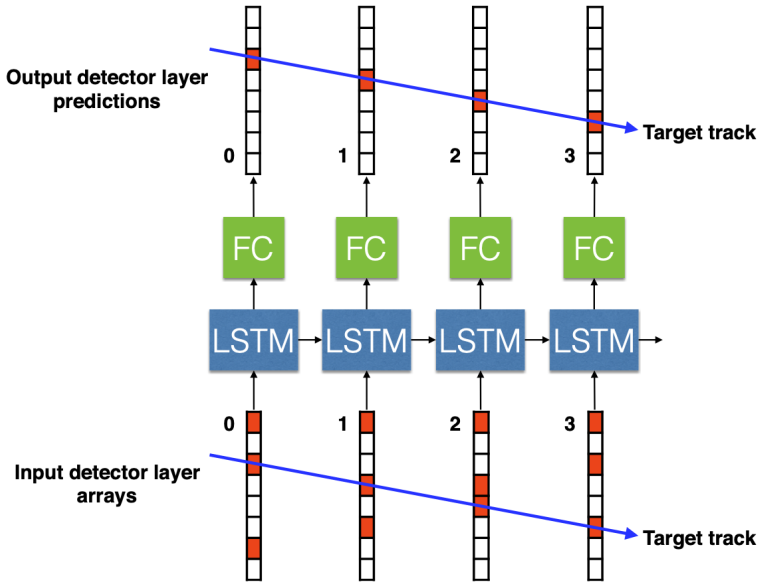


Figure 2.11: RNN approach of the Hep.TrkX project. Each detector layer measurements are fed to an LSTM block. Recurrent LSTM blocks propagate information through different layers. Outputs of each LSTM block is then fed to an FC block to obtain target tracks. Figure is retrieved from [5].

represents the features of the edges via $\mathbf{e}_k$, while r_k and s_k are indices of the receiving and sender nodes respectively. They form the graph $G = (\mathbf{u}, V, E)$ following the formalism of Battaglia et al. [44]. These attributes can be seen on an example graph in Fig. 2.12.

The approach of the HEP.TrkX project was to encode the hits as the nodes and the potential tracks as the edges of a graph [5]. They built a model with 3 major blocks. The InputNet increases the dimensions of the node attributes by using a multi layer perceptron. The EdgeNet takes all the edges one by one and predicts whether each edge is a true or fake connection. The NodeNet takes each node and its neighbours as inputs weighted by the outputs of the EdgeNet. The recursive iterations of EdgeNet and NodeNet allow the network to aggregate the information throughout the graph. The pipeline of the model can be seen in Fig. 2.13. The HEP.TrkX GNN model is chosen to be the base of the Hybrid GNN model, presented in this thesis, due to its success. The working principles, of which, will be detailed in Chapter 4.

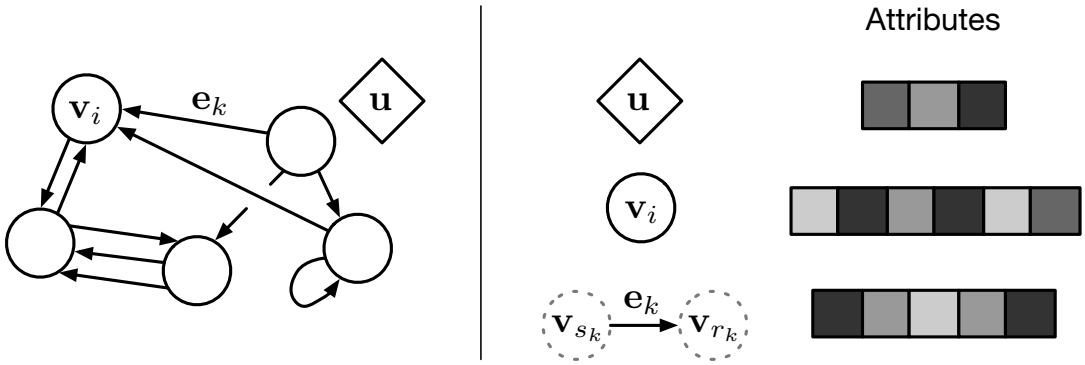


Figure 2.12: Attributes of a graph. Circles represents nodes and $\mathbf{v}_i$ is the feature matrix of i^{th} node. Arrows represent the edges and $\mathbf{e}_k$ is the feature matrix of the k^{th} edge. Square represents the global features and $\mathbf{u}$ is the global feature matrix. Figure is retrieved from [44].

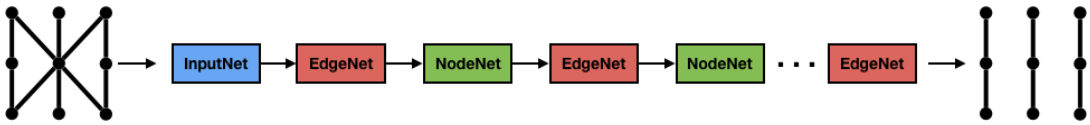


Figure 2.13: The GNN model of the HEP.TrkX project [5]. The model takes a graph as its input and feeds it to InputNet to increase the dimensions of node features. Then, recurrent EdgeNet and NodeNet models update the graph features. An EdgeNet is used at the final layer to extract edge features of the graph to decide whether if the edges of the graph are true or fake connection.

2.3.5 Quantum Computing Based Approaches

Quantum Computing is a relatively new type of computing, which uses qubits rather than bits. Qubits have properties such as superposition and entanglement that have no equivalent in classical bits. On top of that, the Hilbert Space ("state space" of qubits) grows with 2^n, while the "state space" of classical bits only grows linearly in classical computing. These features open new possibilities in computation, both in algorithm development and in physical realization of Quantum Computers which are very active fields of research. An extensive overview of Quantum Computing and methods relevant to this thesis will be presented in Chapter 3.

Researchers at CERN became attracted to QC as well as ML due its potential in

reducing the scaling of certain algorithms [45]. Multiple attempts to use Quantum Computing in particle track reconstruction were made using quantum adiabatic computing. Bapst and Zlokapa et al. use a pattern recognition approach [8, 9], while Quiroz et al. combines quantum adiabatic computing with Quantum Associative Memory [10]. Recently Magano et al. used Quantum search routines to reduce scaling of parts of the traditional tracking methods [2].

2.3.5.1 Quantum Adiabatic Computing

Quantum adiabatic computing, also known as Quantum Annealing (QA) is one of two major paradigms in Quantum Computing with the other being circuit-based Quantum Computing. It uses adiabatic evolution of quantum states to approach the ground state of a certain Hamiltonian [46]. The adiabatic theorem states that if a Hamiltonian is in the ground state and if the Hamiltonian is slowly varied to represent another system, the system will stay in the ground state. This allows some problems to be approximately solved [45], such as the Quadratic Unconstrained Binary Optimization (QUBO) problem given in Eq. 2.3. This defines the target Hamiltonian for n qubits, where the J and h matrices sets the two body coupling strength of the spin states in z-direction, as the σ_z is the Pauli-Z matrix. After preparing the target Hamiltonian, the system is evolved during a predetermined time. Then, the spin states are measured to get binary $\{-1, 1\}$ (or $\{0, 1\}$ depending on the convention) output for each of the qubits and the result is expected to be an approximate solution that minimizes the H_{QUBO} given in Eq. 2.3.

$$H_{QUBO} = \sum_{i,j=1}^{n} J_{ij}\sigma_i^z\sigma_j^z + \sum_{i=1}^{n} h_i\sigma_i^z \tag{2.3}$$

The QA approaches to particle track reconstruction uses the Denby-Peterson formulation [47, 48], and in fact improves it, to map the problem into a QUBO [9]. The Hamiltonian H_{DP}, where a, b, c are hits (triplets), r_{ab} is the 3D distance vector, θ_{abc} is the 3D angle that is formed by three hits is given in Eq. 2.4. If hits i and j belong to same particle, s_{ij} is 1 and 0, otherwise [9]. A graphical representation of these variables are presented in Fig. 2.14.

$$H_{DP} = -\frac{1}{2}\Bigg[\sum_{a,b,c}\left(\frac{\cos^{\lambda}\theta_{abc}}{r_{ab}+r_{bc}}s_{ab}s_{bc}\right) - \alpha\left(\sum_{b\neq c}s_{ab}s_{ac} + \sum_{a\neq c}s_{ab}s_{cb}\right) - \beta\left(\sum_{a,b}s_{ab} - N\right)^{2}\Bigg] \tag{2.4}$$

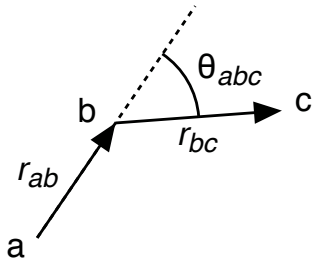


Figure 2.14: Coordinate definitions of a triplet in Denby-Peterson representation. a, b and c represent spatial coordinates of three hits. r_{ab} and r_{bc} are the 3D distance vectors between hits. θ_{abc} is the 3D angle between the two distance vectors. Figure is retrieved from [9]

Although the QA methods show great performance, they also suffer from a scaling problem, which is intrinsic to all QA methods. Zlokapa et al. showed that the annealing time needed to get a result increases exponentially with the number of track segments [9]. Furthermore, there are hardware limitations such as number of qubits, accurate measurements of qubits and decoherence, which is a finite duration a qubit can hold information. These factors limits the QA methods' to scalability to larger data-sets.

In this work, a new look at the problem is presented by combining GNNs and circuit-based Quantum Computing (the second paradigm in Quantum Computing) to investigate the potential of particle tracking.

2.4 The TrackML Challenge and the Dataset

As a part of the preparations for HL-LHC conditions, researchers created the TrackML Challenge to invite experts in Machine Learning to contribute to improving performance of particle track reconstruction in 2018 [11]. They created a simulated dataset

to resemble the HL-LHC conditions using an open-source A Common Tracking Software (ACTS) [49]. A generic detector architecture that is common to most tracking detectors is used to simulate the events. A drawing of the detector layout can be seen in Fig. 2.15. After the conclusion of the challenge, the dataset is still being used by many researchers in the field, to benchmark their results. This work follows the same trend and uses the TrackML dataset, which is hosted on the Kaggle website [50].

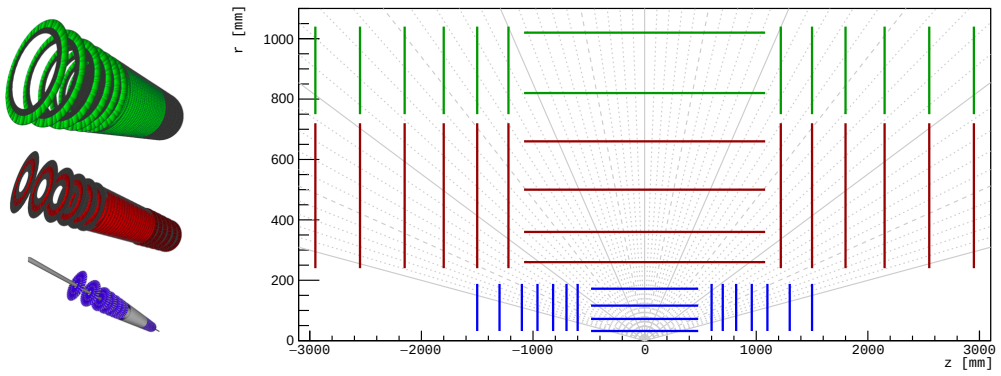


Figure 2.15: Layer geometry of the TrackML detector. Tracking layers with barrel and end-cap geometries are shown on the left. The r-z plane projection of the overall geometry is shown on the right with respective colors according to their geometries. Figure is adapted from [11].

The TrackML detector consists of 3 types of silicon detector layers and has barrel and end-cap regions. The barrel region is located at the center as shown in Fig. 2.15. The end-cap region has the detectors attached on disk at the outer regions in the same figure. Both the barrel and end-cap regions consists of 3 types of layers. The inner layers are pixel detectors, plotted in blue. The outer layers (red and green) are strip detectors, where the outmost (green) layer have less resolution. The 3D geometry of these 3 types of layers are also shown in Fig. 2.15.

The dataset contains 10000 bunch-crossing events that are results of hard QCD interactions that generate a $t\bar{t}$-pair with an additional 200 soft QCD interactions to resemble high pile-up conditions of HL-LHC. There are 4 files for each event and their physics contents are listed in Table 2.2. The `hits` file contains information regarding the spatial coordinates of particle hits as well as unique identifiers for both the hit and the module that registered it. The `cells` file contain more details about hits, such as the

cells of the module that registered the hit along with a value for the electrical signal that is deposited by the particle. The `particles` file contains information about the particles' physical properties such as position, velocity, momentum and number of hits it created throughout the detectors. Finally, the `truth` file provides the information regarding the particles true trajectory, momentum and a weight value that is used in the TrackML challenge to benchmark the results.

Table 2.2: Physics contents of different types of TrackML files [11].

File Name	Description
Hits	• a unique identifier for each hit • (x,y,z) location of each hit • a unique identifier for each module that registered a hit
Cells	• a unique identifier for each hit • a unique identifier for each cell of the channels of the module that registered a hit • value of the electrical signal that is deposited by the particle
Particles	• a unique identifier for each particle • initial velocity vector each particle • initial momentum vector each particle • number of hits left by each particle
Truth	• a unique identifier for each hit and the unique particle identifier that belongs to that hit • (x,y,z) true location of the intersection point with the hit and the particle trajectory • true particle momentum at the intersection point • weights for each hit that is used to calculate TrackML score

CHAPTER 3

QUANTUM COMPUTING AND MACHINE LEARNING

Humankind has been in a search to increase the computational power since the invention of abacus. The search gained a momentum after the invention of digital computers, which is also known as the beginning of the third industrial revolution. All fields from fundamental sciences to industrial applications required more processing power to advance their technology and improve our understanding of the nature. However, certain problems turned out to require exponentially more processing power with an increasing system size, making some problems almost impossible to solve with finite resources. Schrödinger's equation, which expresses the behaviour of Quantum Mechanics, is one of these problems. In 1982, Richard P. Feynman authored a famous article, where he states we need computers that are based on Quantum Mechanics to simulate Quantum systems [51]. This and many more valuable ideas led to the development of Quantum Computers that we can access even through a web browser today [52, 53].

A Quantum Computer (QC) employs qubits rather than the classical bits and need specific manipulation and measurement techniques that depend on the physical nature of the qubits. Unlike classical bits, qubits can have superposition states where they store both 0 and 1 with a certain weight, which can only be accessed when the qubits are measured. Qubits can also be entangled with each other meaning that they can interact with each other without external intervention. These two facets bring a new understanding to our classical computational perspective and open many doors to potential applications that we might not be able to imagine as of today.

The most famous application is the Shor's algorithm in the field of cryptography. Shor's algorithm uses QCs to solve the integer factorization problem in polynomial

time, which would threaten the cryptosystems that depend on prime number factorization [54]. RSA is a public-key cryptosystem, based on the computational hardness of prime number factorization and it is widely used to secure most of the internet connections from bank transactions to private messages [55]. Another promising application is the Grover's algorithm, which is a search algorithm, that can find a marked item in a set of N elements in $\mathcal{O}(\sqrt{N})$ queries, while the best classical algorithm needs at least $\mathcal{O}(N)$ queries [56]. There are also applications, such as the Harrow-Hassidim-Lloyd (HHL) algorithm, that can speed-up solving systems of linear equations [57].

Applications of QCs also cover simulating Quantum systems. The Variational Quantum Eigensolver (VQE), which estimates the ground state of a given Hamiltonian is an important step toward realizing Feynman's dream [58]. Researchers showed in many instances, that one can estimate the ground state energy of various molecules at different bond lengths [59]. Early success of variational methods in Quantum Chemistry led researchers to use them to solve machine learning problems in a similar fashion to DL [60–62].

This chapter gives an overview of the circuit-based QC paradigm, followed by a look at the current and future of the QC hardware. Then, the Variational Quantum Classification method - the backbone of this work - is studied in depth.

3.1 Circuit-based Quantum Computing

A Quantum circuit is a representation of sequential operations on qubits. Each qubit is placed on a line and operations on them are represented with certain shapes in this representation. This provides an easy visualization for the operations on qubits and simplifies the reconstruction of the braket notation from the circuit.

In the braket notation of quantum mechanics, states (Ψ) are represented with the bra ($\langle|$) and the ket ($|\rangle$) signs, which are the Hermitian conjugate of one another.

$$|\psi\rangle = \langle\psi|^{\dagger} \tag{3.1}$$

The density matrix of a quantum state represents the system with the given $|\psi\rangle$, such

that;

$$\rho = |\psi\rangle \langle\psi| \tag{3.2}$$

There are two types of quantum states depending on the restriction on the density matrix. Pure states are states with a unit length and their density matrix is defined as;

$$Tr[\rho^2] = 1 \tag{3.3}$$

Mixed states on are combinations of pure states and follow a different constraint;

$$Tr[\rho^2] < 1 \tag{3.4}$$

Two-level (e.g. a spin 1/2 system) pure and mixed states can be visualized with the help of the Bloch sphere. Pure states are located on the surface, while mixed states are located inside the sphere. Representations of these states on the Bloch sphere can be seen in Fig. 3.1, where θ is the angle to the z-axis, and the φ is the angle to the x-axis on the x-y plane.

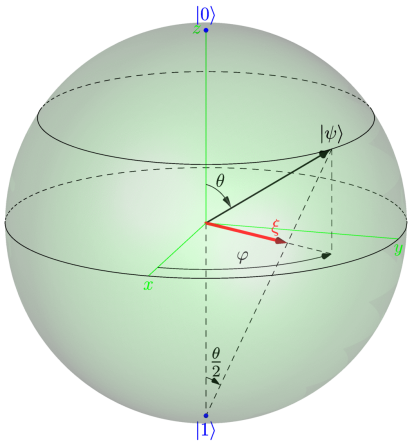


Figure 3.1: Bloch sphere representation of quantum states. A pure ($|\psi\rangle$) is shown with a black arrow. A mixed (ξ) state is shown with a red arrow. Figure is retrieved from [63].

In this work, we only consider two-level pure states. Therefore, the constraint set by Eq. 3.3 will be valid throughout. Such a generic state can be represented as;

$$|\psi\rangle = \cos\frac{\theta}{2}\,|z;\uparrow\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|z;\downarrow\rangle \tag{3.5}$$

It is also possible to represent these states with matrices. In this notation, a qubit's state can be represented with a 1x2 matrix. The convention is to define the spin-up (in the positive z-direction) state as the $|0\rangle$ or $|\uparrow\rangle$ and the spin-down (in the negative z-direction) state as the $|1\rangle$ or $|\downarrow\rangle$. This convention allows representing the states similar to $\{0,\ 1\}$ binary representation of the classical computing.

$$|0\rangle = |z;\uparrow\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad\qquad |1\rangle = |z;\downarrow\rangle = \begin{bmatrix}0\\1\end{bmatrix} \tag{3.6}$$

Quantum states of many qubits can be combined by using the tensor product, to be viewed as a single object. The overall state becomes a matrix with the size $1\text{x}2^{N_{qubits}}$.

$$|\Psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_N\rangle \tag{3.7}$$

Reading out a value from a qubit is different compared to the classical case. This procedure is referred to as a measurement in QC. A measurement of a state, defined as $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$, where a and b are complex scalars, can result in two of the outcomes with a certain probability defined by the modulus square of each coefficients, with the condition of their sum being 1. This means that, different outcomes might observed each time the state is measured.

$$P(0) = \|a\|^2 \tag{3.8}$$

$$P(1) = \|b\|^2 \tag{3.9}$$

$$P(0) + P(1) = 1 \tag{3.10}$$

The ambiguity in measurement can be reduced by taking multiple measurements. Expectation value is the weighted average of the probabilities, where the weights are

eigenvalues of the observable. If the measurements along the z-direction, the Pauli-Z matrix is considered

$$\sigma_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.11}$$

and the expectation value of the Pauli-Z observable is given as;

$$E = \langle\psi|\sigma_Z|\psi\rangle = (+1)\|a\|^2 + (-1)\|b\|^2 = \|a\|^2 - \|b\|^2 \tag{3.12}$$

Manipulation of quantum states are done by using unitary operations. A unitary operation is a norm preserving square matrix such that, its Hermitian conjugate is its inverse and has the dimensions of $2^n \times 2^n$ for an n qubit state. Unitary operations are called quantum gates in QC. The number of elementary gates applied to a qubit is called the circuit depth. A generic unitary single qubit gate can be defined as

$$U(\theta, \phi, \lambda) = \begin{bmatrix} cos(\theta/2) & -e^{i\lambda}sin(\theta/2) \\ e^{i\phi}sin(\theta/2) & e^{i(\lambda+\phi)}cos(\theta/2) \end{bmatrix} \tag{3.13}$$

All of these operations can be expressed with the circuit representation with ease. As a convention, all states are initially set to $|0\rangle$. Then unitary operations (e.g. `U`) on qubits are applied. The final measurements can be represented graphically with a meter sign, as seen in Fig. 3.2. This example can be extended to any number of qubits and gates with ease, as shown in Fig. 3.3.

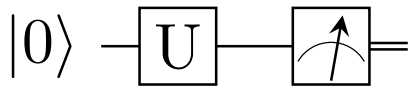


Figure 3.2: A single qubit circuit representation. The qubit starts with the $|0\rangle$ initial state. A unitary operation `U` is applied as shown with the box. Then, the measurement operation is performed as shown with a meter sign in a box.

In QC, certain single and multiple gates are widely used, that have special names due to their significance. For example, the Pauli-X gate is usually referred to as the `NOT`

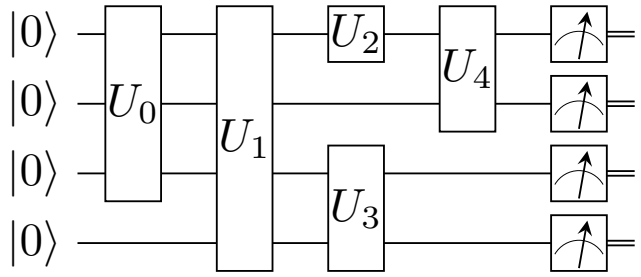


Figure 3.3: An arbitrary Quantum circuit with various qubits and multi-qubit gates. Unitary operations can be applied to multiple qubits as shown in the figure.

gate (or `X` gate), as it flips the state in the z-basis, as seen in Eq. 3.14. Hadamard (`H`) gates, named after Jacques Hadamard, are generally used to create superposition of states. Matrix representation of the Hadamard gate and how it acts on states can be seen in Eq. 3.15. A list of all relevant gates in this work are presented in Appendix A.

$$\sigma_X = \mathrm{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \begin{aligned} \mathrm{X}\,|0\rangle &= |1\rangle \\ \mathrm{X}\,|1\rangle &= |0\rangle \end{aligned} \tag{3.14}$$

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \begin{aligned} \mathrm{H}\,|0\rangle &= \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \\ \mathrm{H}\,|1\rangle &= \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle \end{aligned} \tag{3.15}$$

Single qubit gates can be combined together with the tensor product to form multi-qubit gates. However, the reverse is not generally true. There are multi-qubit gates that cannot be separated to single qubit representations. The most important gates are controlled operations that have the possibility to entangle qubits. Controlled NOT (`CNOT` or `CX`) gate can be used to create entanglement between qubits that are in superposition. The `CNOT` gate acts as a `X` gate on the second qubit, if the first qubit is in the $|1\rangle$ state. The matrix form of CNOT and how it acts on different states can be

seen in Eq. 3.16.

$$\mathrm{CX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{aligned} \mathrm{CX}\,|00\rangle &= |00\rangle \\ \mathrm{CX}\,|01\rangle &= |01\rangle \\ \mathrm{CX}\,|10\rangle &= |11\rangle \\ \mathrm{CX}\,|11\rangle &= |10\rangle \end{aligned} \tag{3.16}$$

We can further combine the H gate with the `CX` gate to obtain the entangled states such as,

$$|\Psi\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle \tag{3.17}$$

and also graphically as shown in Fig. 3.4,

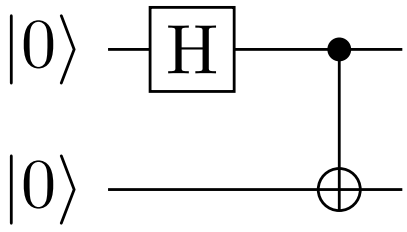


Figure 3.4: Bell state preparation circuit.

The state given in Eq. 3.17 is generally referred to as on of the *Bell state* and has a fundamental role in Quantum Information Theory. The given Bell state defines a state such that, if one measures any of the qubits, the state of the other qubit would be determined without a measurement. This simple circuit can be tested by using a publicly available Quantum Computer. For this purpose, the Bell state preparation circuit given in Fig. 3.4 is prepared and executed 1000 times on the `ibmqx2` [52] device and the results with statistical error bars are compared with simulation results obtained by using `imbq_qasm_simulator` [64] and plotted on Fig. 3.5.

There are two important points that need attention. The first point is the difference between the numbers of $|00\rangle$ and $|11\rangle$ observations. These two states should have equal probability according to Equations 3.9, 3.10 and 3.17. The main reason for

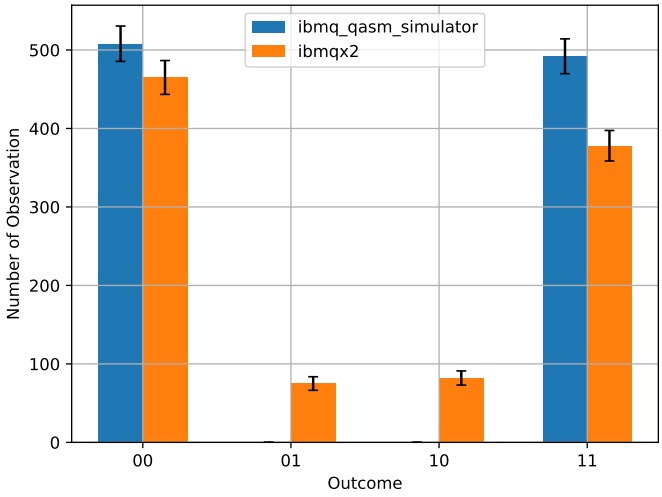


Figure 3.5: Histogram of Bell state measurements. 1000 Bell state are prepared and then measurement outcomes are plotted. ibmqx2 results are plotted in orange and ibmq_qasm_simulator results are plotted in blue.

the observed difference is finite sampling of the states and referred to as shot noise. Although the outcomes are not ideal (500/500), the ideal outcome is still in the statistical error limits. The effects of shot noise will be discussed in detail in the following sections. The second discrepancy is the unexpected outcome of the two states $|01\rangle$ and $|10\rangle$. The reason for observing these states is due to noisy hardware of Quantum Computers. There are various sources that introduce noise on the Quantum systems. Dealing correctly with noise is one of the greatest challenges in front of future Quantum Computers. In the next section, an overview of the current state of the QC hardware is presented to help understanding the current status.

3.2 State of Quantum Computing Hardware

It is important to understand the technical limitations of Quantum Computers before proceeding to the details of algorithms, which will be shaped by them. Current Quantum Computers are noisy, meaning they can produce false results, have low connectivity between qubits and do not allow implementation of complicated algorithms. The name "Noisy Intermediate Scale Quantum (NISQ) era" [65] emphasises

this situation.

There are many ways to build a Quantum Computer, which makes comparisons across different devices very difficult. Therefore it is important to understand what different hardware paradigms offer. To do this, we present a brief overview of the some publicly available QC hardware in Section 3.2.1, then we compare their properties according to DiVincenzo's criteria [66] in Section 3.2.2. Finally, we present a benchmark that aims to quantify the quality of different hardware in Section 3.2.3.

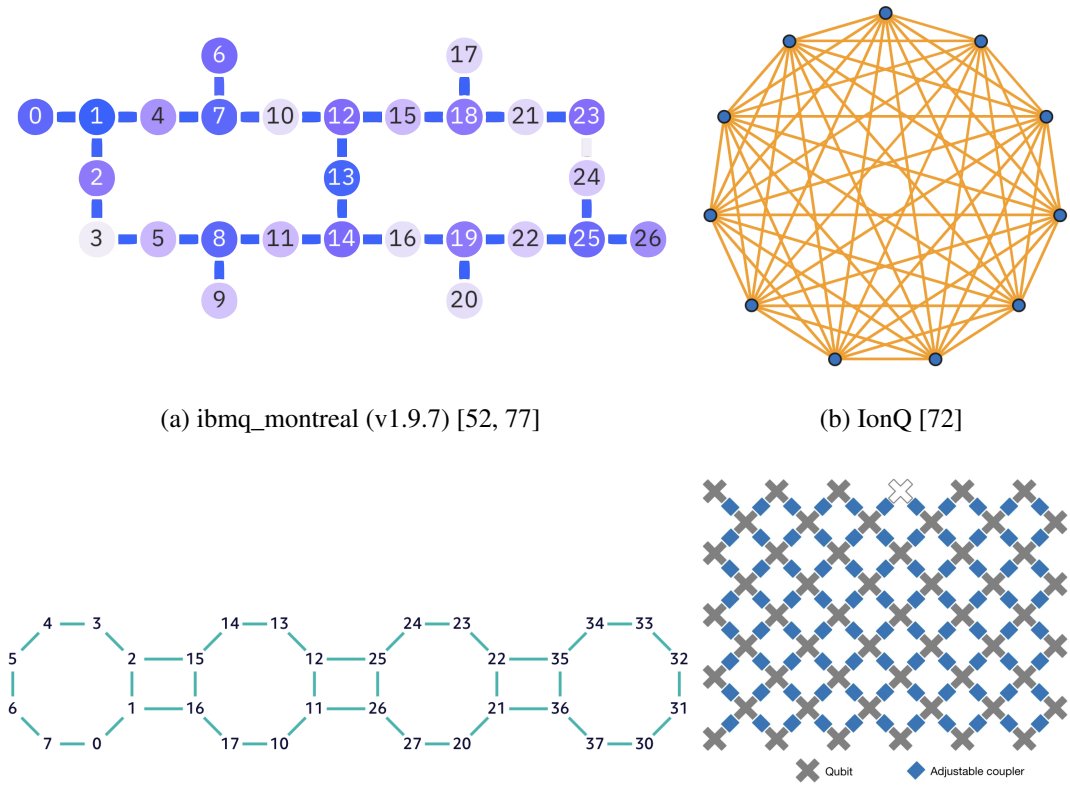
3.2.1 Quantum Computing Hardware Paradigms

Many hardware and software technologies have been developed to build Quantum Computers and all of them have their advantages and disadvantages. One of the earlier devices are Quantum Annealers, whose working principles were discussed in Section 2.3.5.1. The DWave QC company [67] uses superconducting qubit technology to exploit a natural evolution of quantum states. Thanks to chip manufacturing advancements in semiconductor industry, they can build systems with as large as 5000 qubits [68]. Their devices showed great success in several optimization tasks, from particle tracking [9] to traffic flow optimization [69]. However, this number shouldn't be misleading as their qubits can only do annealing, meaning that they are mostly used to estimate the result of a problem, instead of finding the exact solution (they can also be used to find a minimum in some cases). On the other hand, other companies work on Quantum Computers which hopes to implement any given unitary operation. Companies such as IBM [52], Google [70] and Rigetti [71] also use the superconducting qubit technology to build their gate based Quantum Computers. This technology has seen a great interest from companies due to its easy scalability and the availability of the semiconductor technology to build such devices. One of the major setbacks they are facing is the limited connectivity and short decoherence time of qubits.

Another popular approach, adopted by IonQ [72] and Honeywell [73], is to use trapped ions in a crystal. This technology solves some of the problems of superconducting qubits with very large decoherence times and full connectivity. However, they suffer from scalability due to few available qubits in a single crystal. Recently, Honeywell

proposed using movable electric charges to solve the scalability problem [73].

Using photons as qubits has also gained popularity in the last years. Zhong et al. recently showed a Quantum computational advantage using boson sampling [74] and the company Xanadu [75] introduced their photonic chip-based devices [76]. There are many other approaches to build Quantum Computers, but listing all of them would be beyond scope of this thesis. Connectivity architecture (topologies) of some of the existing Quantum Computers with different technologies can be seen in Figure 3.6. As mentioned earlier, superconducting devices have less connectivity between qubits as shown in Figures 3.6a, 3.6c and 3.6d. The figure shows that different companies are testing different topologies to increase the connectivity of their qubits. On the other hand, the ion trap approach provides full connectivity as shown in Fig. 3.6b.



(a) ibmq_montreal (v1.9.7) [52, 77]

(b) IonQ [72]

(c) Rigetti Aspen-9 [71]

(d) Google Sycamore [70]

Figure 3.6: Topologies of Quantum Computers with different technologies. Superconducting qubit devices have low connectivity (a,c,d), while ion trap devices have full connectivity (b).

3.2.2 DiVincenzo's Criteria

In the beginning of the 2000s, David P. DiVincenzo has put together $5 + 2$ (optional) requirements that are needed to build a quantum computer system [66]. These can be listed as follows,

- A scalable physical system with well characterized qubits
- The ability to initialize the state of the qubits to a simple fiducial state, such as $|000\rangle$
- Long relevant decoherence times, much longer than the gate operation time
- A "universal" set of quantum gates
- A qubit-specific measurement capability
- The ability to interconvert stationary and flying qubits (optional)
- The ability to faithfully transmit flying qubits between specified locations (optional)

The first requirement is to have "a scalable physical system with well characterized qubits" [66]. A well characterized qubit means that the physical parameters of the qubit is known such that the internal Hamiltonian can be fully defined. Achieving the scalibility of the qubits might require different methods depending on the hardware paradigm. As it was discussed in the above paragraphs, different technologies provide different means to achieving scalability. Superconducting qubits solve this by limiting connectivity and trapped ion qubits are proposed to solve this with the help of moving electrical charges [73].

The second one is "the ability to initialize the state of the qubits to a simple fiducial state, such as $|000\rangle$" [66]. It is very important to know the state of qubits before starting to do operations with them. This criteria requires the system to be able set the state of all qubits to a pre-determined value, such as $|000\rangle$ or $|111\rangle$. This way the user can know the state of the qubits before starting to apply some operations on them.

The third criteria requires the systems to have long decoherence times [66]. Decoherence is loss of information as a consequence of interaction with the environment. A generic state has the possibility to end up in $|0\rangle$ or $|1\rangle$ states in a certain time frame. Therefore, decoherence times should be longer than the gate operation times, assuming that there is enough time to apply gates before states transform into other unknown states. There are two widely used measures of decoherence in the literature. Longitudinal coherence time (T_1) is the time that it takes for a state in the excited state to decay to the ground state. Transverse coherence time (T_2) is the time it takes for either of the basis states to flip [78]. A Quantum Computer can also have qubits with different T_1 and T_2 times, making some of the qubits less favorable. Also, as it was previously discussed, trapped ion qubits outperform superconducting qubits in both T_1 and T_2 [72, 77]. These values are generally provided by companies as a measure of the quality of their qubits. Some examples of T_1 and T_2 times of several Quantum Computers are given in Table 3.1.

Table 3.1: Decoherence properties of some available Quantum Computers with different architectures as of 6 April 2021. These figures are subject to change as companies update their systems.

Device	Num. of Qubits	Avg. T_1 [s]	Avg. T_2 [s]	Avg. Gate time [ns]
IonQ [72]	11	10000	0.2	1231
Google Sycamore [70]	32	16.04×10^{-6}	N/A	12
Rigetti Aspen-9 [71]	32	29.67×10^{-6}	18.39×10^{-6}	60-160
ibmq_montreal (v1.9.7) [52, 77]	27	113×10^{-6}	122×10^{-6}	199-309

The next criteria requires systems to have universal set of gates. In classical computing, universal logic gate sets are defined as the set of gates that can implement any boolean function using finite amount of gates (e.g. {`AND`, `NOT`}, {`OR`, `NOT`}). Similarly, a "universal quantum gate set" is a set of gates that can implement any given unitary function using finite gates (e.g. {`H`, `CX`, $\pi/8$}) [78]. As a design choice, Quantum Computer providers usually determine a set of gates that can be implemented on a certain device and use those gates from the set to implement any other gates a user may require. For example, one can implement `CX`, `ID`, `RZ`, `SX` and `X` gates natively on ibmq_montreal (v1.9.7) and any other unitary gate can be implemented by combining

these gates with an operation called transpiling [52]. Definitions of these gates are presented in Appendix A.

The last criteria is "the ability to measure a specified qubit". Measurement is the process of reading out the state of the qubit. This process records the quantum state as classical bit. It is important for a Quantum Computer not to change the state of other qubits while measuring a qubit as it will damage the quality of the recorded result. On top of this, there can also be measurement errors on a single qubit, meaning that a state in $|1\rangle$ might be measured as a $|0\rangle$ state due to instrument noise. As an example, IBM reports the single qubit readout assignment error to be 6.0×10^{-3} on their ibmq_montreal device [77].

The two optional criteria involve Quantum communication. First requires the system to be able transmit the state of a qubit, that is used to transmit information between different quantum systems (flying), to a qubit that is on the device (stationary). The second requires to transmitting a quantum state accurately, such that the state does not change during the process.

There are many types of errors in QC, like those listed above. There are also various architectures and technologies with advantages and drawbacks. All of these factors makes it hard to quantify the quality of a Quantum Computer. To help with this situation, researchers from IBM proposed a volumetric benchmark called Quantum Volume (QV or $V_{\mathcal{Q}}$) [79].

3.2.3 Quantum Volume

QV determines a Quantum Computer's capacity to produce reliable heavy outputs with square circuits. A square circuit is a circuit with equal depth (number of gate layers) and width (number of qubits). QV is defined for a QC with m qubits as the logarithm of the largest possible square circuit with d(m) qubits, that has a probability of larger than 2/3 to produce heavy outputs as given in Eq 3.18 [79].

$$log_2 V_{\mathcal{Q}} = \arg\max_{m} \min(m, d(m)) \tag{3.18}$$

Using QV as a benchmark has become more attractive, compared to number of qubits, to show which company has the most powerful device. Recently, IBM announced ibmq_montreal, which is a 27 qubit device with a QV of 64 [77] and Honeywell announced their device, which has 6 qubits, with a QV of 64 [73]. Different hardware systems are better or worse at satisfying the DiVincenzo's criteria. There is still no clear leading technology that proved itself superior to others as of 2021.

Although QV became popular in some parts of the industry, there also has been discussions on whether the QV is enough to capture all properties of Quantum Computers. Some proposals have been made to extend the definition to deeper circuits [80]. Researchers are also working on error correction and error mitigation techniques to increase performance of these devices. It is also important to note that benchmarking Quantum Computers is an active field of research.

Quantum Computers built until today and accessible to researchers are very few in numbers. This limits prototyping new algorithms. Some of the QC companies provide access to researchers from all over the world, allowing them to use their devices from the comfort of their own laptops. IBM provides free access to multiple devices with varying number of qubits and QVs, while Amazon Braket provides paid access to QCs, built by D-Wave, Rigetti and IonQ. Counting both paid and free methods there are not more than 20 publicly available Quantum Computers at the moment. This situation led many researchers to build QC simulators, that uses classical processors (CPUs and GPUs).

QC simulators provide a way to build QC models without the need of a real device, but they are computationally expensive due to the size of the exponentially increasing Hilbert Space. There are many types of simulators that can focus on different aspects of the computation. State Vector/Density Matrix simulators keep the whole state representation, allowing users to estimate the state without a measurement, with the cost of high RAM requirements, while pure state simulators allow fast simulation by not allowing mixed states to occur. Some popular open-source libraries that allows users to easily simulate Quantum circuits are Qiskit [64], Pennylane [81] and Cirq [82]. Quantiki provides an extended list of these libraries [83].

3.3 Variational Quantum Classification

Although some early quantum algorithms such as Shor's factorization algorithm [54], or the HHL algorithm that solves certain sets of linear equations [57] promised a quantum speed-up, they are not implementable at large scales on current Quantum devices due to noise and limitations on the number of qubits [84]. For this reason, algorithms that can be implemented on NISQ hardware, such as Variational Quantum Algorithm (VQA) attracted a lot of interest from the QC community [85]. Flexibility of VQA methods led researchers to experiment with many types of problems from solving energy levels of a Hamiltonian with the VQE [58] to developing new ways to handle classical machine learning tasks with Variational Quantum Classifiers (VQCs) [60–62].

VQAs are hybrid algorithms that leverage both classical processing (CPUs, GPUs, etc.) and Quantum Processing Units (QPU). A VQA first takes inputs from a training set and encodes that into quantum states on the QPU according to the choice of ansatz (also called Parametrized Quantum Circuit (PQC)). The ansatz can also have parametrized and trainable gates that are independent from the input data. The ansatz is implemented and measured multiple times (the amount is called number of shots) to estimate an expectation value on the QPU. Then, the output value is fed to a cost function to determine the size of the error, which will be used by a classical optimizer to update the trainable parameters of the ansatz. After the parameters are updated, a new sample from the training set is fed to the algorithm and this process is repeated until the cost function reaches a minimum or a cutoff. A schematic of this iterative process can be seen in Figure 3.7.

The number of shots that is used estimate the expectation value has significant importance on the performance of VQAs. When the measurements are read from a QPU, they become classical information and can only hold values of 0 or 1. For this reason, to estimate an expectation value, the same circuit needs be implemented and measured multiple times to reduce statistical errors. M measurements allows estimation of an observable $\hat{C}$ with ϵ finite precision with the following condition [86];

$$M \geq \frac{Var[\hat{C}]}{\epsilon^2} \tag{3.19}$$

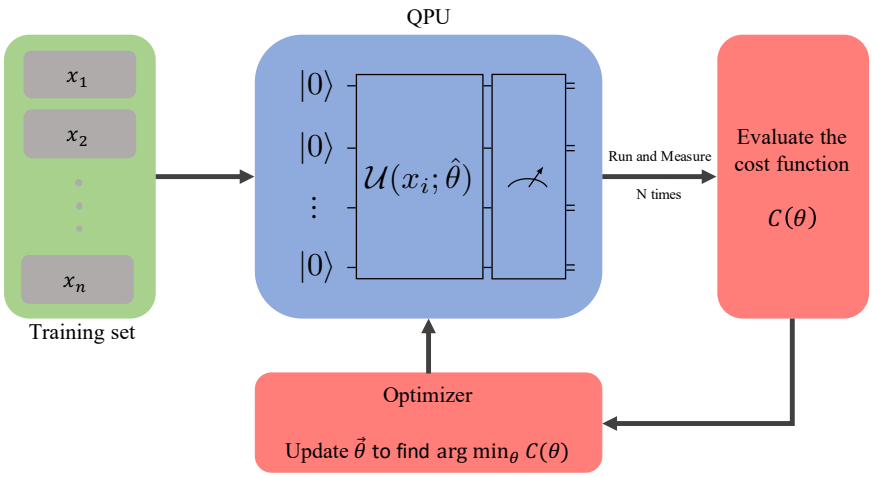


Figure 3.7: Training schematic of a generic VQA. A set of data is fed to the model sequentially. The QPU uses the data to execute the pre-determined ansatz. The output is obtained through measurements, which can be repeated N times. The output is evaluated with a cost function. The optimizer updates the parameters of the ansatz to minimize the cost function. The process is repeated until the cost value reaches to a minimum or a cutoff.

Then, according to Eq. 3.19, number of measurements scales with inverse square of the precision. Expectation values can be calculated analytically by using simulators. Most QC simulators also allow creating noise models to emulate the behaviour of the noisy hardware. However, this aspect of the simulators is not used in this work due to immense computational resources required to simulate Quantum circuits under noise.

An example test scenario can be created by using a simple circuit. In this case, we used the default Pennylane [81] simulator as it allows for both simulating Quantum Circuits as well as access to most of the publicly available hardware. Fig. 3.8 shows the absolute errors (absolute value of the difference) of different scenarios with respect to the analytical expectation value of an arbitrary but same circuit. The plot shows the errors obtained with different shots using the simulator. As expected, the error levels of 0.1 is achieved with shots close to 100. The plot also shows multiple results obtained from various QC hardware with 1000 shots, and is a good indication of their noise level. It is important to note that the errors calculated are not a benchmark as they are results of a single circuit and do not represent devices' performances.

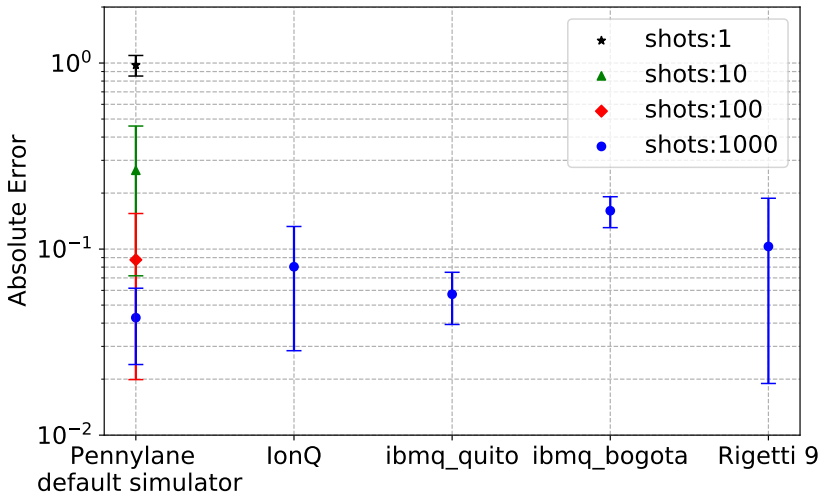


Figure 3.8: Absolute errors of expectation values of an example Quantum circuit[1] with different shots. The errors are calculated with respect to analytical expectation value obtained from the default Pennylane simulator and error bars represent $\pm$ standard deviation of 10 independent runs.

The optimization process of VQAs are similar to classical DL methods. This led researchers to use VQAs for classical machine learning tasks such as classification, where one tries to assign a label to each data sample [85]. These methods have been called many names in the literature such as Quantum Circuit Learning [60], VQCs [61] and Quantum Neural Network (QNNs) [62]. It is important to note here that the name QNNs are given only due to their resemblance to classical Neural Networks and they do not always implement structures such as perceptrons, which are the building blocks of Neural Networks.

The following sections give an in depth look to the details of the VQCs and their constituents.

[1]The Quantum circuit used in this example is a 4 qubit and 1 layer configuration used in Section 3.3.3.3 and is plotted in Fig. 3.19.

3.3.1 Information Encoding

Information or data is the input of all machine learning models. As a famous saying goes "garbage in, garbage out", quality of data and how it is used is an important factor on the performance of ML models. This situation persists in Quantum Machine Learning models as well. In QC, a new type of data, called "quantum data", holds the information of a quantum system captured by quantum sensors and stored in qubits [87]. It is also possible for qubits to store classical information. Storing classical information requires applying a certain set of operations (set of gates) to qubits and this process is called information encoding or embedding. An embedding of data x is realized by a unitary S_x that acts on the initial $|0\rangle^{\otimes N}$ state with N qubits can be expressed as [88];

$$|\phi\rangle = S_x |0\rangle^{\otimes N} \tag{3.20}$$

There are certain properties of a good embedding [88]. First of all, an embedding should have polynomial number of gates in the size of the dataset and the number of qubits. Then, the embedding should be bijective to provide a unique representation for all of the data samples. NISQ hardware brings additional restrictions to the embedding methods with three criteria. First, circuits with subpolynomial depth due to low decoherence times. Second, hardware efficient circuits due to low connectivity. Hardware efficient circuits respect the connectivity of the qubits when applying multiple qubit gates. Third, use of native gates to reduce the total depth of circuits. There are multitude of proposed methods to encode classical information to qubits, such as basis, amplitude and angle embeddings, which will be given as examples here.

Basis encoding embeds the information to binary states with a finite precision τ. With this encoding, m binary strings of x with N features can be embedded in a superposition using $N \times \tau$ qubits. There are linear algorithms that can implement such superposition states [87, 89, 90]. However basis encoding results in a very sparse amplitude matrix which makes it inefficient at large scales [91]. An example of basis encoding x^1 = (11,

01) and x^2 = (00, 11) is;

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}}|1101\rangle + \frac{1}{\sqrt{2}}|0011\rangle \tag{3.21}$$

Amplitude encoding embeds data in the probability amplitudes of states. The amplitudes needed to be divided to $\|x\|_2^2$ to ensure normalization. The state, where a data vector $x \in \mathbb{R}^N$ is encoded using only N qubits, can be expressed as in Eq.3.22 [88]. In contrast to basis encoding, amplitude encoding requires $\mathcal{O}(1/|x_i|^2)$ measurements on average, in order to have enough statistics to discriminate the amplitudes. Another downside of using amplitude encoding is the need for extensive classical computation. Similar to basis encoding, amplitude encoding can also be implemented in linear time, but they both require many expensive multi-qubit gates [91]. A full amplitude encoding can go further and make use of all available amplitudes saving some number of qubits, as it would only require $log_2 N$ qubits. However, such an embedding requires $\mathcal{O}(2^N)$ time to prepare [91].

$$|\mathcal{D}\rangle = \frac{1}{\|x\|_2^2}\sum_{i=1}^{N} x_i |i\rangle \tag{3.22}$$

Angle (Qubit) encoding provides an embedding that needs constant depth and easy to implement circuits. In this embedding, where a data vector $x \in \mathbb{R}^N$ can be encoded using N qubits and N single qubit R_Y gates as given in Eq. 3.23 [88]. Angle encoding provides flexibility in choice of the rotation axis to embed data. An example embedding of 1000 data points uniformly sampled between [0, 2π], plotted on the Bloch Sphere can be seen in Fig. 3.9.

$$|\mathcal{D}\rangle = \bigotimes_{i=1}^{N} cos(x_i)|0\rangle + sin(x_i)|1\rangle \tag{3.23}$$

Dense angle embedding further improves the angle embedding by exploiting the phase of the states to embed the data vector $x \in \mathbb{R}^N$ using N/2 qubits with the small cost of additional single qubit gates. The mathematical representation of this embedding is

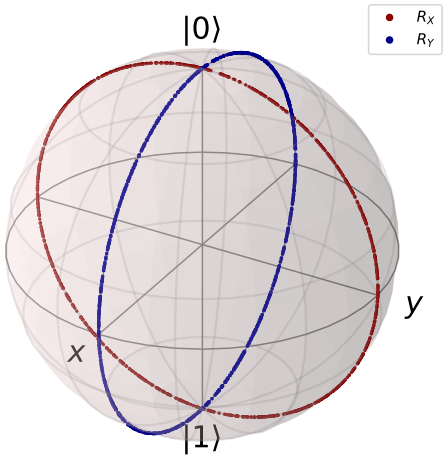


Figure 3.9: Angle embedding representations on the Bloch Sphere. Single qubit encoding with R_Y is plotted with blue circles and R_X in red.

given as [88];

$$|\mathcal{D}\rangle = \bigotimes_{i=1}^{N/2} cos(\pi x_{2i-1}) |0\rangle + e^{2\pi i x_2} sin(\pi x_{2i-1}) |1\rangle \tag{3.24}$$

There are many other types of embedding such as the general qubit embedding, which generalizes the angle embedding and Feature Maps that creates entanglements of data points between qubits. Recently, it has also been shown that the choice of the embedding can help avoid hardware noise, as some embeddings are more robust to certain noise sources [88]. There are also approaches where the embedding is repetitively applied in between variational layers [92], and it has been showed that the selection of embedding impacts the expressive power of a quantum model [93]. It is for sure that, the choice of embedding is an integral part of building a VQC.

3.3.2 Parametrized Quantum Circuits

Parametrized Quantum Circuits (PQCs) are tunable circuits that consists of a set of trainable parameters and is a prominent factor in determining the learning capacity of a VQA. There are many proposed ways to build a PQC. First, there are mathematically

driven, fixed architectures that focus on solving certain problems such as the Quantum Approximate Optimization Algorithm (QAOA) [94]. Then, there are PQC models that are built to express physical systems such as the Unitary Coupled-Cluster Singles and Doubles (UCCSD) ansatz [95]. This is followed by models inspired from physical systems such as Matrix Product State (MPS) [96], Tree Tensor Network (TTN) [97] and Multi-scale Entanglement Renormalization Ansatz (MERA) [97]. Lastly, there are hardware efficient circuits models [98]. The last two types of PQC models are suited best for machine learning tasks as their architectures are problem agnostic.

First of these two categories can be called hierarchical architectures. The Hamiltonians that are defined by some of these PQCs were already in use in quantum many body physics. MPS for instance, can be used to express quantum states of many particles that lie on a chain in one spatial dimension (1D) formation. Similarly, TTNs can express quantum states of particles that lie on a tree structure [99]. These types of PQCs start with using some certain number of qubits, then gradually decrease the number of qubits used in each layer. Their architecture is determined by the type of interactions and the number of qubits. Towards the end of the algorithm, the interacting number of qubits reduces to one and measurement is performed on this qubit. This architecture allows these type of circuits to be good PQC candidates in the NISQ era, as they have limited interaction between qubits and are not very deep. These circuits are often built with the help of 2 qubit gate blocks which both have entangling gates such as (e.g. `CX`, `CZ`) and parametrized gates (e.g. R_Y, R_X). An example construction of such a 2 qubit gate is given in Fig.3.10. Examples of these architectures are presented in their 4 qubit configurations can be seen in Fig.3.11.

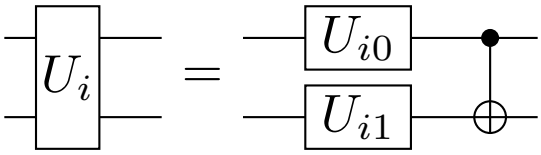
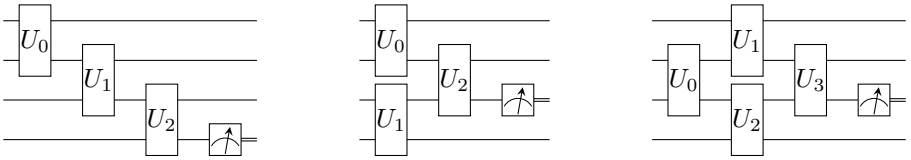


Figure 3.10: An example construction of a two qubit parametrized gate.

The second type of PQCs provides a more flexible and generic architecture. They consist of major building blocks, called layers. First of them is the parametrized layer, which consists of single qubit gates that might consist of both parametrized



(a) MPS [96] (b) TTN [97] (c) MERA [97]

Figure 3.11: Some of the hierarchical architecture types of PQCs in their 4 qubit configurations.

and non-parametrized gates. The second layer type is the entangling layer and it is responsible for creating interactions between qubits with the help of entangling gates. When these two layers are combined, they are referred as a PQC layer, and a complete PQC consists of multiple layers. At the end of the circuit, any qubit can be measured depending on the design choice. These facets allow this architecture to express a very wide range of Hamiltonians, while still allowing NISQ applications. An example configuration of these layers is given in Fig.3.12.

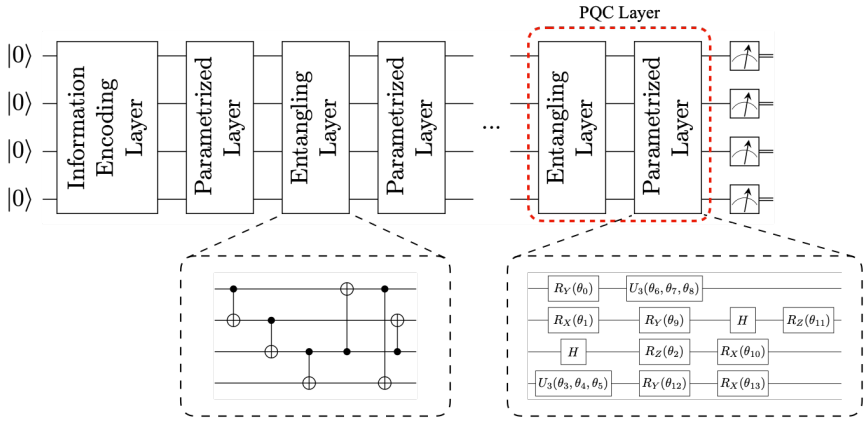


Figure 3.12: A generic variational Quantum circuit with an arbitrary PQC. An information encoding layer encodes the classical information on the Hilbert Space of qubits. Then a series of parametrized and entangling layers are used to transform the encoded state. At the end, the quantum state is measured.

There are many ways to build both of these layers. Their design is usually restricted by the hardware. In low connectivity cases, entangling gates that only act on the nearest neighbour qubits might be preferred (Fig. 3.13a). However, in certain cases,

the hardware might allow all-to-all connections (Fig. 3.13b). The parametrized layers are also very flexible. Design choices such as axis and amount of rotation gates remain. While one can only use a single axis rotation in each layer (Fig. 3.13c), it is also common to stack different axis rotations in a single layer (Fig 3.13d).

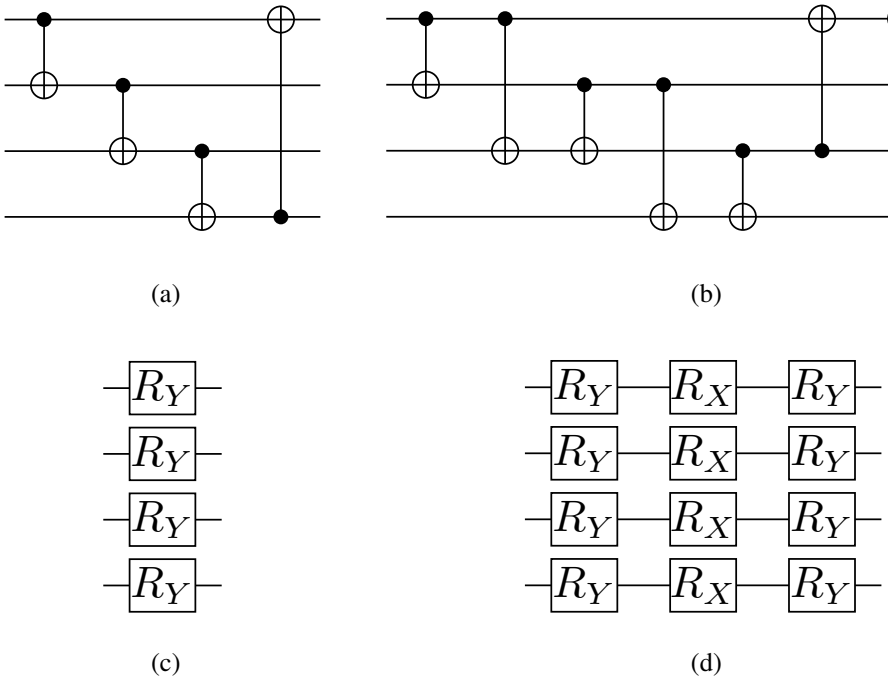


Figure 3.13: Example 4 qubit toy configurations for entangling and parametrized layers. An entangling circuit with nearest neighbour (2D) interactions (a). An entangling circuit with all-to-all interactions (b). A PQC with a single layer of gates (c). A PQC with multiple layers of gates (d).

It is important to point out here that the architecture presented here is not a fixed blueprint. There are instances where the entangling layers include parametrized gates such as controlled rotation gates (e.g. CR_X) and parametrized layers that consists of two qubit gates. The checkerboard (nearest-neighbour (2D)) ansatz given in Fig. 3.14 is one example to these kind of models [98].

There has been significant progress in designing and testing different PQCs in the last few years. Most of these models has been tested and showed promising performance in various machine learning tasks [100]. However, the question of which model is best for any given problem still lacks an answer. There has been several attempts to

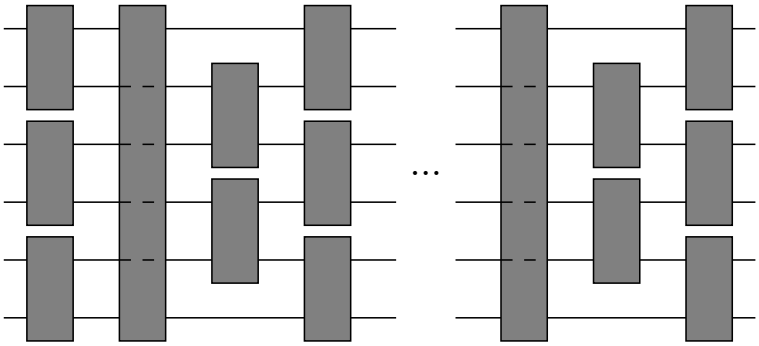


Figure 3.14: A nearest-neighbor (checkerboard) PQC architecture. The gray boxes represent 2 qubit parametrized gates which may include parametrized and/or entangling gates. Figure is adapted from [98].

quantify properties of PQCs models. One of the most interesting of these attempts introduced two descriptors to quantify a PQC model's ability to express different states and create entangles states called expressibility and capacity of entanglement [101].

Expressibility measures a PQC model's ability to represent pure states uniformly in the Hilbert Space. This metric is calculated numerically by comparing fidelities of statevector samples from the PQC model and from Haar random matrices. Haar random matrices are unitary matrices that satisfy the Haar measure for uniformity [102]. Fidelity is a measure that quantifies the closeness of two quantum states. At each sampling, two random instances from the PQC is obtained and their fidelity is computed. This operation is also performed for Haar random states.

$$\mathcal{F} = |\langle \psi_\theta | \psi_\phi \rangle|^2 \tag{3.25}$$

These samples are collected until there is enough statistics to obtain a probability distribution of fidelities that can represent the distribution well enough. Finally, Kullback-Leibler (KL) divergence [103] of these distributions are calculated to obtain the value of the Expressibility descriptor [101].

$$E = D_{KL}(\hat{P}_{PQC}(F;\theta) \parallel P_{Haar}(F)) \tag{3.26}$$

The *E* value gets smaller as the PQC model gets better at expressing more states in the Hilbert Space. This makes *E* hard to compare for highly expressive circuits. Recently,

researchers started using an altered version (*E'*) such that, it is the negative of the logarithm of *E* [104].

$$E' = -log_{10}(E) \tag{3.27}$$

This way, *E'* grows as the PQC model gets more expressive. In Fig. 3.15, we present a comparison of these values for two different PQC models with various number of layers with 5000 samples for 4 qubits that represent a large enough dataset so that the statistical error bars are negligible [101]. Model `a+c` uses circuit in Fig. 3.13c as its parametrized layer, while Model Model `a+d` uses circuit in Fig. 3.13d. Both of these models use the same entangling layer (Fig. 3.13a) to create a controlled expressibility test. The increase of *E'* at each layer shows that additional gates increase a model's expressive power, but this value saturates after certain amount of layers. As expected, the model with a higher number of parameters and degrees of freedom (Model `a+d`) can access more states, resulting in a larger *E'* value.

The second descriptor is the Entangling Capability. As the name suggests, this descriptor quantifies the model's ability to create entangled states. This metric is also numerically calculated by averaging Meyer-Wallach entanglement measures [105] of samples with high enough statistics [101]. Meyer-Wallach entanglement measure (Q) calculates the average linear entropy (i.e. $1 - \{\rho^2\}$) of all the single qubit reduced states [106]. For example, a fully entangled 2 qubit state $\left(|\Psi\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}\right)$ has $Q(|\Psi\rangle) = 1$ and a state with no entanglement (e.g. $|\Phi\rangle = |01\rangle$) has $Q(|\Phi\rangle) = 0$. Then, the Entangling Capability is calculated by taking the averages Meyer-Wallach entanglement measure over the set of sampled states as given in Eq. 3.28 below [101].

$$Ent = \frac{1}{\|S\|} \sum_{\theta_i \in S} Q(|\psi_{\theta_i}\rangle) \tag{3.28}$$

We can compare different entangling layers similar to the comparison in Fig. 3.15 with same statistics. For this purpose let's consider two models with same parametrized layers (Fig. 3.13c). Model `a+c` uses the nearest neighbour fashion entangling gates (e.g. Fig. 3.13a), while Model `b+c` uses an all-to-all type entangling layer (e.g. Fig. 3.13b). As it can be seen from the figure, the Entangling Capability increases

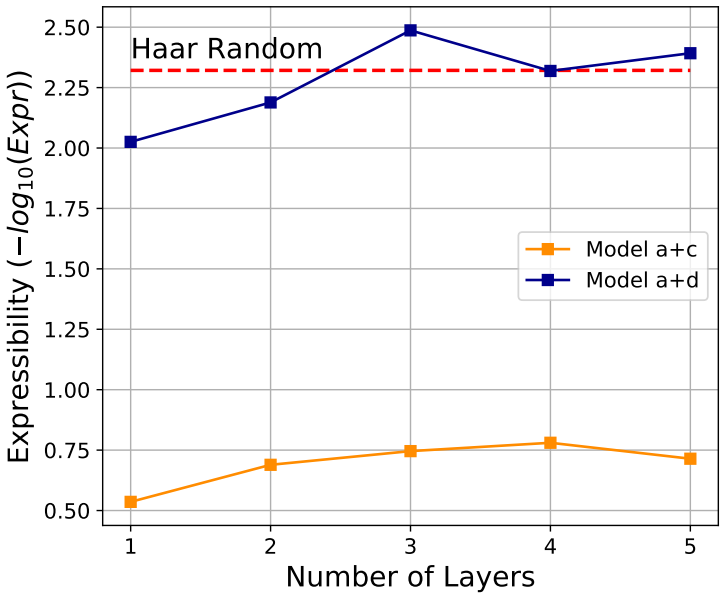


Figure 3.15: Model Expressibility vs. number of layers comparing two PQCs with different parametrized layers.. Expressibility of Model `a+c` is plotted in orange and Model `a+c` in blue for different number of layers. Both models saturate after some certain number of layers. Model `a+c` reaches to values of Haar Random states with more layers, while Model `a+c` saturates to a lower value.

with increasing number of layers and starts to saturate after some point. Both of the models converge near the Entangling Capability of Haar Random states, while it is hard to reach the maximally entangled limit. Results of Fig. 3.15 and 3.16 show that as a model has more layers, having more parametrized and entangling gates, the values reach to values of Haar random unitary transformations.

Expressibility and Entangling Capability provide a systematic method to quantify PQC model's properties. However, both of these metrics are still not enough to answer the initial question, which was how to choose a PQC model. Although these descriptors provide a starting point for PQC design, they do not cover the whole picture. For example, they do not account for the type of measurement and post processing, which has been shown to have a significant effect on performance [95]. Secondly, they evaluate the states starting from a $|0\rangle^{\otimes N}$ initialization, which is not the case when there is an information embedding layer before the PQC. For these reasons, in order to maximize the benefit of using these models, there is a need for new metrics that can

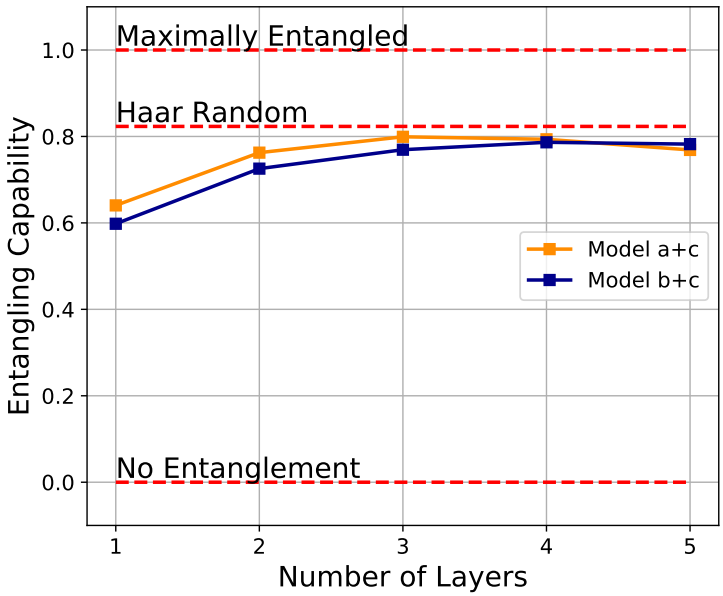


Figure 3.16: Model Entangling Capability vs. number of layers comparing two PQCs with different entangling layers. Entangling Capability of Model `a+c` is plotted in orange and Model `b+c` in blue for different number of layers.

cover a larger set of PQCs and their properties.

3.3.3 Training Classical and Quantum Neural Networks

Similar to DL algorithms, QNNs require an iterative process (training) to find optimal parameters that minimizes a cost function for a given set of inputs, as it was previously shown in Fig. 3.7. In the past two decades, many techniques to train DL methods were developed and luckily, most of them are applicable to QNN training [107].

3.3.3.1 Optimization

The training processes consist of 4 elements: data, model, cost function and optimizer. We discussed details regarding the data and the model to some extent until this point. The other two elements are investigated in this section.

Cost function (or loss function) quantifies a model's performance over a given set of

inputs. If the task is a supervised learning task such as classification, then the model's output is compared against the input's class label. Binary cross entropy is a robust loss function, that is widely used in many classical DL applications. It also found its way into Quantum Machine Learning tasks and proved itself very useful in many occasions. Although there are many other loss functions, to keep the consistency with the literature, this work employs binary cross entropy in all training processes. The definition of the binary cross entropy loss function for a binary classification task, where, N is number of data samples, y_i is the binary truth label (i.e. $y_i \in \{0, 1\}$) and $\hat{y}_i$ is the model output (i.e. $\hat{y}_i \in [0, 1]$) for the i^{th} data sample is;

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i log(1 - \hat{y}_i) + (1 - y_i) log\hat{y}_i \tag{3.29}$$

The binary cross entropy loss function scales the predictions with a logarithm function allowing a smooth penalty and award for wrong and true predictions respectively. This scaling for two classes with different predictions is presented in Fig. 3.17.

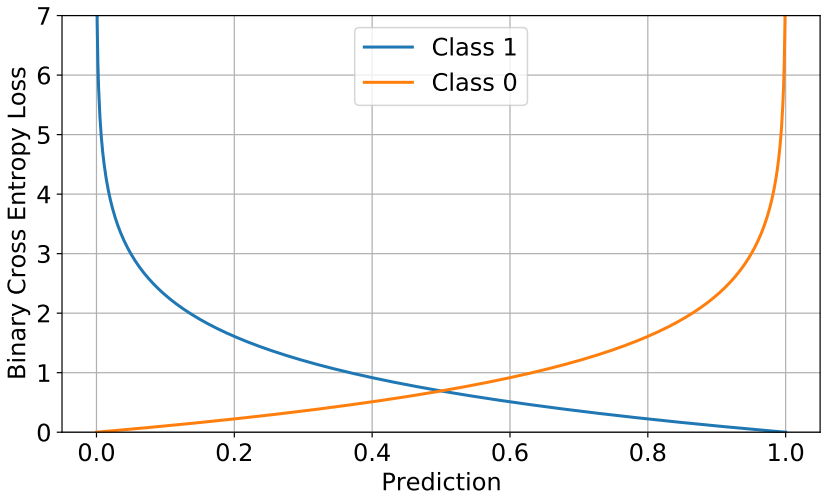


Figure 3.17: Binary cross entropy for different predictions of a sample. Class 0 is shown in blue and the true prediction of it should be 0, while the true prediction of Class 1 is expected to be 1.

The final element in the training process is the optimizer. Optimizer combines the other three ingredients, which were data, model and loss function, to update the

parameters of the model with the goal of optimizing the loss function (it can minimize or maximize it depending on the definition). For a model that is parametrized with θ, the optimization task is expressed as follows, where x is the dataset, y is the labels of each sample from the dataset and L is the loss function;

$$\underset{\boldsymbol{\theta}}{\text{argmax}} \min(L(x, y; \theta)) \tag{3.30}$$

The loss functions of DL and QNN algorithms are generally non-convex, which means that the loss landscapes consist of local minima. This makes non-convex optimization an NP-hard problem, meaning that it takes exponentially more time to find the global minima with respect to number of parameters. However, this is not a big problem, as in most cases, a local minima within a certain error threshold is sufficient. Popular non-convex optimization algorithms can be viewed in two groups, which are derivative based and derivative free. Now, they will be briefly explained.

First group of algorithms use derivatives of the loss function. The partial derivatives of each parameter with respect to the loss function is used to guide the optimizer to optimal values. Training the model continues as the loss function gets better values at each iteration. For example, Gradient Descent, the simplest form of these algorithms, updates the parameters with the following rule, where η is the learning rate;

$$\theta \coloneqq \theta - \eta \nabla_\theta L(x, y; \theta) \tag{3.31}$$

In the early days of Machine Learning and DL, simplicity of the Gradient Descent allowed fast prototyping. However, this algorithm couldn't handle larger and more complex models. Therefore, researchers constantly searched for ways to improve. Adaptive Moment Estimation (ADAM) algorithm was born as a product of several development stages of algorithms [107]. ADAM proved itself in many occasions and was shown to be robust. This led many researchers to adapt ADAM and use it for most of the DL and also QNN training schemes today.

ADAM adapts the learning rate for each parameter which allows a smooth movement on the loss landscape. This is done by keeping a running average of gradients and their

second moments. The gradients' running average (m_t) can be expressed with;

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta L(x, y; \theta) \tag{3.32}$$

where β_1 is the first decay term, and if we divide this term with $(1 - \beta_1)$, we obtain;

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{3.33}$$

The running average of second moments of gradients (v_t) can be written as;

$$v_t = \beta_1 v_{t-1} + (1 - \beta_2)(\nabla_\theta L(x, y; \theta))^2 \tag{3.34}$$

where β_2 is the second decay term, and if we divide this term with $(1 - \beta_2)$, we obtain;

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{3.35}$$

Then, these two terms are used to adapt the learning rate such that θ_t represents the parameters at iteration t;

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3.36}$$

where ϵ is a very small value (e.g. 10^{-8}) to avoid a possible division by zero. The three so called "hyper-parameters" of ADAM has recommended values of 0.001 for η, 0.9 for β_1 and 0.999 for β_2 and these are generally used as the default values in many open-source machine learning libraries such as Tensorflow [108], PyTorch [109] and Scikit-learn [110].

Although ADAM is a very robust and well performing algorithm, it requires a lot of classical resources to keep track of all the running averages. It also has a problem common to all gradient descent type algorithms, which is the computational cost of calculating gradients of all parameters. Novel classical DL algorithms might contain

millions of parameters, which is a burden on the optimization process. As it was discussed in Section 2.3.2, GPUs help speed-up these calculations and allow researchers to train large models with the currently available computational resources [36–38]. However, this issue persists in optimization of VQA and presents itself as a significant problem. This matter will be discussed in the next section.

The second group of optimizers in non-convex optimization is called derivative-free optimizers. These algorithms basically explore the loss landscapes by evaluating points on it and do not rely on the derivatives of the loss function [111]. Some popular examples include Bayesian optimization [112], particle swarm optimization [113] and Constrained Optimization by Linear Approximation (COBYLA) [112]. Although these optimizers have a computational advantage by not calculating the gradients, their ability to find good solutions has been shown to be limited as the system size increases [114]. This is one of the major reasons why they are not widely used to train classical DL algorithms. Although derivative-free methods had been used to train VQAs [115, 116], their use in hybrid models where a VQA is combined with a classical DL algorithm is generally not preferred.

3.3.3.2 Gradients in Variational Quantum Algorithms

Optimizers such as ADAM needs partial derivatives (gradients) of all parameters with respect to the loss function. They can be obtained using back propagation [117]. In classical DL, analytical gradients of all parameters can be computed with this method. However, the situation is different in VQAs. Although one can get analytical gradients and back propagate with the help of simulators, the gradient can only be estimated numerically using hardware due to finite sampling [118].

There are proposals to estimate the gradients on QC hardware based on finite difference methods (parameter shift rules) [81, 86]. The cost function of a PQC, which is parametrized with the unitary operation $U(x;\theta)$ and observed with the $\hat{M}$ operator, can be expressed as;

$$C(x;\theta) = \langle 0|U(x;\theta)^{\dagger}\hat{M}U(x;\theta)|0\rangle \tag{3.37}$$

Then, if the observable has two distinct eigenvalues, the parameter shift rule gives the gradient as [119];

$$\nabla_{\theta_i} C(x;\theta) = r[C(x;\theta_i + \pi/4r) - C(x;\theta_i - \pi/4r)] \tag{3.38}$$

where the r is a constant number. Eq. 3.38 shows that, to calculate the gradient of a single parameter, two different executions of the same circuit with shifted parameters is required. This situation adds one more problem to train VQAs, because to obtain gradients of a PQC with m parameters, 2m different PQC executions are needed. Naturally, the parameter shift rules also suffer from finite sampling with the constraint given in Eq. 3.19. In Fig. 3.38, the gradient errors for a single parameter of a PQC with different settings are compared, using a simulator and various NISQ hardware similar to the comparison in Fig. 3.8. The results here are in alignment with the constraint set by Eq. 3.19 and they show that NISQ hardware can estimate gradients but with large errors. All results are obtained 10 times and their mean are plotted.

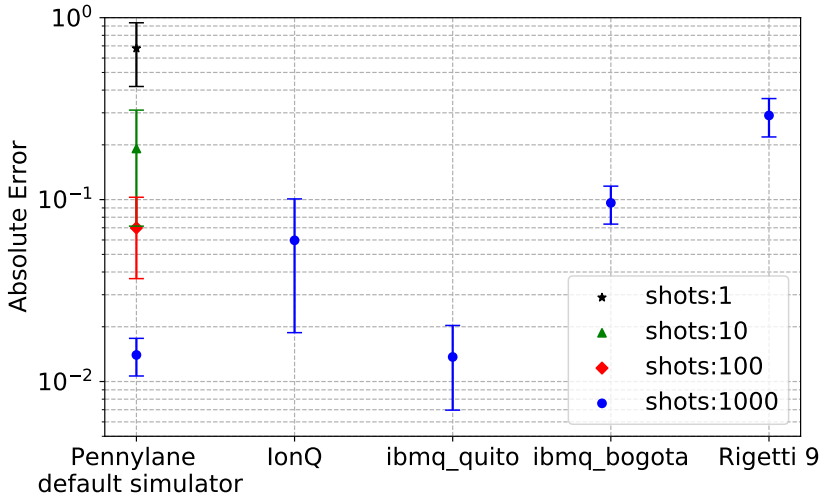


Figure 3.18: Absolute errors of estimated gradient values of a PQC[2] with different number of shots. The errors are calculated with respect to the analytical value obtained using the default Pennylane simulator and error bars represent standard deviation of 10 independent runs.

Training a model, whether classical or hybrid, comes with many challenges. Some of

[2]The Quantum circuit used in this example is a 4 qubit and 1 layer configuration of the PQC plotted in Fig. 3.19.

these challenges were already mentioned. There is another important aspect called the vanishing gradient problem [41] that was discussed in Section 2.3.3, which requires careful handling especially when training a VQA. The vanishing gradient problem was initially observed in classical DL applications such as RNNs but is also observed in VQAs and generally called the Barren Plateaus (BPs) [120], which will now be explained.

3.3.3.3 Barren Plateaus in loss landscapes

RNNs have a layered structure similar to some PQC models. The size of the gradients decrease exponentially with the increasing model size, which prevents optimizers to propagate to further distances on the loss landscape. The Barren Plateau (BP) name is given due to the flat shape of the loss landscape [120]. To mitigate the problem in the classical case, researchers introduced a new model called LSTM that prevented the vanishing gradients, which was discussed in Section 2.3.3 [43].

A similar situation also arose for training PQCs. It has been shown that some PQCs have exponentially vanishing gradients with increasing number of layers and number of qubits [120]. This can easily be shown with numerical simulations of a PQC which fits that definition, and a simple PQC with 4 layers, given in Fig. 3.19 is chosen as an example. Expectation values of the circuit with various number of qubits are obtained only by varying the first parameter ($\theta_{1,1}$) and keeping the others fixed. The values are plotted on Fig. 3.20. The plot clearly shows how the expectation values get flatter with increasing number of qubits resulting in a BP.

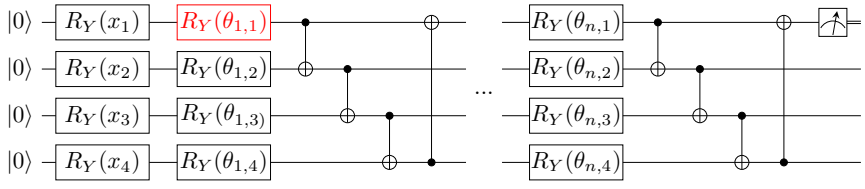


Figure 3.19: The layout of the PQC that is used to obtain gradient samples. The gate shown in red indicates the parameter whose gradient has been taken.

To give another example of this effect, 1000 samples of gradients of $\theta_{1,1}$ with varying number of qubits and layers are collected. The variances for each instance is plotted

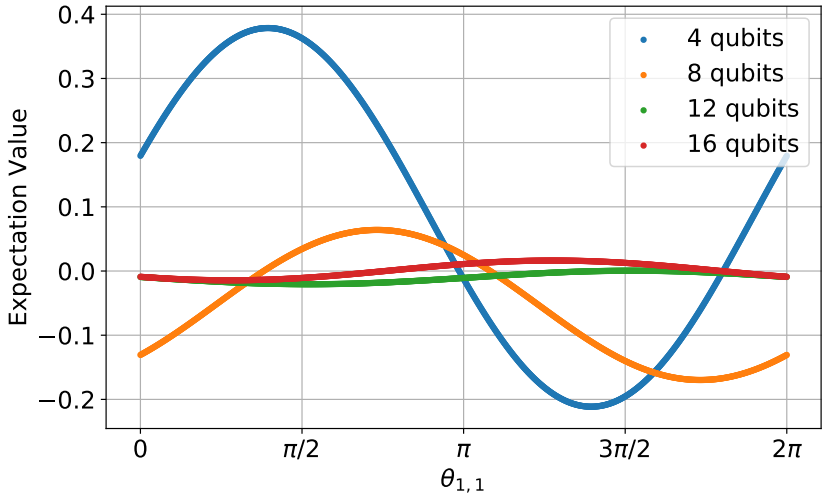


Figure 3.20: Expectation values of the PQC in Fig. 3.19 with 4 layers using various number of qubits vs. the $\theta_{1,1}$ parameter.

in Fig. 3.21. This plot is a clear indication of BP formation, as the variance of the gradients decreases exponentially with the number of qubits. The decrease can also be observed with increasing number of layers. This phenomenon makes training larger models difficult and sometimes impossible.

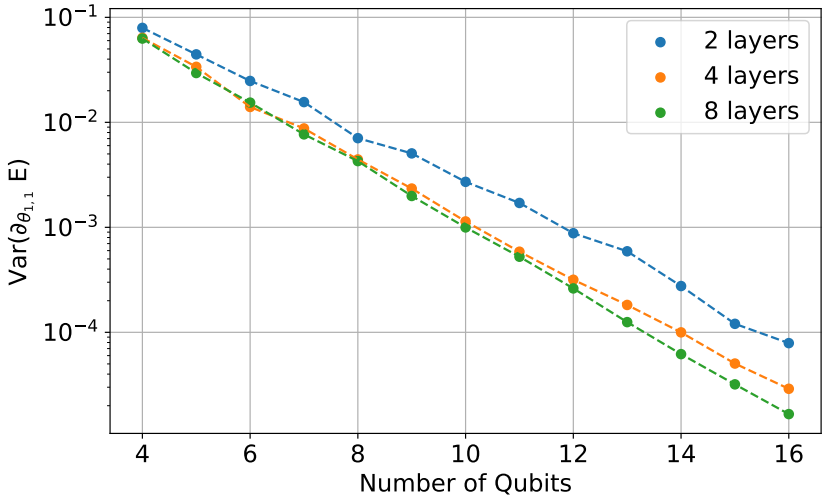


Figure 3.21: Variance of gradient of a single parameter of a PQC vs. number of qubits and layers. The variances of the gradients that belongs to the first parameter of PQC shown in Fig. 3.19. Different colors indicate number of layers the PQC contains.

BPs are one of the most significant obstacles against scalable VQAs [85]. In recent years and especially last year (2020), there has been progress in this field. There is still a lot to discover and unlocking the full potential of VQAs rely on better understanding this phenomenon.

Some of the recent progress that involves BPs can be listed as follows. There is an initialization strategy that allows PQCs to escape BPs, here the PQC is initialized to act as an identity gate [121]. Also, some architectures (TTN like architectures) were shown to be resistant to BPs [122]. The rate of entanglement propagation between qubits causes BPs [123], which might be the reason why TTN-like architectures are more resistant, as they generally limit interactions between qubits significantly. Last but not least, certain types of noise can create BPs, which might forbid training larger models in the NISQ era [124].

Variational Quantum Classification is a very young and promising method. There is still a lot to understand, test and validate. In this work, we try to shed light to some of the effects while trying to achieve a real life scale task.

CHAPTER 4

METHODOLOGY

In this Chapter, the methodology of the proposed QGNN model to solve the particle track reconstruction problem is presented. The first section contains details of the input dataset's manipulation. Next, the proposed QGNN model is laid out with its design details. Finally, some quality metrics that are used to quantify the performance of the proposed QGNN model are presented.

4.1 Data Pre-processing

The publicly available TrackML Challenge dataset provides 10000 events with a Pile-up of 200 events to emulate the HL-LHC conditions [11]. For this work, due to computing time limitations, only 100 events were selected. The particulars of the dataset have been already discussed in section 2.4. Now, how this dataset is manipulated and later used to train and test the proposed model will be explained.

In most Quantum Machine Learning applications it is tedious to work with large datasets due to long duration of simulations [83, 125], limited QV of NISQ devices [79], as well as limited access to NISQ hardware [52, 53]. Thus, a pre-processing step to simplify the dataset is necessary. For this purpose, only 100 events were selected and randomly divided in half as the train and test sets for the QGNN model.

The HEP.TrkX project restricts the tracking detectors to cover only the barrel region of the TrackML detector [5] to reduce total number of particles and make tracking more straightforward. This is done in order to both reduce number of possible tracks and resolve the ambiguity in identifying the particle trajectories. This work follows

the same approach and uses the TrackML files which were defined in Table 2.2. The geometry in 2 cylindrical coordinates (r,z) and the selected barrel region can be seen in Fig. 4.1 as it was explained in Chapter 2.2.

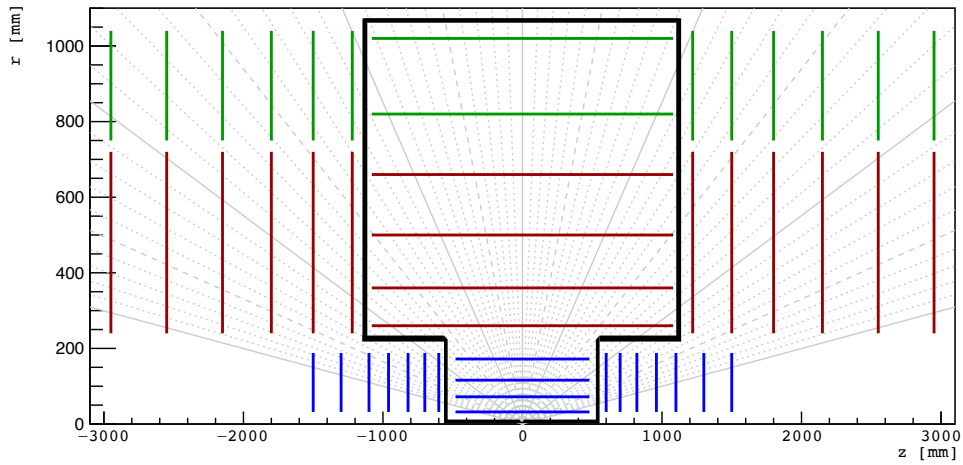


Figure 4.1: 2D projection of the TrackML detector geometry. The region delimited with the black lines indicates the detector layers used in this work. Drawing is adapted from [11].

Next in pre-processing, a restriction ($pT > 1\ GeV$) is applied on the particles' transverse momentum (pT) to further reduce the number of particles. The transverse momentum pT is the particles' momentum along the transverse (x-y) plane, and determines how much they bend under the influence of a magnetic field. Particles with a low pT bend more, while particles having a high pT bend less, *cf.* Eq. 2.1. Limiting the minimum allowed pT reduces the number of particles significantly, while not losing any interesting high pT transfer collisions, which are the main interest of physics analyses. But, this comes with an unintended side effect, which makes the job of the model easier, as more curved (low pT) tracks are harder to track. In order to be able to compare our results to HEP.TrkX, the same data pre-processing is also used for it. The pT distribution of the measurements (hits) of particles from the selected 100 events used in this work can be seen in Fig. 4.2 and the total number of hits for train and test datasets are given in Table 4.1.

After reducing the number of tracks, the events are converted to graphs. Particle hits are the nodes of the graphs and the track segment candidates are defined as edges at

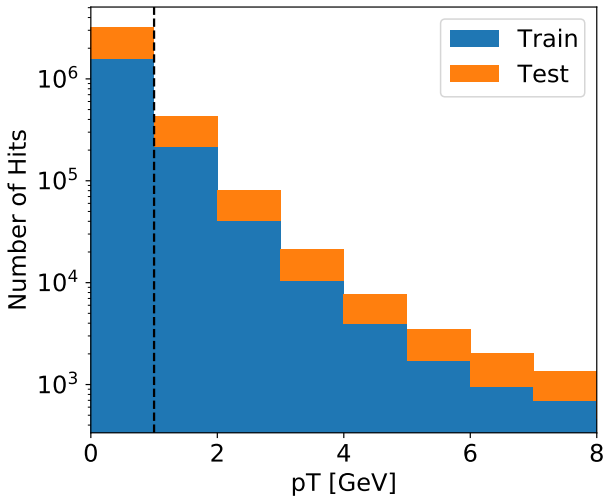


Figure 4.2: Stacked histogram of number of hits vs pT. Hits of particles that passed through the barrel region of the TrackML detector from the selected 100 events are plotted. The dashed line represents the 1 GeV pT threshold used in order to reduce total number of particles and tracks. Values are plotted until 8 GeV for visual purposes.

Table 4.1: Number of particle hits before and after the pT cut in the train and test datasets.

Dataset	All hits	Hits > 1 GeV
Train Set	1,865,514	278,791
Test Set	1,900,858	279,514
Total	3,766,372	558,305

this stage. Then a set of criteria (cuts) is applied to all possible graph edges to create the final graphs. Here, the goal is to have as few fake edges as possible, and as many true edges as possible.

The restrictions are defined in a cylindrical coordinate system, which is widely used in HEP to leverage the symmetries of the detectors. We follow the same convention and present some of the definitions visually in Fig. 4.3 for further clarification. Particles travel along the z-axis and collide near z=0. Then, the products of the collisions travel away from the interaction point and leave signals on the detectors. Pseudorapidity (η)

is a measure for the angle to the z-axis in HEP. Its definition is given in Eq. 4.1, where ϕ is the angle to the z-axis.

$$\eta = -\ln\left[(\tan(\phi/2))\right] \tag{4.1}$$

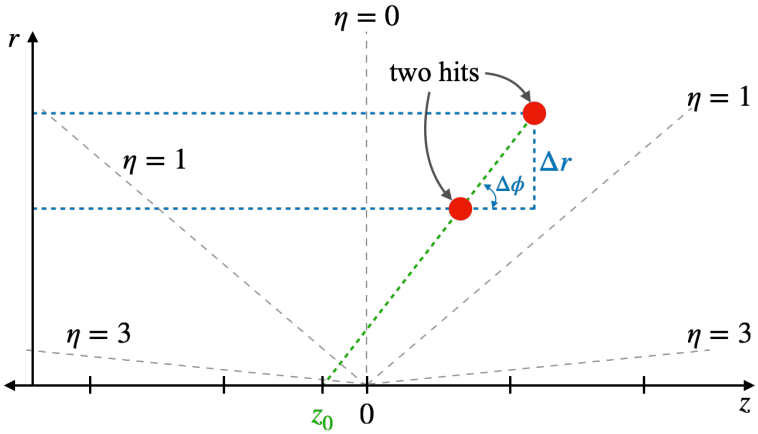


Figure 4.3: A drawing of the cylindrical coordinate system for particle collisions. The beam is on the z-axis and the particles collide near z=0. The r axis is the projection of the transverse (x-y) plane.

The first criterion of graph construction is to keep only the edges that connect nodes from consecutive detector layers. These edges are the track segment candidates. An example drawing, which shows this selection process is given in Fig. 4.4, where edges are drawn with dashed black lines.

Next, tracks with η larger than 5 are eliminated as they point towards the end-caps. This cut is used as a sanity check, that eliminates these particles that the detector was not able to measure geometrically. Also, the cut on the slope of particle track segments $(\Delta\phi/\Delta r)$ is required to be smaller than 6×10^{-4}. where, ϕ is the angle to the z-axis. Finally, z intercept (z_0) of all edges is required to be smaller than 100 mm to eliminate highly oblique edges. The distribution of fake and true edges with respect to these parameters is shown in Fig. 4.5, along with the choice of cuts. These cuts are shown with dashed lines. This step is important as it allows graph construction with fewer fake edges and reduces the computation time significantly.

In the end, 100 constructed graphs from 100 events are obtained with this method. The

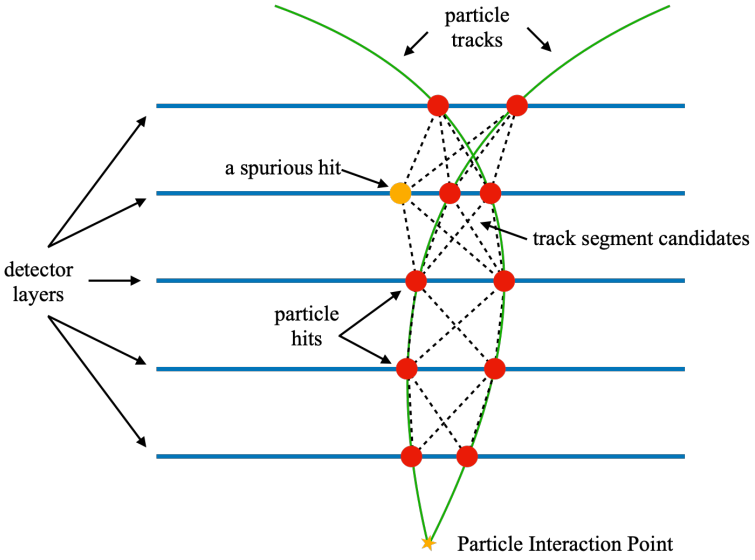


Figure 4.4: A sketch of particle track reconstruction. Blue horizontal lines represent detector layers. Orange star represents the particle interaction point. Green lines represent the trajectories of outgoing particles. Red circles represent detector measurements of the particles and orange circles represent spurious hits. Dashed black lines represent the potential track segments that connect measurements from two consecutive detector layers.

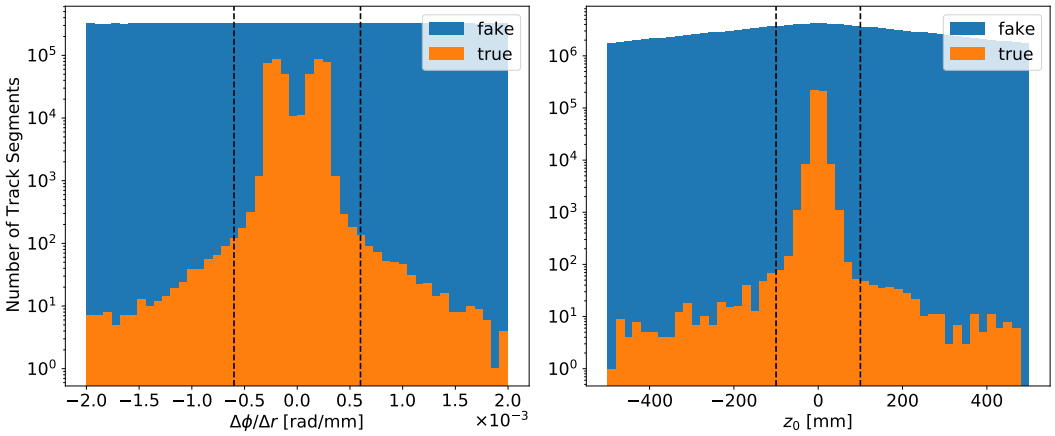


Figure 4.5: Histogram of fake and true segments in construction of graphs from 100 events.] All fake (blue) and true (orange) segments obtained during the procedure is plotted with respect to ratio of difference in ϕ to r ($\Delta\phi/\Delta r$) (on the left) and z intercept (z_0) (on the right).

graph production is performed with 0.99 efficiency and 0.51 purity, where efficiency and purity are defined in Eq. 4.2 and Eq. 4.3.

$$\text{Efficiency} = \frac{\text{Number of selected true track segments}}{\text{Number of initial track segments}} \tag{4.2}$$

$$\text{Purity} = \frac{\text{Number of selected true track segments}}{\text{Number of selected track segments}} \tag{4.3}$$

In return, this means 1% of the true track segments are lost and the QGNN model here will then work to increase the purity from 51% to as high as possible.

After the graph construction, the dataset is expressed by 4 matrices; $x \in \mathbb{R}^{N_V \times 3}$ stores 3 spatial coordinates of all nodes (in cylindrical coordinates; r, ϕ, z), $R_i \in \mathbb{R}^{N_V \times N_E}$ and $R_o \in \mathbb{R}^{N_V \times N_E}$ stores input and output nodes of all edges, and $y \in \mathbb{R}^{N_V}$ stores the labels of edges. They are defined as follows;

$$R_i^{jk} = \begin{cases} 1, & \text{if } k^{th} \text{ edge is input of } j^{th} \text{ node} \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

$$R_o^{jk} = \begin{cases} 1, & \text{if } k^{th} \text{ edge is output of } j^{th} \text{ node} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

$$y^k = \begin{cases} 1, & \text{if } k^{th} \text{ edge connects nodes that belong to same particle} \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Constructed graphs have 8784 $\pm$ 1877 (1σ) edges (N_E) and 5583 $\pm$ 804 (1σ) nodes (N_V) on average. An example graph, showing the fake and true edges in the transverse plane, is presented in Fig. 4.6. Most of the fake edges are in between the first two layers, since these can pass the $\Delta\phi/\Delta r$ and z_0 cuts due to being close to the origin. On the other hand, true edges are distributed more evenly throughout the layers. The total number of fake and true edges are similar as indicated by the purity of 0.51.

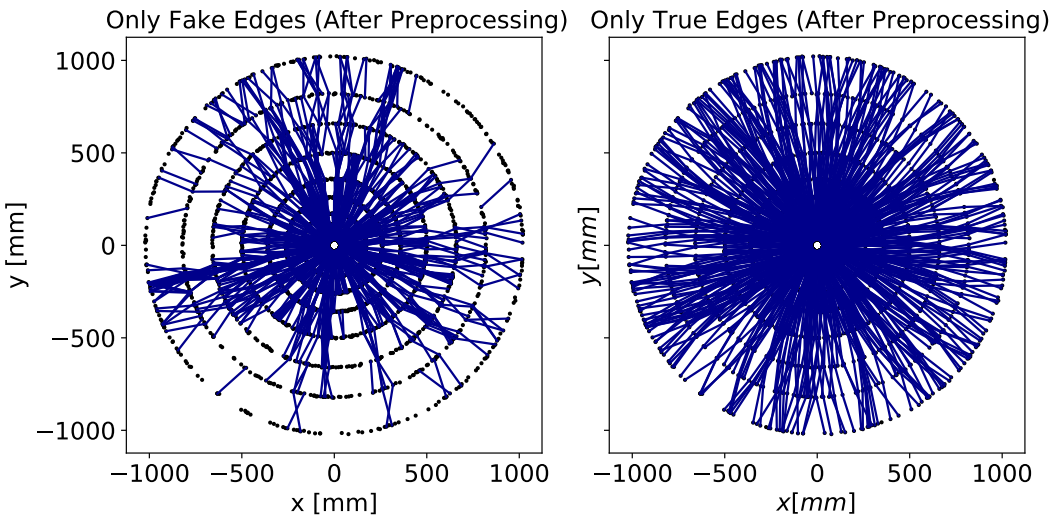


Figure 4.6: 2D projection of hits, fake and true edges of an event after pre-processing. All hits are plotted with black circles. Fake (on the left) and true (on the right) edges of a graph is plotted in the Cartesian coordinates (transverse plane). There are 5162 true and 5508 false edges plotted.

4.2 Hybrid Graph Neural Network

The Hybrid Quantum-Classical Graph Neural Network (QGNN) model takes a graph as an input and provides a scalar output for each edge, which is the likelihood of an edge being true. The model builds up on an Attention Passing Graph Neural Network model [126] following the same strategy as the HEP.TrkX project [5]. In contrast to the classical approach, the QGNN combines Multi Layer Perceptrons with QNN.

The QGNN consists of 3 parts, which are Input, Edge and Node Networks. The Input Network is responsible for increasing the dimension of the input data. It takes the spatial coordinate information (e.g. 3 cylindrical coordinates) and passes them through a single fully connected neural network with a sigmoid activation. The output size corresponds to the hidden dimension size (N_D). Then, these new data points are concatenated to form the initial node feature vector, where $v \in \mathbb{R}^{N_V \times (3+N_D)}$.

$$v = x \oplus \phi_{FC}(x) \tag{4.7}$$

Then, the node feature vector is fed into Edge and Node Networks, which process the graph iteratively $N_{iteration}$ times, in order to obtain a final edge probability value (e) for each of the edges. At the end of the model, the Edge Network is executed one more time to obtain final edge probabilities ($e \in \mathbb{R}^{N_V}$). This pipeline is summarized with a simple drawing in Fig. 4.7.

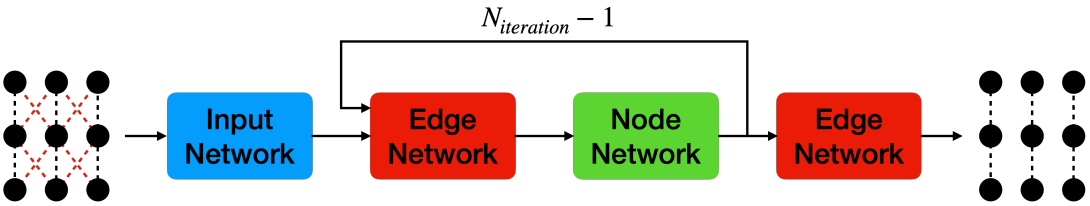


Figure 4.7: A schematic of the GNN architecture. The pre-processed graph is fed to an Input Network, which increases the dimension of the node features. Then, the graph's features are updated with the Edge and Node Networks iteratively, $N_{iteration}$ times. Finally, the Edge Network is used one more time to extract the edge features of the graph that predicts the track segments.

4.2.1 The Edge Network

The Edge Network takes pairs of nodes into account and returns the probability for the connection of the nodes. The initial connection between each pair of nodes is given by the connectivity matrices R_i and R_o, which were defined in Eq. 4.4 and Eq. 4.5. Node feature vectors of all initially connected edges are multiplied with R_i and R_o matrices to obtain respective input ($b_i \in \mathbb{R}^{N_E \times 3}$) and output ($b_o \in \mathbb{R}^{N_E \times 3}$) feature vectors of all edges as follows,

$$b_o^j = \sum_{k=1}^{N_V} R_o^{kj} v_k \qquad\qquad b_i^j = \sum_{k=1}^{N_V} R_i^{kj} v_k \tag{4.8}$$

The feature vectors of input and output nodes of each edge (b_i^j and b_o^j are the input and output feature vectors of the j^{th} edge) are concatenated to be fed into a Hybrid Neural Network ($\phi_{EdgeNetwork}$, HNN). The HNN returns edges features ($e \in \mathbb{R}^{N_E}$). This is the probability of how likely the edge is part of a real trajectory and then these

edge features are passed to the Node Network.

$$e_j = \phi_{EdgeNetwork}(b_o^j \oplus b_i^j) \tag{4.9}$$

4.2.2 The Node Network

The Node Network builds up on the edge feature matrix (e). Based on this input information, the node features are updated. A *triplet* is formed by combining each node of interest and its neighbors from upper and lower detectors. The node features of the neighbors' are weighted in this case with the corresponding edge features such that,

$$v'_{j,input} = \sum_{k=1}^{N_E} R_i^{jk} e_k b_o^j \qquad\qquad v'_{j,output} = \sum_{k=1}^{N_E} R_o^{jk} e_k b_i^j \tag{4.10}$$

Similar to the Edge Network, the triplet is fed to yet another HNN ($\phi_{NodeNetwork}$),

$$v_k \coloneqq \phi_{NodeNetwork}(v'_{k,input} \oplus v'_{k,output} \oplus v_k) \tag{4.11}$$

This time, the HNN returns new node features, v. The updated features are passed again to the Edge Network and this process is repeated for $N_{iteration}$ times. As we are interested in classifying hit segments, the final Network of the model is always an Edge Network. This behavior can be altered to create a model that can achieve different tasks, such as event classification or noisy hit detection.

4.2.3 The Hybrid Neural Network

The QGNN approach uses HNNs, which combine both classical and quantum layers. The HNN model takes the input and feeds it to a single FC NN layer. The output dimension of this layer is equal to number of qubits used by the quantum layer. Then, the output of the classical layer is used in the encoding step of the quantum layer.

Finally, the measurements of the quantum layer are fed to another single FC NN, which has the output dimension of 1 (in case of Edge Network) or hidden dimension size (in case of Node Network). This architecture, as presented in Fig. 4.8, allows full flexibility in choosing the number of hidden dimensions and the number of qubits as well as choosing the type of the quantum model.

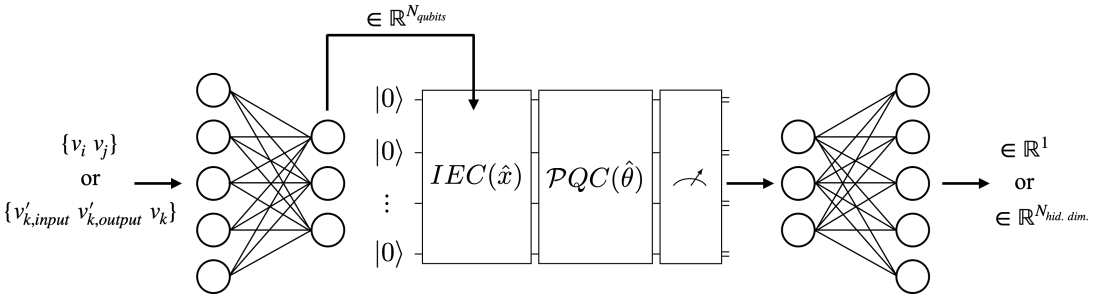


Figure 4.8: The HNN architecture design. The input is first fed through a classical FC NN layer. Then, its output is encoded in the Quantum circuit. The outputs are obtained as expectation values for each qubit from the circuit. A final FC NN layer is used to combine the results of different qubit measurements. The same HNN design is used in Edge (upper output and input dimension) and Node Networks (lower output and input dimension). The input and output dimension sizes change according to the network type.

The FC NNs of this model are standard classical layers, which have been widely used since decades. Therefore are kept fixed (except their dimension) and test the QGNN model with different quantum layers to understand potential benefits.

The QNN of choice consists of three consecutive parts. An Information Encoding Circuit (IEC) encodes classical data to the states of the qubits followed by a PQC that transforms these states to their optimal location in the Hilbert Space. Finally, measurements are performed along the z-axis with the σ_z operator.

As it was discussed in Section 3.3.1, the information encoding has a significant effect on the training capacity of QNN models. This requires a careful choice of encoding [93] and an angle encoding here was selected for two reasons. An angle encoding uses significantly less gates compared to other encoding methods, *e.g.* an amplitude encoding, and it needs almost no classical processing [88]. The QNN encodes the classical incoming information on the qubit states via rotational gates

in the desired axis. In order to obtain a unique and bijective representation of the classical data, the rotation angle is limited to $\theta \in [0, \pi]$ due to the periodicity of the cosine function. This is important since the expectation value is taken with respect to the σ_z-operator at the end of the circuit execution. The angle encoding with the R_Y gate is given as

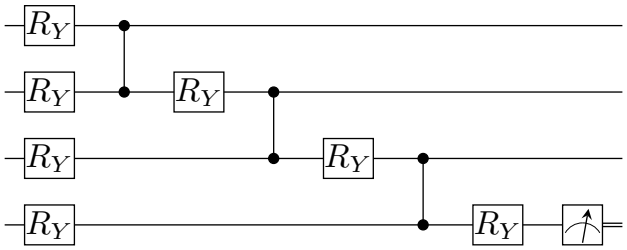
$$|\mathcal{D}\rangle = \bigotimes_{i=1}^{N} \{cos(x_i)\,|0\rangle + sin(x_i)\,|1\rangle\} \tag{4.12}$$

The PQC is the part of the QNN model that will be tuned in order to provide the desired output. As in classical NN layers, those initially randomly assigned variables are optimized during training to fit certain training objectives, *i.e.*, minimize the overall loss function. In order to achieve a good training performance, choosing a good combination of IEC and PQC is essential. Although there are several theoretical an experimental studies [101, 104, 127], the current understanding of which combination works best for which task is still limited [93]. Therefore, the IEC will be fixed to a specific angle encoding so that two selected types of PQCs can be examined and compared in this thesis.

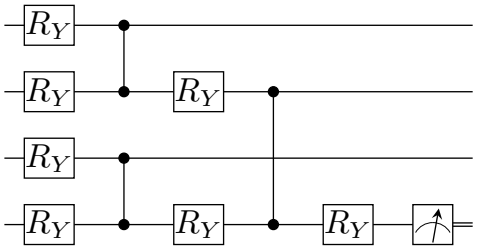
The first PQC type used here consists of circuits with a hierarchical architecture. MPS [96] and TTN [99] inspired circuits are selected from this group as shown in Fig. 4.9 (a ,b). However, these PQCs measure only one qubit. Thus, they are only implemented in case of the Edge Network as a multi-dimensional output is needed for the Node Network. The second type of PQCs (as shown in Fig. 4.9 (c, d)) are more common in the QML literature. They consist of repeated layers of parametrized gates that are followed by controlled operations. These circuits act on all qubits equally, meaning one can measure all qubits to obtain information. This makes them suitable for both Edge and Node Networks. As it was discussed in Section 3.3.2, there are descriptors for the second type of PQCs (Circuit 10 and 19)[1].

Expressibility and Entangling Capacity provides information about the type of circuits before training [101]. Circuit 10 and Circuit 19 are selected for comparisons. Circuit 19 has more Expressibility, whereas circuit 10 has a larger Entangling Capacity. As it

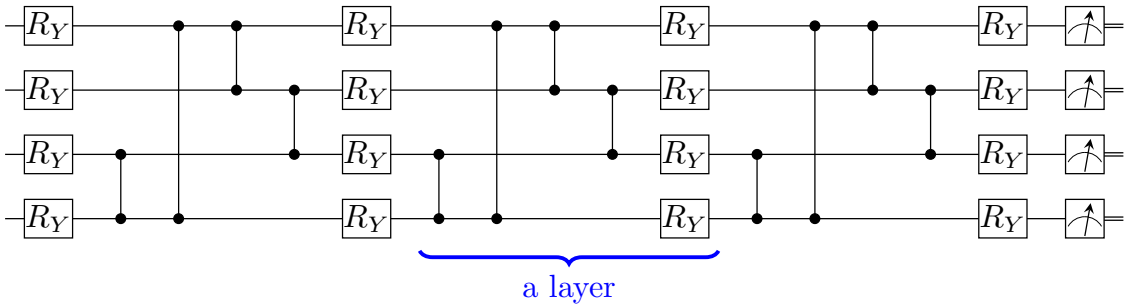
[1] Circuit 10 and Circuit 19 are initially defined in [101]. For this reason we don't change their name and use the same convention.



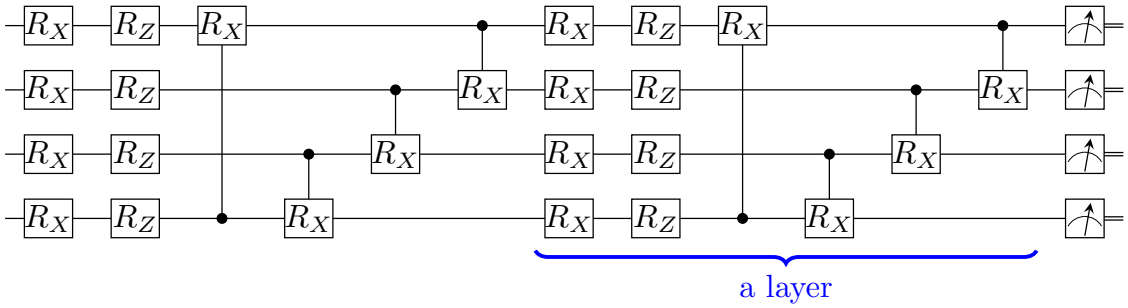
(a) Matrix Product State (MPS) inspired 4 qubit circuit. Circuit is adapted from [96].



(b) Tree Tensor Network (TTN) inspired 4 qubit circuit. Circuit is adapted from [99].



(c) Circuit 10 in 3 layer and 4 qubit configuration. Adapted from [101].



(d) Circuit 19 in 2 layer and 4 qubit configuration. Adapted from [101].

Figure 4.9: The 4 qubit configurations of the PQCs used in this work.

can be seen in Fig. 4.10, Circuit 19 always has more Expressibility and Entangling Capacity, and can yield similar values to the Haar Random case with enough number of layers.

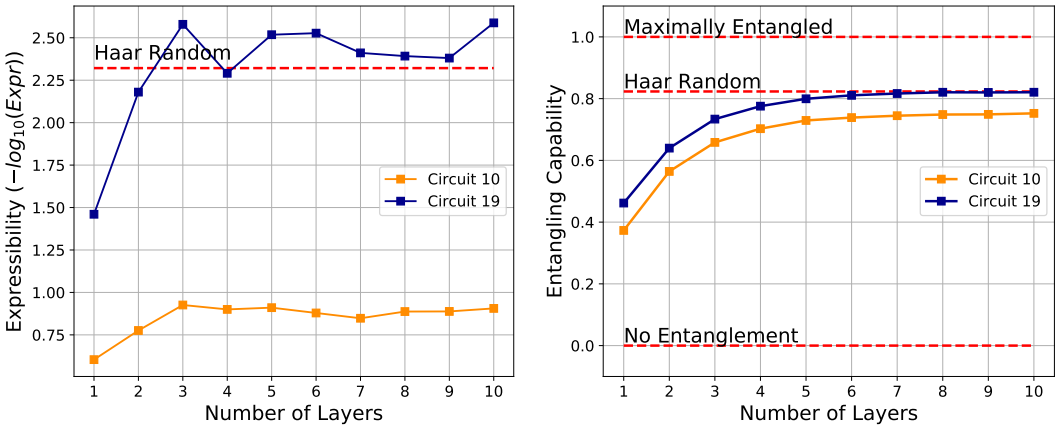


Figure 4.10: Expressibility and Entanglement Capacity vs. number of layers for some of the PQCs in their 4 qubit configurations. Expressibility (on the left) and Entanglement Capacity (on the right) of Circuit 10 (orange) and 19 (blue) are plotted with various number of layers.

4.3 Training the Model

Training hybrid quantum-classical NNs requires software that is able to differentiate gradients for both classical and quantum NNs. Pennylane [81] is one of the most popular open-source tools that provides this. Pennylane has been used along with Pytorch [109] during the early stages of this work to train the model. However, this combination turned out to be too slow to train the proposed QGNN model with the TrackML dataset. A computational speed-up in training was achieved using Qulacs [128]. Finally, the combination of Cirq [82], Tensorflow [108] and Tensorflow Quantum [129] produced the optimal scenario, in which we were able to reduce the duration of training to less than a week. The quantum circuit simulations are performed by only taking analytical results into account, which analytical results do not reflect hardware limitations. However, this choice was made in order to obtain results in a reasonable amount of time.

The 100 events selected from the dataset are separated randomly with a 50/50 ratio to be used as training and validation sets as it was mentioned in pre-processing. A weighted binary cross entropy loss function is used to account for the uneven distribution of true/fake segments. The Adam optimizer [107] with a learning rate of 0.01 is used to train all trainable parameters of the hybrid model. The learning is done with a batch size of 1 per graph and continued up-to 10 or 20 epochs depending on model complexity. Batch size is the number of events that the optimizer uses to update the parameters and an epoch corresponds to a full pass of the dataset. All models are trained with a different initialization for 5 times.

4.4 Performance Metrics

There are many metrics used to quantify goodness of a classification besides the loss function [117]. Although the optimizer only uses the loss function to train the model, it is important to observe the models' performance with respect to common metrics. Some of these metrics require a threshold between 0 and 1 to make a prediction, such that;

$$\text{prediction} = \begin{cases} \text{True}, & \text{output} > \text{threshold} \\ \text{False}, & \text{output} \leq \text{threshold} \end{cases} \tag{4.13}$$

A prediction can belong to 4 possible groups in the binary classification task. These groups and their names are given with the confusion matrix;

Table 4.2: Confusion matrix for binary classification.

Actual \ Prediction	True	False
True	True Positive (TP)	False Negative (FN)
False	False Positive (FP)	True Negative (TN)

Then, the metrics depending on the confusion matrix (accuracy, precision, recall and F1 score) require a threshold. These will be discussed now. Accuracy is the ratio of

true classifications and defined as,

$$Accuracy = \frac{TP + TN}{TN + FP + FN + TP} \tag{4.14}$$

A true random model produces an accuracy of 0.5, while a perfect model has an accuracy of 1.0. Precision is the ratio of correct predictions of true edges to all true predictions and defined as,

$$Precision = \frac{TP}{TP + FP} \tag{4.15}$$

Precision measures a model's ability not to classify fake edges as true edges. Recall is the ratio of correct predictions of true edges to all true edges and defined as,

$$Recall = \frac{TP}{TP + FN} \tag{4.16}$$

Recall measures the model's ability to find all true edges. There is usually a trade-off between precision and recall. A model with high precision low recall would be good at selecting the true predictions with the cost of choosing only few. A model with low precision and high recall would identify more samples as true but most of them would be wrong. A good model should have high precision and high recall at the same time [117]. Another possibility to quantify a model's performance is the F1 Score, which is a more inclusive metric. The F1 Score takes the harmonic mean of precision and recall and is defined as

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{4.17}$$

In the best case, the F1 score takes the value 1.0, and 0.0 in the worst case. There are also metrics that do not require a specific threshold. One of these metrics is the Area Under the Receiver Operating Characteristic Curve (ROC AUC or AUC) [130] as shown in Fig. 4.11. As the name suggests, AUC is computed by calculating the area under a Receiver Operating Characteristic (ROC) curve. The ROC curve is computed

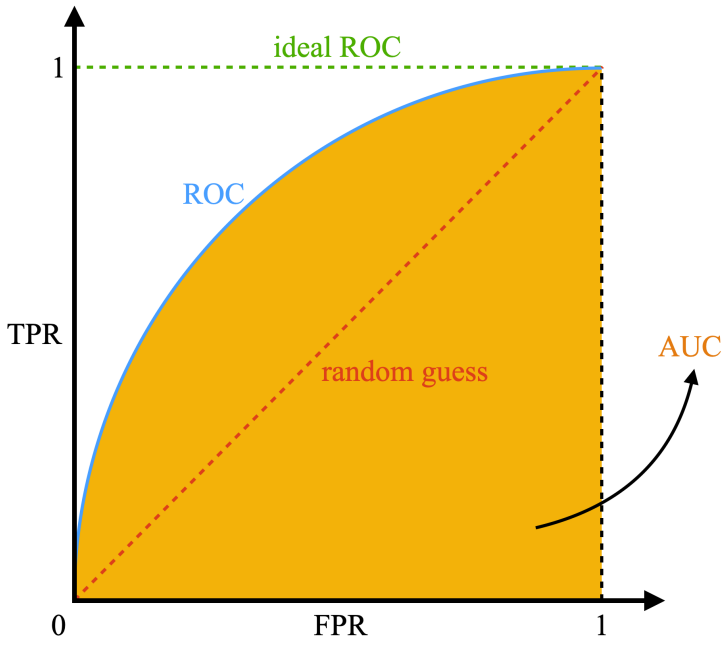


Figure 4.11: ROC plot of an arbitrary model. The blue curve represents the ROC of the model. The area under the blue line, highlighted in orange, represents the AUC. The dashed red line shows the true random prediction and the dashed green line shows the ideal prediction case.

by evaluating the model's True Positive Rate (TPR) and False Positive Rate (FPR) at various thresholds from 0.0 to 1.0. TPR and FPR is defined as,

$$TPR = \frac{TP}{TP + FN} \tag{4.18}$$

$$FPR = \frac{FP}{FP + TN} \tag{4.19}$$

ROC has the FPR on the x-axis, while the TPR is on the y-axis. Evaluating a model's performance at various threshold makes AUC a very inclusive metric. This way a model's performance can be quantified without a choice made by a user, while a choice of threshold is required for those models that require it.

4.5 Hardware and Software Information

All results presented in this work used the following hardware and software specifications.

A Dell rack computer with two Intel Xeon Silver 4216 CPU, 4x64 GB 2933MHz RAM, 2x1 TB SSD storage and two 2080 Ti Nvidia GPU. The open-source software used for this work can be listed as follows. Python 3.8.5, NumPy v1.18.5 [131], Cirq v0.9.1 [82], Tensorflow v2.3.1 [108], Tensorflow Quantum v0.4.0 [129], Scikit-learn v0.23.2 [110], qpic v1.0.2 [132], Matplotlib v3.2.2 [133]. The project website and codebase to reproduce all of the results presented here can be accessed through `https://qtrkx.github.io` and `https://github.com/QTrkX/qtrkx-gnn-tracking`.

CHAPTER 5

RESULTS & DISCUSSION

5.1 Results

In this section, training results of the proposed QGNN model with various settings are presented. Trials are presented with 4 key comparisons of features that have a significant effect on the performance of the model. The learning is performed with a batch size of 1 per graph and continued up-to 10 or 20 epochs depending on model complexity. All models are trained for 5 independent initializations and their mean is presented. The shaded regions, in each graph, represent the $\pm$ standard deviation of the best losses of all 5 runs. A 0.5 threshold is used for metrics that require a threshold. Please refer to Section 4.3 for more details.

5.1.1 Embedding Axis Comparison

The embedding axis comparison considers angle embeddings over different axes of the Bloch Sphere. For this comparison Circuit 10 and Circuit 19 are chosen as they exhibit rotation gates with different axes.

A setting of 4 qubits is chosen for this comparison including a hidden dimension size of 4 and a single layer PQC as well as an iteration number of 3. All models are trained for 20 epochs and their best losses are plotted in Fig. 5.1. In both cases, the y-axis embedding resulted in better loss values with a marginal difference to the x-axis embedding. The z-axis embeddings performed the worst with both PQCs. Due to this outcome, the y-axis angle embedding is considered for the rest of the trials.

Training curves of this comparison, with respect to different metrics, are presented in

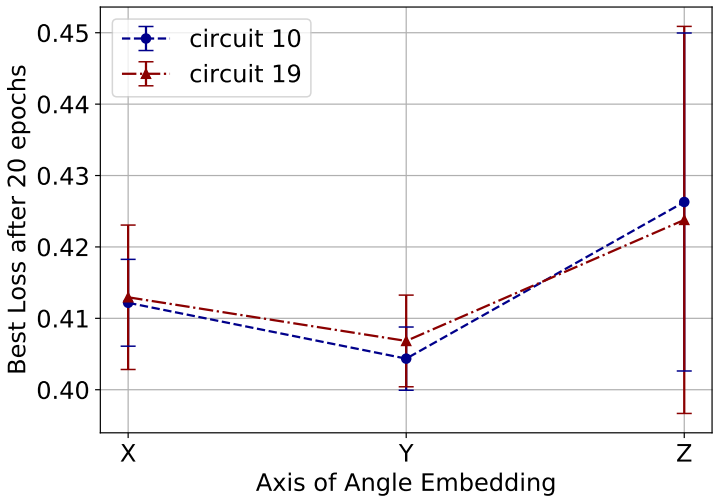


Figure 5.1: Axis of angle embedding comparison. The x-axis determines the axis of the embedding, while the y axis shows the best loss values after 20 epochs of training. All models are trained 5 times and their means are plotted in blue (circuit 10) and red (circuit 19), respectively. The error bars represent the $\pm$ standard deviation of the best losses of all 5 runs.

Fig. 5.2 in the same scales for ease of comparison and each row represents a different axis of embedding. Circuit 10 slightly outperforms circuit 19 in all cases. The learning curves are very similar in the case of x and y axis embeddings but standard deviations of Circuit 10 are tighter for loss, accuracy and AUC. In contrast to these, the z axis embedding performs poorly and has a considerably larger standard deviation for both PQCs. After 20 epochs, all models start to converge, meaning that the optimizer is close to find a local or global minima. Different final values show different learning capacities of models.

5.1.2 Number of Layers Comparison

Next, the number of layers are compared for models with a different amount of PQC layers. For this purpose, two models with circuit 10 and circuit 19 with a hidden dimension size of 4 and an iteration number of 3 are selected. The best losses obtained after 20 epochs of training is plotted with respect to number of layers in Fig. 5.3. In both cases, the model's performance did not improve with an increasing number

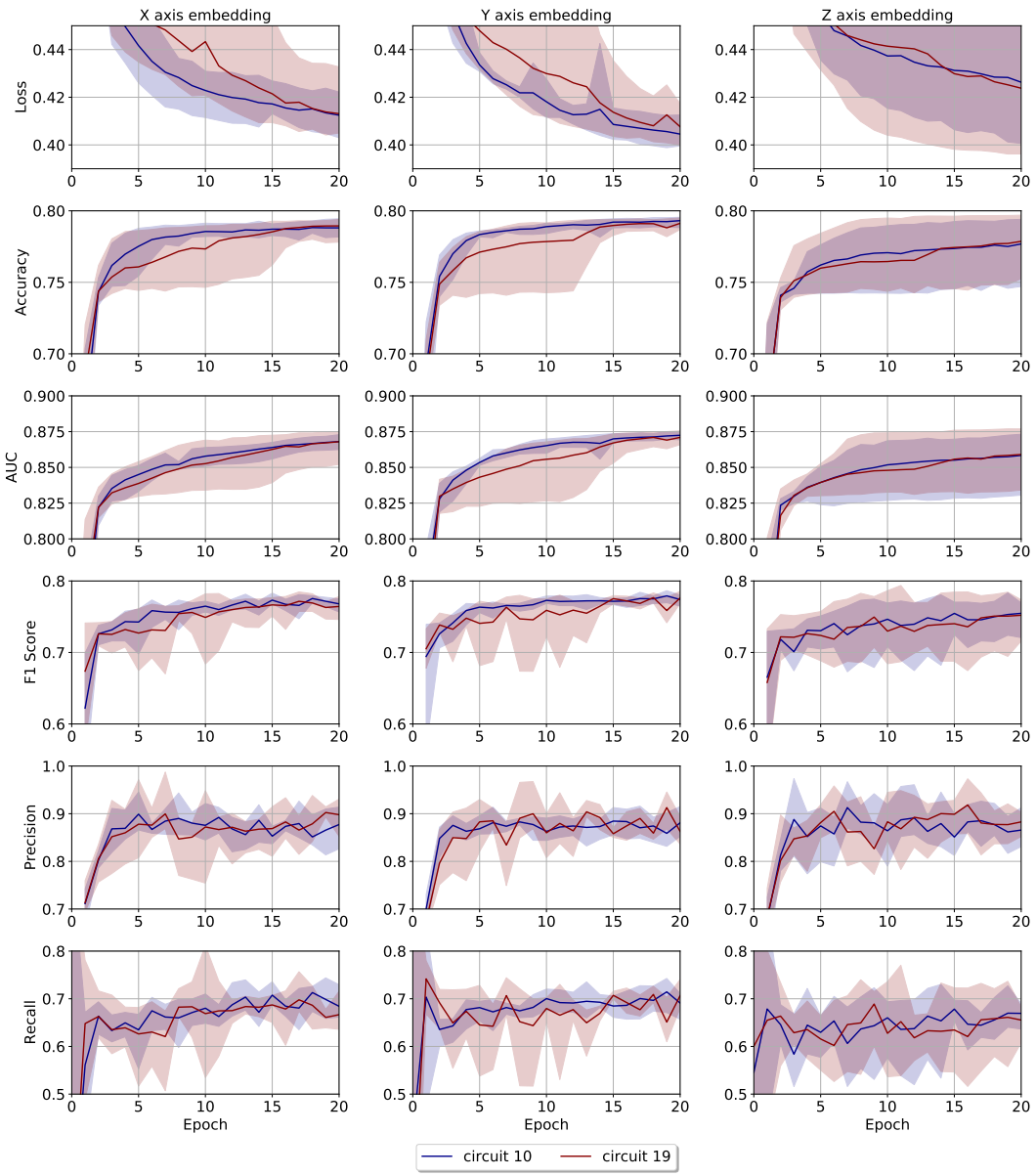


Figure 5.2: Embedding axis comparison training curves. Three columns are X, Y and Z axis embeddings respectively and each row presents results for a different performance metric. All models are trained 5 times and their means are plotted in blue (circuit 10) and red (circuit 19), respectively. The shaded regions represent the $\pm$ standard deviation of the best losses of all 5 runs.

of layers. But, it is also hard to say it did not get worse due to low statistics. The increasing size of the error bars indicate that both models become harder to train, the more layers they contain.

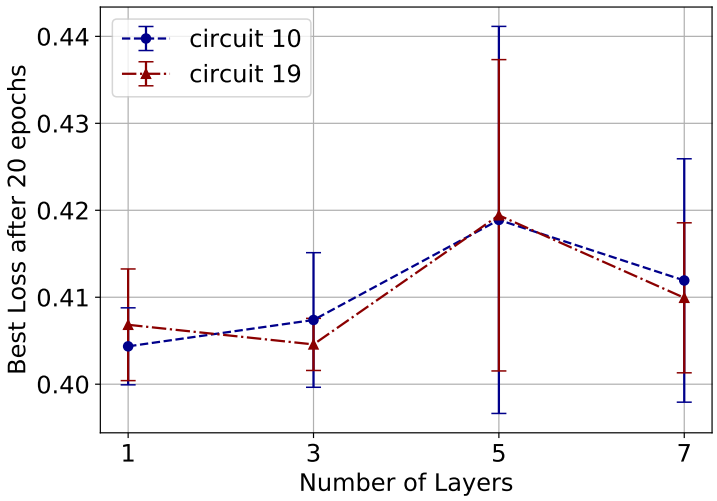


Figure 5.3: Number of layers comparison. The x-axis determines the number of PQC layers, while the y axis shows the best loss values after 20 epochs of training. All models are trained 5 times and their means are plotted in blue (circuit 10) and red (circuit 19), respectively. The error bars represent the $\pm$ standard deviation of the best losses of all 5 runs.

Training curves of this comparison, with respect to different metrics, are presented in Fig. 5.4. Circuit 10 slightly outperforms circuit 19 with models that have less layers. But this situation seems to change with an increasing number of layers. The standard deviation of the training curves also gets significantly larger with an increasing number of layers. The models with more layers reach a plateau later for loss, accuracy and AUC.

5.1.3 Number of Iterations Comparison

The number of iterations are compared looks to see the effect of graph iterations on model performance. A comparison of 1, 3, 5 and 7 iterations with a hidden dimension size of 4, with 4 qubits and single layer PQC hybrid model is performed with Circuit 10 and Circuit 19 and the results are presented in Fig. 5.5. The training results show that the best loss is obtained for 3 iterations for the hybrid cases. However, this is not true in the classical case, as the classical model continues to improve with more iterations.

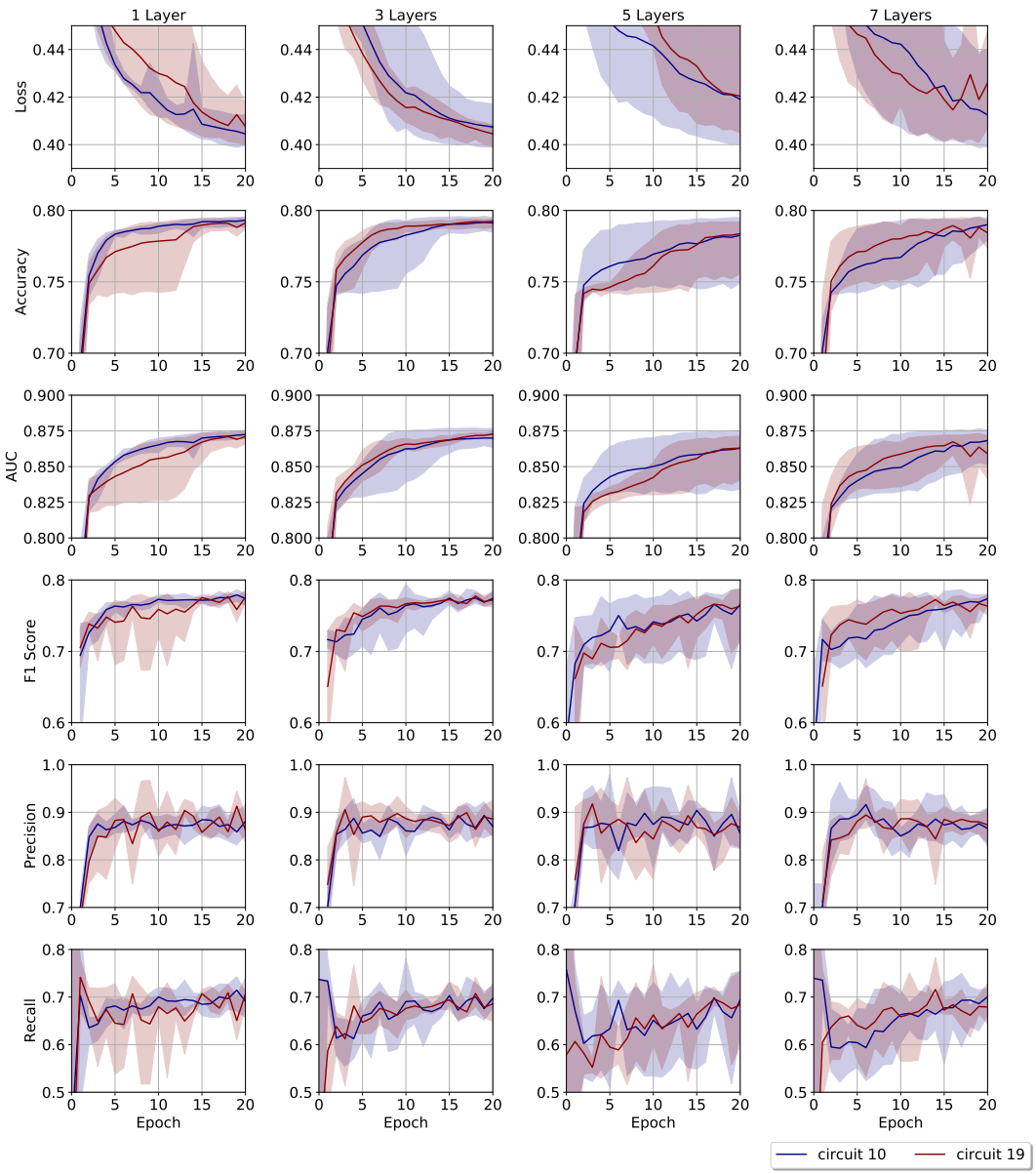


Figure 5.4: Comparison of training curves for different (1, 3, 5 and 7) layers. Each column represents models with a different number of layers, while each row presents results for a different performance metric. All models are trained 5 times and their means are plotted with blue (circuit 10) and red (circuit 19) lines. The shaded regions represent the $\pm$ standard deviation of the best losses of all 5 runs.

Training curves of this comparison with respect to different metrics are presented in Fig. 5.6 in the same scale for ease of comparison. Results are presented for 1, 3, 5 and 7 layers in different columns. In the hybrid cases, the model gets harder to train

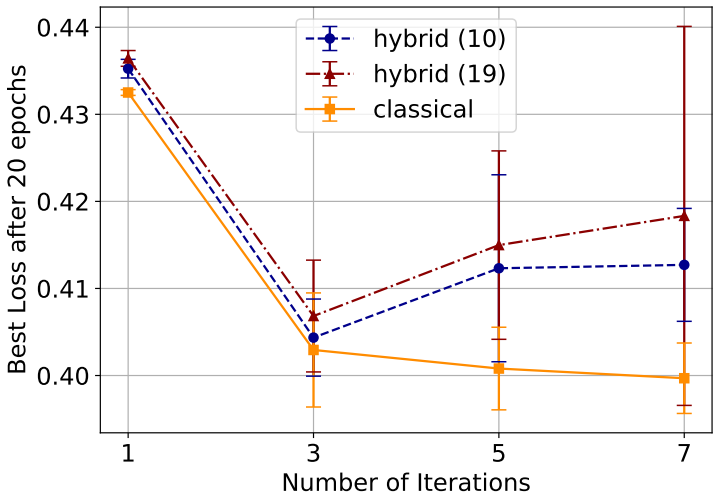


Figure 5.5: Comparison of the number of iterations. The x-axis determines the number of graph iterations, while the y axis shows the best losses after 20 epochs of training. All models are trained 5 times and their means are plotted in blue (circuit 10), red (circuit 19) and orange (HEP.TrkX [5]) respectively. The error bars represent the $\pm$ standard deviation of the best losses of all 5 runs.

with the increasing number of iterations. Both the performance with respect to almost all metrics gets worse and the standard deviations grow after number of an iteration number of 3. In contrast, the classical model continues to improve marginally with more iterations.

5.1.4 Hidden Dimension Size Comparison

Performance of models with different hidden dimension sizes are compared to investigate how they scale. This comparison is made with the choice of $N_{qubits} = N_{hid.dim.}$, 3 iterations and using only a single layer of the PQCs. Two different configurations of PQCs are compared. Models with the labels `circuit 10` and `circuit 19` use the same circuits with different initial parameters for the Edge and Node Networks, as it was done in previous comparisons. While the models with `TTN-10` and `MPS-10` labels use `circuit 10` for the Node Network, a TTN or an MPS type of PQC is used in case for the Edge Network. TTN and MPS circuits are not used in the Node Network, because the Node Network needs an output dimension size that is equal to

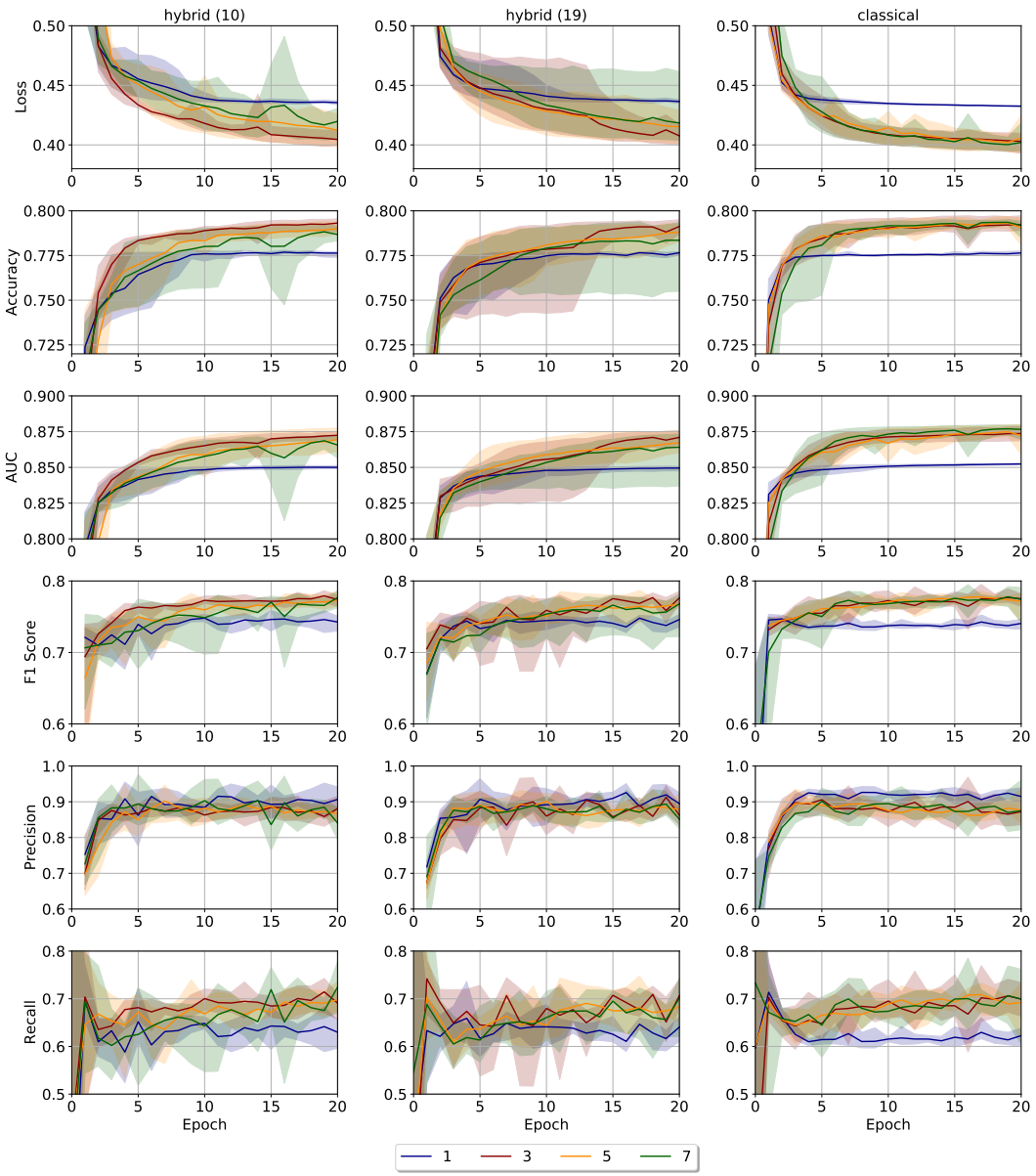


Figure 5.6: Comparison of the training curves of number of iterations. The first two columns represent hybrid models with Circuit 10 and Circuit 19, and the last column is for the classical HEP.TrkX model [5]. Each row presents results for a different performance metric. All models are trained 5 times and their means are plotted with different colored lines for each number of iteration. The shaded regions represent the $\pm$ standard deviation of the best losses of all 5 runs.

the hidden dimension size. However, TTN and MPS circuits only measure a single qubit. The definitions of the models used in this comparison are also given in Table 5.1

for further clarification.

Table 5.1: Model labels and their respective PQCs used in different parts of the network.

Label	PQC of Edge Network	PQC of Node Network
`MPS-10`	MPS	Circuit 10
`TTN-10`	TTN	Circuit 10

The results show in all settings that the model performance improves with increasing hidden dimension size. The model with `circuit 10` outperforms the rest consistently. However, there seems to be a saturation of the best loss as the hidden dimension size increases.

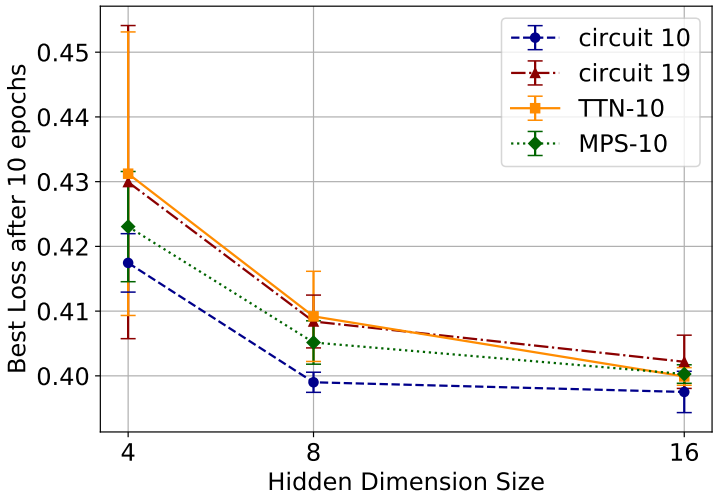


Figure 5.7: Comparison of hidden dimension size. The x-axis determines the hidden dimension size, while the y axis shows the best losses after 10 epochs of training. All models are trained 5 times and their means are plotted in blue (`circuit 10`), red (`circuit 19`), orange (`TTN-10`) and green (`MPS-10`) respectively. The error bars represent the $\pm$ standard deviation of the best losses of all 5 runs.

Training curves of this comparison with respect to different metrics are presented in Fig. 5.8. `circuit 10` slightly outperforms other models consistently for all hidden dimension sizes. All models improve with increasing the hidden dimension size and

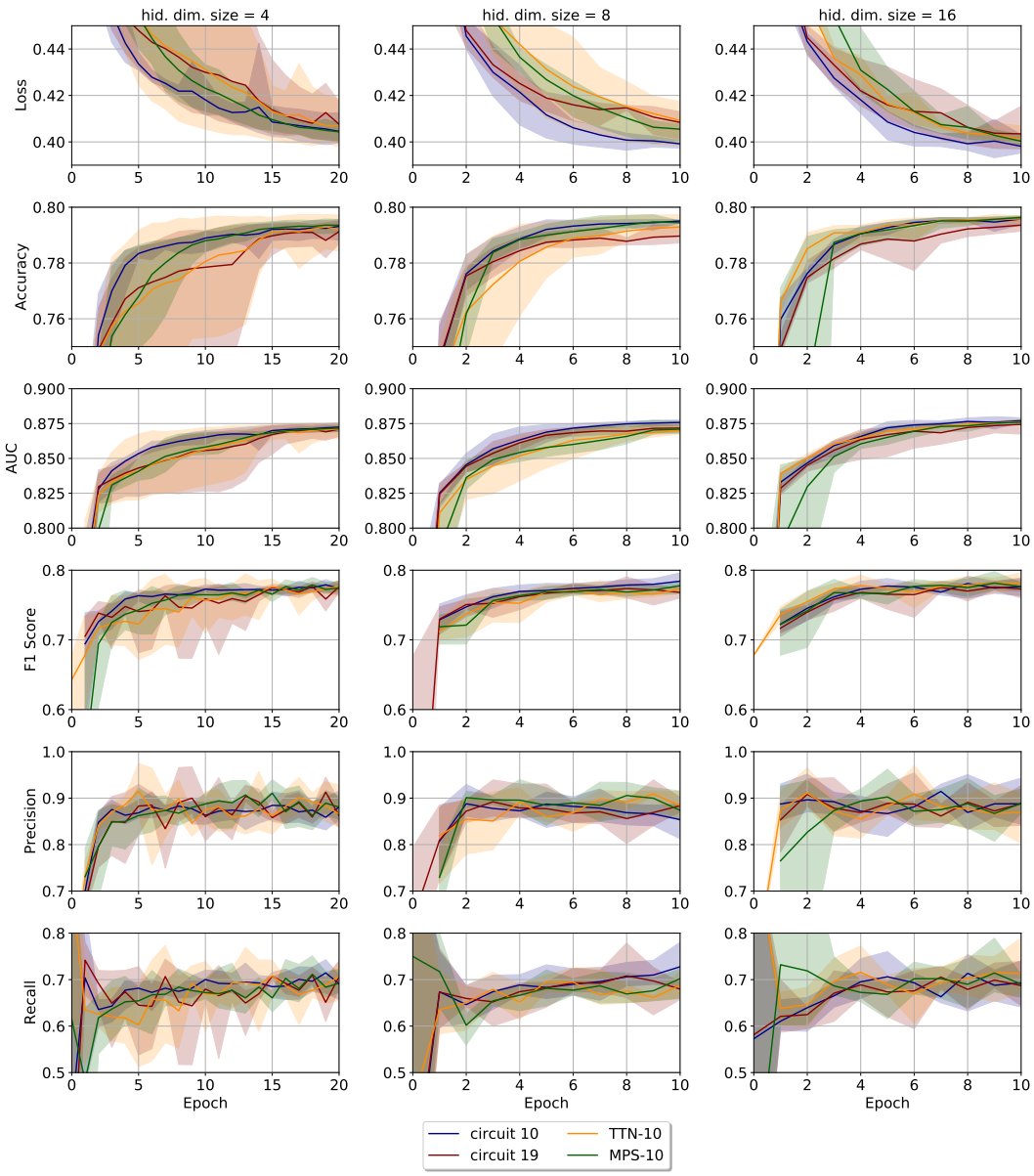


Figure 5.8: Comparison of training curves for hidden dimension sizes. Each column represents models with different hidden dimension sizes (4, 8, 16), while each row presents results for a different performance metric. All models are trained 5 times and their means are plotted with blue (`circuit 10`), red (`circuit 19`), orange (`TTN-10`) and green (`MPS-10`) lines. The shaded regions represent the $\pm$ standard deviation of the best losses of all 5 runs.

also do not seem to show large standard deviations indicating a good scaling.

Finally, the hybrid model is compared against the classical model at different hidden

dimension sizes and presented in Fig. 5.9. For this comparison, 3 iterations and the same choice of $N_{qubits} = N_{hid.dim.}$ is employed. This result shows that the hybrid model scales and behaves similarly to the classical model until a hidden dimension size of at least 16.

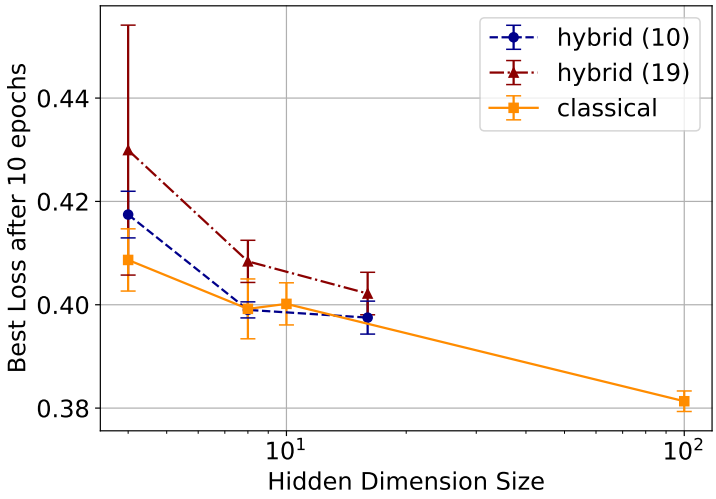


Figure 5.9: Comparison of hybrid and classical models for different hidden dimension sizes. All models are trained 5 times and their means are plotted in blue (Circuit 10), red (Circuit 19), orange (HEP.TrkX [5]) respectively. The x-axis determines the hidden dimension size, while the y axis shows the best loss values after a training of 10 epochs. The error bars represent the $\pm$ standard deviation of the best losses of all 5 runs.

5.2 Discussion

There are several factors that determine performance of hybrid quantum classical models. This work presents a systematic study of these factors and analyze their affect on selected QGNN models.

The embedding of classical information is one of the key components that determine the effectiveness of hybrid models. It was shown in this work that, this can be achieved with a simple angle encoding. Results indicate that PQCs that consist of R_Y parametrized gates benefit from embeddings on x or y axes but z-axis embedding performs poorly. The main reason of this outcome is the fact that the measurements are

also along this axis (σ_Z operator was used to perform all measurements). Therefore, the extraction of information depends heavily on the choice of PQC in this case. Fig 5.1 shows the results for single PQC models, which do not have enough depth to do achieve a transformation on the Bloch Sphere such that any embedding axis can benefit. Thus, both models performed poorly under z-axis embedding.

Recently it was shown that the accuracy of a model significantly improves with a better embedding [134]. For example, the Feature Map embedding requires polynomial depth and it is hard to simulate and Tensorflow Quantum is not able to differentiate them as of version 0.4.0 [129]. For this reason, only results obtained with angle embedding were presented here. In general, the accuracy of the proposed QGNN model should improve with a better embedding.

After that, a comparison to understand the effect of more layers on the models' performance were presented. As it was previously discussed, adding more layers to the model improves expressibility and entanglement capacity [101] at the cost of introducing Barren Plateaus [120], which reduce trainability of a model. There are contrasting results in the literature on how expressibility and entanglement capacities affect the training performance. Recently, both a positive [104] and an anti correlation [127] between expressibility and accuracy was shown. The second also found that more expressive models perform worse and also overfit more. On the other hand, entanglement has been shown to limit the trainability of models depending on how it propagates between qubits [123, 135].

Fig 5.3 shows that models with increasing expressivity and entanglement capacity perform worse. In this way, our results are consistent with results from [127]. This behaviour is thought to be the result of a Barren Plateau formation [120].

The next feature of the QGNN model analyzed here was the number of graph iterations. It is an important parameter that determines a GNN model's performance as it allows for the propagation of information to further nodes [5, 126]. Training results in Fig. 5.5 show that the best loss is obtained for 3 iterations for the hybrid cases. However, this is not the case for the classical model. Hep.TrkX team reports an iteration number of 8 as the optimal value for their model with 128 hidden dimensions [5, 136]. The increase in best loss with increasing number of iterations might be due to a low expressive

capacity of the whole model, as this comparison is made only for a hidden dimension size of 4. Therefore, an analysis with more hidden dimension size is necessary to understand the observed behaviour. However, this could not be performed due to the long simulation times required by larger models.

Next, a comparison of models' performance with respect to increasing hidden dimension size was given in Fig 5.7. In this comparison, two types of PQCs were targeted. Circuit 10 and Circuit 19 are PQCs that measure all qubits, while TTN and MPS are PQCs that measure only one qubit. The important difference to note here is the TTN-like circuits have been shown not to exhibit Barren Plateaus. Results showed that in all cases, the training capacity of the model improved. However, no improvement was observed when the Edge Network of the model used TTN or MPS PQCs instead of Circuit 10, which should have shown Barren Plateau formation. In both cases, a similar scaling is observed. This result might be due to presence of encoder and decoder classical Neural Networks in the HNN architecture, which could dominate the model for smaller hidden dimension sizes.

Finally, the selected QGNN models' performance is compared with the HEP.TrkX model [5]. The results showed that both models perform and scale similarly at low hidden dimension sizes. Although we were not able to show an improvement over the classical model, an improvement might be achieved with further modifications to the model. These include but are not limited to using more sophisticated embeddings such as the Feature Map [134], using PQCs that do not exhibit Barren Plateaus [120, 135], or methods to avoid them during training [137].

CHAPTER 6

CONCLUSION

HEP experiments are constantly upgrading with new technologies to obtain higher statistics and better measurements of HEP processes. The rapidly developing hardware requires the software to improve with it, to provide optimal results. This is very much the case in the upcoming HL-LHC and will continue to be the case for future HEP experiments.

Recent developments show that GNNs have a huge potential to play a considerable role overcoming the track reconstruction problem [5, 136]. However, their power is limited to the size of the GPU memories, and therefore need additional methods. Although future developments in GPU or maybe Tensor Processing Unit (TPU) technologies might resolve this problem, it would be beneficial to search for new possibilities to use GNN.

The flexibility of VQAs provides a ground for ML researchers to implement algorithms that can achieve similar tasks to the ones that exist in classical DL paradigms. Although this field of research is developing very fast and has become very popular in recent years, there is still no clear evidence for a VQA model to show an advantage in a classical problem [138]. However, this should not restrain us from continuing the search.

In this work, we have implemented a QGNN to solve the particle track reconstruction problem. The proposed method showed similar performance compared to its classical counterpart within the number of qubits limit of NISQ devices.

Furthermore, we have investigated the proposed model's scaling abilities and trainability limits and obtained parallel results to recent developments in the literature. Most

importantly, we have observed that neither an increase in expressibility nor an increase in entangling capacity do not improve a hybrid model's performance in the low qubit limits. These results cover important properties of HNNs and would be insightful for future model designs.

The current state of NISQ hardware and the classical simulation restrictions prevented us from providing more extended results. However, we predict that the current pace of research in the field will allow these algorithms to be implemented on NISQ hardware and provide more insight to solve the Barren Plateau problem very soon.

REFERENCES

[1] L. Evans and P. Bryant, "LHC Machine", *Journal of Instrumentation*, vol. 3, no. 08, S08001, Aug. 2008. DOI: 10.1088/1748-0221/3/08/s08001.

[2] D. Magano, A. Kumar, M. Kālis, A. Locāns, A. Glos, *et al.*, "Investigating Quantum Speedup for Track Reconstruction: Classical and Quantum Computational Complexity Analysis", 2021. arXiv: 2104.11583 [quant-ph].

[3] G. Apollinari, O. Brüning, T. Nakamoto, and L. Rossi, "Chapter 1: High Luminosity Large Hadron Collider HL-LHC", *CERN Yellow Report*, May 2017. DOI: 10.5170/CERN-2015-005.1. arXiv: 1705.08830 [physics.acc-ph].

[4] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, *et al.*, "A Comprehensive Survey on Graph Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021, Publisher: IEEE, ISSN: 21622388. DOI: 10.1109/TNNLS.2020.2978386.

[5] S. Farrell, P. Calafiura, M. Mudigonda, Prabhat, D. Anderson, *et al.*, *Novel deep learning methods for track reconstruction*, 2018. arXiv: 1810.06111 [hep-ex].

[6] D. Murnane, "CTD2020: Graph Neural Networks for Track Finding", Zenodo, Apr. 2020. DOI: 10.5281/zenodo.4088460.

[7] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics", *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 021 001, Jan. 2021. DOI: 10.1088/2632-2153/abbf9a.

[8] F. Bapst, W. Bhimji, P. Calafiura, H. Gray, W. Lavrijsen, *et al.*, "A Pattern Recognition Algorithm for Quantum Annealers", *Computing and Software for Big Science*, vol. 4, no. 1, p. 1, Dec. 2019, ISSN: 2510-2044. DOI: 10.1007/s41781-019-0032-5.

[9] A. Zlokapa, A. Anand, J.-R. Vlimant, J. M. Duarte, J. Job, *et al.* (2019). "Charged particle tracking with quantum annealing-inspired optimization". arXiv: 1908.04475 [quant-ph].

[10] G. Quiroz, L. Ice, A. Delgado, and T. S. Humble, "Particle Track Classification Using Quantum Associative Memory", 2020. arXiv: 2011.11848 [quant-ph].

[11] S. Amrouche, L. Basara, P. Calafiura, V. Estrade, S. Farrell, *et al.*, "The Tracking Machine Learning Challenge: Accuracy Phase", in *The NeurIPS 2018 Competition*, Springer International Publishing, Nov. 2019, pp. 231–264. DOI: 10.1007/978-3-030-29135-8_9. arXiv: 1904.06778 [hep-ex].

[12] S. Amrouche, L. Basara, P. Calafiura, D. Emeliyanov, V. Estrade, *et al.* (2021). "The Tracking Machine Learning challenge : Throughput phase". arXiv: 2105.01160 [cs.LG].

[13] *CERN Accelerator Complex*, https://stfc.ukri.org/research/particle-physics-and-particle-astrophysics/large-hadron-collider/cern-accelerator-complex/, Accessed: 2021-02-01.

[14] *Taking a closer look at LHC*, https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.lhc_p_collisions, Accessed: 2021-02-16.

[15] *The CMS Detector*, https://cms.cern/detector, Accessed: 2021-02-01.

[16] ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", *Physics Letters B*, vol. 716, no. 1, pp. 1–29, 2012, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020.

[17] CMS Collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC", *Physics Letters B*, vol. 716, no. 1, pp. 30–61, 2012, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.021.

[18] "Evidence for Higgs boson decays to a low-mass dilepton system and a photon in pp collisions at $\sqrt{s}$ = 13 TeV with the ATLAS detector", CERN, Geneva, Tech. Rep. ATLAS-CONF-2021-002, Feb. 2021.

[19] *The Inner Detector*, https://atlas.cern/discover/detector/inner-detector, Accessed: 2021-02-01.

[20] F. Ragusa and L. Rolandi, "Tracking at LHC", *New Journal of Physics*, vol. 9, no. 9, pp. 336–336, Sep. 2007. DOI: 10.1088/1367-2630/9/9/336.

[21] CMS Collaboration, "Description and performance of track and primary-vertex reconstruction with the CMS tracker", *Journal of Instrumentation*, vol. 9, no. 10, P10009–P10009, Oct. 2014. DOI: 10.1088/1748-0221/9/10/p10009.

[22] R. Frühwirth, "Application of kalman filtering to track and vertex fitting", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 262, no. 2, pp. 444–450, 1987, ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(87)90887-4.

[23] S. Farrell, D. Anderson, P. Calafiura, G. Cerati, L. Gray, *et al.*, "The HEP.TrkX Project: deep neural networks for HL-LHC online and offline tracking", *EPJ Web Conf.*, vol. 150, p. 00 003, 2017. DOI: 10.1051/epjconf/201715000003.

[24] G. Cerati, "Vertexing and Tracking Algoritms at High Pile-Up", *Proceedings of The 23rd International Workshop on Vertex Detectors — PoS(Vertex2014)*, vol. 227, p. 037, 2015. DOI: 10.22323/1.227.0037.

[25] "Expected Tracking Performance of the ATLAS Inner Tracker at the HL-LHC", CERN, Geneva, Tech. Rep. ATL-PHYS-PUB-2019-014, Mar. 2019, https://cds.cern.ch/record/2669540.

[26] "Fast Track Reconstruction for HL-LHC", CERN, Geneva, Tech. Rep. ATL-PHYS-PUB-2019-041, Oct. 2019, https://cds.cern.ch/record/2693670.

[27] *Benchmarking Working Group*, https://w3.hepix.org/benchmarking.html, Accessed: 2021-06-21.

[28] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[29] A. M. Turing, *Mechanical intelligence*. North-Holland, 1992, ISBN: 9780444880581.

[30] F. Rosenblatt, “The perceptron: a perceiving and recognizing automaton”, Cornell Aeronautical Laboratory, Tech. Rep. Report 85-60-1, 1957.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1, 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0.

[32] H. J. Kelley, “Gradient Theory of Optimal Flight Paths”, *ARS Journal*, vol. 30, no. 10, pp. 947–954, 1960. DOI: 10.2514/8.5282.

[33] A. E. Bryson, “A gradient method for optimizing multi-stage allocation processes”, *Proceedings of the Harvard Univ. Symposium on digital computers and their applications 3–6 April 1961*, 1962.

[34] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, *et al.*, “Backpropagation applied to handwritten zip code recognition”, *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[35] B. Yegnanarayana, “Artificial neural networks for pattern recognition”, en, *Sadhana*, vol. 19, no. 2, pp. 189–238, Apr. 1994, ISSN: 0973-7677. DOI: 10.1007/BF02811896.

[36] K. S. Oh and K. Jung, “GPU implementation of neural networks”, *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2004.01.013.

[37] R. Raina, A. Madhavan, and A. Y. Ng, “Large-Scale Deep Unsupervised Learning Using Graphics Processors”, in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09, Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 873–880, ISBN: 9781605585161. DOI: 10.1145/1553374.1553486.

[38] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, Big, Simple Neural Nets for Handwritten Digit Recognition”, *Neural Comput.*,

vol. 22, no. 12, pp. 3207–3220, Dec. 2010, ISSN: 0899-7667. DOI: 10.1162/NECO_a_00052.

[39] J. Pata, J. Duarte, J.-R. Vlimant, M. Pierini, and M. Spiropulu. (2021). "Mlpf: efficient machine-learned particle-flow reconstruction using graph neural networks". arXiv: 2101.08578 [physics.data-an].

[40] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks", in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, III–1310–III–1318. DOI: 10.5555/3042817.3043083.

[41] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, J. Kolen, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", in *A field guide to dynamical recurrent neural networks*. Wiley-IEEE Press, 2001, pp. 237–243.

[42] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[43] F. Gers, "Long short-term memory in recurrent neural networks", Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2001.

[44] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, *et al.*, *Relational inductive biases, deep learning, and graph networks*, 2018. arXiv: 1806.01261 [cs.LG].

[45] W. Guan, G. Perdue, A. Pesah, M. Schuld, K. Terashi, *et al.*, "Quantum machine learning in high energy physics", *Machine Learning: Science and Technology*, Oct. 2020, ISSN: 2632-2153. DOI: 10.1088/2632-2153/abc17d.

[46] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum Computation by Adiabatic Evolution", *arXiv:quant-ph/0001106*, Jan. 2000, arXiv: quant-ph/0001106.

[47] B. Denby, "Neural networks and cellular automata in experimental high energy physics", *Computer Physics Communications*, vol. 49, no. 3, pp. 429–448, 1988, ISSN: 0010-4655. DOI: 10.1016/0010-4655(88)90004-5.

[48] C. Peterson, "Track finding with neural networks", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 279, no. 3, pp. 537–545, 1989, ISSN: 0168-9002. DOI: 10.1016/0168-9002(89)91300-4.

[49] X. Ai, C. Allaire, N. Calace, A. Czirkos, I. Ene, *et al.*, "A Common Tracking Software Project", *arXiv:2106.13593 [hep-ex, physics:physics]*, Jun. 2021, arXiv: 2106.13593.

[50] *TrackML Particle Tracking Challenge*, https://www.kaggle.com/c/trackml-particle-identification, Accessed: 08-02-2021.

[51] R. P. Feynman, "Simulating physics with computers", *Int. J. Theor. Phys.*, vol. 21, no. 6, pp. 467–488, 1982, ISSN: 1572-9575. DOI: 10.1007/BF02650179.

[52] *IBM Quantum*, https://quantum-computing.ibm.com/, Accessed: 2021-04-06.

[53] *Amazon Web Services Braket*, https://aws.amazon.com/braket/, Accessed: 2021-05-17.

[54] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997, ISSN: 00975397. DOI: 10.1137/S0097539795293172. arXiv: 9508027 [quant-ph].

[55] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342.

[56] L. K. Grover, "A fast quantum mechanical algorithm for database search", in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219, ISBN: 0897917855. DOI: 10.1145/237814.237866.

[57] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations", *Phys. Rev. Lett.*, vol. 103, no. 15, pp. 1–4, 2009, ISSN: 00319007. DOI: 10.1103/PhysRevLett.103.150502.

[58] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, *et al.*, "A variational eigenvalue solver on a photonic quantum processor", *Nat. Commun.*, vol. 5, no. 1, p. 4213, 2014, ISSN: 2041-1723. DOI: 10.1038/ncomms5213.

[59] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, *et al.*, "Quantum Chemistry in the Age of Quantum Computing", *Chem. Rev.*, vol. 119, no. 19, pp. 10 856–10 915, Oct. 2019, ISSN: 0009-2665. DOI: 10.1021/acs.chemrev.8b00803.

[60] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning", *Phys. Rev. A*, vol. 98, no. 3, pp. 1–3, 2018, ISSN: 24699934. DOI: 10.1103/PhysRevA.98.032309. arXiv: 1803.00745.

[61] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, "Circuit-centric quantum classifiers", *Phys. Rev. A*, vol. 101, no. 3, pp. 1–8, 2020, ISSN: 24699934. DOI: 10.1103/PhysRevA.101.032308. arXiv: 1804.00633.

[62] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors", *arXiv*, 2018, ISSN: 23318422. arXiv: 1802.06002.

[63] *La sphère de bloch*, http://stla.github.io/stlapblog/posts/BlochSphere.html, Accessed: 2021-04-19.

[64] H. Abraham, AduOffei, R. Agarwal, I. Y. Akhalwaya, G. Aleksandrowicz, *et al.*, *Qiskit: An Open-source Framework for Quantum Computing*, 2019. DOI: 10.5281/zenodo.2562110.

[65] J. Preskill, "Quantum computing in the NISQ era and beyond", *Quantum*, vol. 2, no. July, pp. 1–20, 2018, ISSN: 2521327X. DOI: 10.22331/q-2018-08-06-79. arXiv: 1801.00862.

[66] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation", *Fortschritte der Phys.*, vol. 48, no. 9-11, pp. 771–783, 2000, ISSN: 00158208. DOI: 10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E. arXiv: 0002077 [quant-ph].

[67] *D-Wave: The Quantum Computing Company*, https://www.dwavesys.com, Accessed: 2021-04-08.

[68] *Advantage; The first and only quantum computer built for business*, https://www.dwavesys.com/sites/default/files/Advantage_Datasheet_v9_0.pdf, Accessed: 2021-04-08.

[69] F. Neukart, G. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, *et al.*, "Traffic flow optimization using a quantum annealer", *arXiv:1708.01625 [quant-ph]*, Aug. 2017, arXiv: 1708.01625.

[70] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, *et al.*, "Quantum supremacy using a programmable superconducting processor", *Nature*, vol. 574, no. 7779, pp. 505–510, 2019, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5.

[71] *Rigetti*, https://www.rigetti.com, Accessed: 2021-04-06.

[72] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, *et al.*, "Benchmarking an 11-qubit quantum computer", *Nat. Commun.*, vol. 10, no. 1, p. 5464, 2019, ISSN: 2041-1723. DOI: 10.1038/s41467-019-13534-2.

[73] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, *et al.*, "Demonstration of the QCCD trapped-ion quantum computer architecture", *arXiv*, vol. 592, no. July 2020, 2020, ISSN: 23318422. DOI: 10.1038/s41586-021-03318-4. arXiv: 2003.01293.

[74] H. S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, *et al.*, "Quantum computational advantage using photons", *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020, ISSN: 0036-8075. DOI: 10.1126/science.abe8770.

[75] *Xanadu*, https://www.xanadu.ai, Accessed: 2021-04-08.

[76] J. M. Arrazola, V. Bergholm, K. Brádler, T. R. Bromley, M. J. Collins, *et al.*, "Quantum circuits with many photons on a programmable nanophotonic chip", *Nature*, vol. 591, no. 7848, pp. 54–60, 2021, ISSN: 1476-4687. DOI: 10.1038/s41586-021-03202-1.

[77] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. Borgorin, *et al.*, "Demonstration of quantum volume 64 on a superconducting quantum computing system", *Quantum Sci. Technol.*, vol. 64, 2021, ISSN: 2058-9565. DOI: 10.1088/2058-9565/abe519. arXiv: 2008.08571.

[78] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[79] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits", *Phys. Rev. A*, vol. 100, no. 3, p. 32 328, 2019, ISSN: 24699934. DOI: 10.1103/PhysRevA.100.032328. arXiv: 1811.12926.

[80] R. Blume-Kohout and K. Young, "A volumetric framework for quantum computer benchmarks", *Quantum*, vol. 4, pp. 1–28, 2020, ISSN: 2521327X. DOI: 10.22331/Q-2020-11-15-362. arXiv: 1904.05546.

[81] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, *et al.*, *Pennylane: automatic differentiation of hybrid quantum-classical computations*, 2020. arXiv: 1811.04968 [quant-ph].

[82] Cirq Developers, *Cirq*, version v0.10.0, Mar. 2021. DOI: 10.5281/zenodo.4586899.

[83] *List of QC simulators*, https://quantiki.org/wiki/list-qc-simulators, Accessed: 2021-04-08.

[84] J. Abhijith, A. Adedoyin, J. Ambrosiano, P. Anisimov, A. D. Bärtschi, *et al.*, "Quantum algorithm implementations for beginners", *arXiv*, 2018, ISSN: 23318422. arXiv: 1804.03719.

[85] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, *et al.*, "Variational Quantum Algorithms", 2020. arXiv: 2012.09265 [quant-ph].

[86] G. G. Guerreschi and M. Smelyanskiy, *Practical optimization for hybrid quantum-classical algorithms*, 2017. arXiv: 1701.01450 [quant-ph].

[87] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*, ser. Quantum Science and Technology. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9.

[88] R. Larose and B. Coyle, "Robust data encodings for quantum classifiers", *Phys. Rev. A*, vol. 102, no. 3, pp. 1–24, 2020, ISSN: 24699934. DOI: 10.1103/PhysRevA.102.032420. arXiv: 2003.01695.

[89] D. Ventura and T. Martinez, "Quantum associative memory", *Information Sciences*, vol. 124, no. 1, pp. 273–296, 2000, ISSN: 0020-0255. DOI: 10.1016/S0020-0255(99)00101-2.

[90] C. A. Trugenberger, "Probabilistic quantum memories", *Phys. Rev. Lett.*, vol. 87, p. 067 901, 6 Jul. 2001. DOI: 10.1103/PhysRevLett.87.067901.

[91] M. Schuld and F. Petruccione, "Quantum Information", in *Supervised Learning with Quantum Computers*. Springer International Publishing, 2018, pp. 75–125, ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9_3.

[92] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, "Quantum embeddings for machine learning", *arXiv*, 2020, ISSN: 23318422. arXiv: 2001.03622.

[93] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum machine learning models", *Phys. Rev. A*, vol. 103, no. 3, p. 032 430, 2021. DOI: 10.1103/PhysRevA.103.032430.

[94] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm", *arXiv:1411.4028 [quant-ph]*, Nov. 2014, arXiv: 1411.4028.

[95] J. Lee, W. J. Huggins, M. Head-Gordon, and K. B. Whaley, "Generalized Unitary Coupled Cluster Wave functions for Quantum Computation", *J. Chem. Theory Comput.*, vol. 15, no. 1, pp. 311–324, Jan. 2019, ISSN: 1549-9618. DOI: 10.1021/acs.jctc.8b01004.

[96] A. S. Bhatia, M. K. Saggi, A. Kumar, and S. Jain, "Matrix product state–based quantum classifier", *Neural Comput.*, vol. 31, no. 7, pp. 1499–1517, 2019, ISSN: 1530888X. DOI: 10.1162/neco_a_01202. arXiv: 1905.01426.

[97] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, *et al.*, "Hierarchical quantum classifiers", *npj Quantum Inf.*, vol. 4, no. 1, 2018, ISSN: 20566387. DOI: 10.1038/s41534-018-0116-9. arXiv: 1804.03680.

[98] A. V. Uvarov, A. S. Kardashin, and J. D. Biamonte, "Machine learning phase transitions with a quantum processor", *Phys. Rev. A*, vol. 102, p. 012 415, 1 Jul. 2020. DOI: 10.1103/PhysRevA.102.012415.

[99] Y.-Y. Shi, L.-M. Duan, and G. Vidal, "Classical simulation of quantum many-body systems with a tree tensor network", *Phys. Rev. A*, vol. 74, p. 022 320, 2 Aug. 2006. DOI: 10.1103/PhysRevA.74.022320.

[100] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, *et al.*, "Variational Quantum Algorithms", *arXiv*, pp. 1–29, 2020, ISSN: 23318422.

[101] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms", *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1 900 070, 2019. DOI: https://doi.org/10.1002/qute.201900070.

[102] M. Lundberg and L. Svensson, "The Haar measure and the generation of random unitary matrices", in *Processing Workshop Proceedings, 2004 Sensor Array and Multichannel Signal*, 2004, pp. 114–118. DOI: 10.1109/SAM.2004.1502919.

[103] S. Kullback and R. A. Leibler, "On Information and Sufficiency", *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. DOI: 10.1214/aoms/1177729694.

[104] T. Hubregtsen, J. Pichlmeier, P. Stecher, and K. Bertels, "Evaluation of parameterized quantum circuits: on the relation between classification accuracy, expressibility, and entangling capability", *Quantum Mach. Intell.*, vol. 3, no. 1, p. 9, 2021, ISSN: 2524-4914. DOI: 10.1007/s42484-021-00038-w.

[105] D. A. Meyer and N. R. Wallach, "Global entanglement in multiparticle systems", *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4273–4278, 2002. DOI: 10.1063/1.1497700.

[106] G. K. Brennen, "An observable measure of entanglement for pure states of multi-qubit systems", *Quantum Info. Comput.*, vol. 3, no. 6, pp. 619–626, Nov. 2003, ISSN: 1533-7146. DOI: 10.5555/2011556.2011561.

[107] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. arXiv: 1412.6980 [cs.LG].

[108] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, *et al.*, "Tensorflow: a system for large-scale machine learning", in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[109] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, *et al.*, "Pytorch: an imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, *et al.*, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. arXiv: 1912.01703 [cs.LG].

[110] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, *et al.*, "Scikit-Learn: Machine Learning in Python", *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011, ISSN: 1532-4435.

[111] G. G. Guerreschi and M. Smelyanskiy, *Practical optimization for hybrid quantum-classical algorithms*, 2017. arXiv: 1701.01450 [quant-ph].

[112] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms", in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. DOI: 10.5555/2999325.2999464.

[113] J. Kennedy and R. Eberhart, "Particle swarm optimization", in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[114] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations", *J. Glob. Optim.*,

vol. 56, no. 3, pp. 1247–1293, 2013, ISSN: 1573-2916. DOI: 10.1007/s10898-012-9951-y.

[115] Z. C. Yang, A. Rahmani, A. Shabani, H. Neven, and C. Chamon, "Optimizing Variational Quantum Algorithms Using Pontryagin's Minimum Principle", *Phys. Rev. X*, vol. 7, p. 021 027, 2 May 2017. DOI: 10.1103/PhysRevX.7.021027.

[116] D. Wecker, M. B. Hastings, and M. Troyer, "Training a quantum optimizer", *Phys. Rev. A*, vol. 94, p. 022 309, 2 Aug. 2016. DOI: 10.1103/PhysRevA.94.022309.

[117] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[118] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware", *Phys. Rev. A*, vol. 99, p. 032 331, 3 Mar. 2019. DOI: 10.1103/PhysRevA.99.032331.

[119] G. E. Crooks, *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition*, 2019. arXiv: 1905.13311 [quant-ph].

[120] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes", *Nat. Commun.*, vol. 9, no. 1, pp. 1–8, 2018, ISSN: 20411723. DOI: 10.1038/s41467-018-07090-4. arXiv: 1803.11173.

[121] E. Grant, M. Ostaszewski, L. Wossnig, and M. Benedetti, "An initialization strategy for addressing barren plateaus in parametrized quantum circuits", *Quantum*, vol. 3, 2019, ISSN: 2521327X. DOI: 10.22331/q-2019-12-09-214. arXiv: 1903.05076.

[122] A. Pesah, M. Cerezo, S. Wang, T. Volkoff, A. T. Sornborger, *et al.*, "Absence of Barren Plateaus in Quantum Convolutional Neural Networks", 2020. arXiv: 2011.02966 [quant-ph].

[123] C. O. Marrero, M. Kieferová, and N. Wiebe, "Entanglement Induced Barren Plateaus", 2020. arXiv: 2010.15968.

[124] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, *et al.*, "Noise-Induced Barren Plateaus in Variational Quantum Algorithms", 2020, ISSN: 23318422. arXiv: 2007.14384.

[125] B. Villalonga, S. Boixo, B. Nelson, C. Henze, E. Rieffel, *et al.*, "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware", *npj Quantum Information*, vol. 5, no. 1, p. 86, Oct. 10, 2019, ISSN: 2056-6387. DOI: 10.1038/s41534-019-0196-1.

[126] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, *et al.*, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[127] V. Leyton-Ortega, A. Perdomo-Ortiz, and O. Perdomo, "Robust implementation of generative modeling with parametrized quantum circuits", *Quantum Mach. Intell.*, vol. 3, no. 1, p. 17, 2021, ISSN: 2524-4914. DOI: 10.1007/s42484-021-00040-2.

[128] Y. Suzuki, Y. Kawase, Y. Masumura, Y. Hiraga, M. Nakadai, *et al.*, *Qulacs: a fast and versatile quantum circuit simulator for research purpose*, 2020. arXiv: 2011.13524 [quant-ph].

[129] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, *et al.*, *Tensorflow quantum: a software framework for quantum machine learning*, 2020. arXiv: 2003.02989 [quant-ph].

[130] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms", *Pattern Recogn.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997, ISSN: 0031-3203. DOI: 10.1016/S0031-3203(96)00142-2.

[131] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2.

[132] T. G. Draper and S. A. Kutin, *<q|pic>: Quantum circuits made easy*, https://github.com/qpic/qpic, 2020.

[133] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[134] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, *et al.*, "The power of quantum neural networks", 2020. arXiv: 2011.00027 [quant-ph].

[135] K. Zhang, M.-H. Hsieh, L. Liu, and D. Tao, *Toward trainability of quantum neural networks*, 2020. arXiv: 2011.06258 [quant-ph].

[136] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, *et al.*, "Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors", 2020. arXiv: 2003.11603.

[137] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, "An initialization strategy for addressing barren plateaus in parametrized quantum circuits", *Quantum*, vol. 3, p. 214, 2019, ISSN: 2521-327X. DOI: 10.22331/q-2019-12-09-214. arXiv: 1903.05076.

[138] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, *et al.*, "Power of data in quantum machine learning", *Nature Communications*, vol. 12, no. 1, p. 2631, May 2021, ISSN: 2041-1723. DOI: 10.1038/s41467-021-22539-9.

APPENDIX A

DEFINITIONS OF QUANTUM GATES

$$\text{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \text{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \text{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{A.1}$$

$$\text{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \text{S} = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \qquad \text{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{A.2}$$

$$R_X(\theta) = \begin{bmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{bmatrix} \tag{A.3}$$

$$R_Y(\theta) = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix} \tag{A.4}$$

$$R_Z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \tag{A.5}$$

$$\text{CX} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \text{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{A.6}$$