Distributed Learning in a Quantum-Classical Hybrid System

BY

Anthony D'Onofrio Jr.
MS, Fordham University, 2023

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF SCIENCE

IN THE DEPARTMENT OF COMPUTER AND INFORMATION

SCIENCE

AT FORDHAM UNIVERSITY

NEW YORK

MAY, 2023

Contents

List of Tables

List of Figures

INTRODUCTION

Machine learning and deep learning have made significant advances in recent years, transforming the way we approach and solve complex problems. These techniques leverage the power of artificial neural networks to learn patterns and relationships in data, enabling machines to make accurate predictions and decisions based on large and complex datasets.

However, the viability of machine learning and deep learning on classical computers has began to come into question as the need for complex problem solving has increased. One of the restrictions of classical computing is the limited processing power it can support. Classical computers are limited by the physical constraints of their components, such as transistors. As a result, they can only perform a finite number of operations per second, making them unable to handle complex and massive computations. Classical computers also have memory limitations which restricts their ability to handle large datasets or perform memory-intensive computations.

These limitations have motivated the development of alternative computing paradigms, such as quantum computing, which offer new ways of processing information and solving problems. While classical computing will remain a vital tool for many applications, the limitations it faces mean that it is not suitable for solving certain types of problems that require higher processing power, faster speed, or more memory capacity.

Quantum computing is a new paradigm of computing that uses the principles of quantum mechanics to process and manipulate information. Unlike classical computers, which operate using bits that can only be in one of two states (0 or 1), quantum computers use quantum bits, or qubits, which can exist in multiple states simultaneously. This property, known as superposition, allows quantum computers to perform certain types of computations much more efficiently than classical computers.

One advantage quantum computing has over classical computing is parallelism. Quantum computers can perform many computations simultaneously, whereas classical computers perform computations sequentially. This parallelism enables quantum computers to solve certain problems exponentially faster than classical computers. Quantum computing also allows for the development of specialized quantum algorithms that can solve specific problems much more efficiently than classical algorithms. For example, Shor's algorithm can efficiently factor large integers, which is a task that is believed to be difficult for classical computers.

The history of quantum computing dates back to the early 1980s, when physicist Richard Feynman proposed the idea of using quantum systems to simulate complex physical systems that are difficult to model using classical computers [1]. This idea was later formalized by physicist Paul Benioff, who proposed a theoretical model for a quantum computer.

In the 1990s, several key breakthroughs were made in the development of quantum computing hardware. In 1994, Peter Shor, a mathematician at Bell Labs, developed an algorithm for a quantum computer that could factor large integers much faster than classical computers. This algorithm demonstrated the potential for quantum computers to solve problems that are intractable for classical computers. In 1995, physicist Seth Lloyd proposed the first physical implementation of a quantum computer using a system of interacting atoms or ions, known as a quantum register.

In the early 2000s, experimentalists began to build small-scale quantum computers using various physical systems, including trapped ions, superconducting circuits, and quantum dots. These early quantum computers demonstrated some of the key properties of quantum computing, such as superposition and entanglement, but were limited in their size and scalability.

In 2016, the IBM-Q Experience was introduced, providing an open-source quantum computing platform that developers could access [2]. Then, in 2019, Google AI announced that they had accomplished quantum supremacy by completing a series of tasks in only 200 seconds using a 53-qubit computer, a feat that would take a classical computer over 10,000 years to complete [3]. Before this milestone was reached, quantum supremacy was only a theoretical idea.

Despite these achievements of quantum computing, it still has its drawbacks. One of the most significant challenges is the issue of decoherence, where the fragile quantum states of qubits can be disrupted by environmental noise. This issue requires the development of error-correcting codes and fault-tolerant quantum computing, which is still an active area of research.

Another limitation is the fact that most of the quantum computers available to the public contain only five or seven qubits. Increasingly complex problems require more qubits, but they are not readily available to the public. Moreover, with an increase in the number of qubits, the complexity of the quantum system grows exponentially [4]. Consequently, this results in a higher level of noise in the quantum device overall.

A potential solution to resource limitation challenges is the use of a distributed system to complete the necessary task. A distributed system is a network of interconnected computers that work together to achieve a common goal. In such a system, each computer (also called a node) operates independently and has its own memory and processing capabilities. The nodes communicate with each other by passing messages over a network, and collaborate to provide a service or run an application. The key characteristic of a distributed system is that the nodes work together as a single system, and the user sees the system as a whole, rather than a collection of independent computers.

While distributed systems are a valid approach to utilizing resources efficiently, they have to be carefully managed to ensure their success. Managing the resources in a distributed system involves ensuring that there are enough resources available to meet the system's requirements, such as CPU, memory, and storage. Resource management also involves monitoring resource usage, identifying bottlenecks, and dynamically allocating resources as needed. Data management is also an important aspect of a distributed system. It involves ensuring that data is consistent and available to all nodes in the system. This is often achieved through data replication, where data is stored on multiple nodes, or through partitioning, where data is split across multiple nodes.

Distributed systems also pose a bigger security risk than traditional systems because there are more computers involved. The bigger the system is, the more likely it is that a portion of it can become compromised. Security management in distributed systems involves ensuring that the system is secure from external threats, such as hacking or denial-of-service attacks. It also involves managing access control and ensuring that data is encrypted and transmitted securely. That is why distributed systems also need to be designed to provide fault tolerance to make up for any outages caused by security breaches or other issues.

Since quantum computers with five and seven qubits are the most readily available resource, it would be more efficient to split a task up amongst multiple smaller quantum computers rather than using one quantum computer with a higher qubit count. The jobs could be split up using a Convolutional Neural Network (CNN). A CNN is a type of artificial neural network that is commonly used in image and video processing applications. The key idea behind a CNN is to use a mathematical operation called convolution to extract features from an input image or video.

The convolution operation involves sliding a small matrix called a filter or kernel over the input image, and performing a dot product between the filter and the corresponding patch of the input. This process generates a new matrix, known as a feature map, that highlights the areas of the input that are most relevant for a specific task. These filters can then be sent to various quantum machines that complete the jobs in parallel.

In this work, we introduce a distributed quantum system. IBM-Q's current system is not shared. It is a series of independent computers. Our system will allow for efficient management of qubits on each individual machine while also expanding the number of qubits available by managing a concurrent distributed system of quantum computers.

Chapter 1

RELATED WORK

Machine learning and deep learning applications are computationally intensive and data hungry [5]. Typically, large learning models are trained and deployed on the cloud, a distributed system. Many proposals aim to optimize the performance of deep learning models in a distributed environment from different perspectives, such as resource allocation [6]–[18], user experiences [19]–[23], and data center management [24]–[31]. For instance, researchers have proposed Differentiated Quality of Experiences (DQoEs)[19], a framework that accommodates clients' specifications regarding targeted quality of experiences and dynamically adjusts resources to meet user-defined deep learning objectives. In FlowCon[18], the system allocates resources based on the growth efficiency of individual deep learning models. Such optimizations considerably enhance deep learning performance from a system perspective. However, due to the ever-growing data sizes, increased complexity of model designs, and hardware limitations, achieving these improvements poses an increasingly challenging task.

Due to the great potential of processing complex problems beyond current abilities at a fraction of the time, quantum-based algorithms and applications have received great attention recently as innovations in quantum machine learning [32]–[48] and have been proposed to improve existing models from various perspectives. However, these proposals either focus on theoretical proofs that assume unlimited quantum resources or primitive applications with a toy-like data set. The main reason lies in the fact that practical applications of quantum computers require a large amount of physical qubits, which is unfeasible in the NISQ era. With the current quantum development frameworks, programming languages and compilers [49]–[55] significantly expedite prototyping quantum algorithms and applications. Unfortunately, none of the existing systems support distributed quantum computing in a practical setting.

Recent efforts have been made to push it a step forward [56]–[61]. For example, authors in [58] propose a distributed programming language that allows local quantum operations and classical communication. It investigates the formal description and verification of distributed quantum systems. Based on established distributed systems, QMPI [59] proposes an extension of the classical Message Passing Interface to enable high-performance implementations of distributed quantum algorithms. In addition, it includes a quantum communication model (SENDQ) to evaluate the performance of these algorithms and foster algorithmic optimizations. Besides, a distributed quantum simulator, IQS, is presented by Intel Labs [60], [61]. IQS is able to utilize distributed resources in a cloud computing infrastructure and accelerate the simulation. However, they consider a system that consists entirely of quantum computers. The classical computing resources are merely intended for simulation. They fail to leverage these widely-available cloud resources to solve a practical problem with quantum computers collaboratively.

Chapter 2

BACKGROUND AND MOTIVATION

2.1 What is Quantum?

Quantum physics is a mathematical model that was first used to describe the behavior of small things in a lab. Quantum physics plays a fundamental role in quantum computing because it provides the underlying principles and laws that govern the behavior of quantum systems. Quantum computing utilizes these principles to perform certain computations at an exponential speedup compared to classical computing.

2.2 Quantum Superposition

Qubits, or quantum bits, are the basic building blocks of quantum computing. Unlike classical bits that can only exist in one of two possible states (0 or 1), qubits can exist in multiple states at the same time due to the phenomenon known as quantum superposition.

Quantum superposition is a fundamental property of quantum mechanics that allows a qubit to exist in multiple states at the same time. These states are represented by a complex vector in a mathematical space known as a Hilbert space. For example, consider a qubit that can be in a state of 0 or 1, represented by the two basis vectors $|0\rangle$ and $|1\rangle$, respectively. In quantum computing, this qubit can also exist in a superposition of these two states, represented by a linear combination of the two basis vectors as seen in Equations 2.1 and 2.2.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, |\Psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.1}$$

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2}$$

These equations are an example of $\langle bra|\ |ket\rangle$ notation, where $\langle bra|$ and $|ket\rangle$ represent horizontal and vertical quantum state vectors. The linear combination described by them is referred to as a qubit's statevector. $|0\rangle$ and $|1\rangle$ are orthonormal vectors in an eigenspace. In Equation 2.2, $|\Psi\rangle$ represents the qubit state, a probabilistic combination of $|0\rangle$ and $|1\rangle$.

In addition, the quantum states of multiple qubits can be described using the tensor product of qubit states. The tensor product between qubits is shown in Equations 2.2 and 2.3 can be described using Equation 2.4.

$$|\Phi\rangle = \gamma|0\rangle + \omega|1\rangle \tag{2.3}$$

$$|\Psi\Phi\rangle = |\Psi\rangle \otimes |\Phi\rangle = \gamma\alpha|00\rangle + \omega\alpha|01\rangle + \gamma\beta|10\rangle + \omega\beta|11\rangle \tag{2.4}$$

2.3 The Bloch Sphere

Quantum states can be visualized using the Bloch sphere as seen in Figure 2.1. The Bloch sphere provides a convenient way to visualize the state of a qubit as a point on the surface of a sphere, where the north and south poles represent the states $|0\rangle$ and $|1\rangle$, respectively, and the equator represents the states that are in a superposition of $|0\rangle$ and $|1\rangle$. The surface of the sphere can be thought of as a two-dimensional complex vector space, where the z-axis represents the real part of the probability amplitude of the $|0\rangle$ state and the x- and y-axes represent the real and imaginary parts of the probability amplitude of the $|1\rangle$ state.

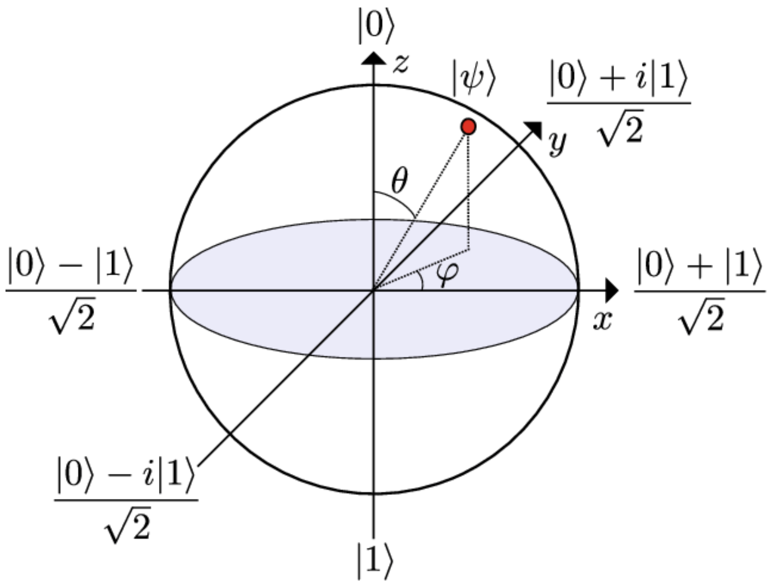

$|0\rangle$
z
$|\psi\rangle$
$\frac{|0\rangle + i|1\rangle}{\sqrt{2}}$
y
θ
$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$
φ
$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$
x
$\frac{|0\rangle - i|1\rangle}{\sqrt{2}}$
$|1\rangle$

Fig. 2.1: Visualization of the Bloch Sphere

Any arbitrary state of a qubit can be represented as a point on the surface of the Bloch sphere. For example, the state of a qubit in a superposition of $|0\rangle$ and $|1\rangle$, such as ($|0\rangle$ +

$|1\rangle$)/$\sqrt{2}$, is represented as a point on the equator of the sphere. Similarly, the state of a qubit in a superposition of $|0\rangle$ and -$|1\rangle$, such as ($|0\rangle$ - $|1\rangle$)/$\sqrt{2}$, is represented as a point on the equator that is opposite to the first example.

2.4 Quantum Gates

Quantum gates are the basic building blocks of quantum circuits. In a quantum circuit, quantum gates are used to manipulate the state of qubits. A quantum gate is a unitary operator that acts on the state of one or more qubits to transform them into a new state. These gates are represented by matrices, which are used to calculate the output state of a qubit given its input state. Quantum gates either perform a rotation about an axis or an operation on a qubit based on the value of another qubit. These operations are called rotation gates and controlled gates.

2.4.1 Pauli Gates

Pauli gates are a set of quantum gates that are named after the physicist Wolfgang Pauli. There are three Pauli gates, each of which acts on a single qubit and is represented by a 2x2 unitary matrix. These gates are denoted by the symbols X, Y, and Z, and they are defined as follows:

- **Pauli-X gate (X gate):** Equivalent to a classical NOT gate and flips the state of a single qubit from $|0\rangle$ to $|1\rangle$ or vice versa. It is represented by Equation 2.5:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \langle 0| \; \langle 1| + |1\rangle \; |0\rangle \tag{2.5}$$

- **Pauli-Y gate (Y gate):** Rotates the state of a single qubit around the y-axis of the Bloch sphere by π radians. It is represented by Equation 2.6:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i \, \langle 0| \; \langle 1| + i|1\rangle \; |0\rangle \tag{2.6}$$

- **Pauli-Z gate (Z gate):** Applies a phase shift of π to the state $|1\rangle$ and leaves the state $|0\rangle$ unchanged. It is represented by Equation 2.7:

$$Z = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = \langle 0| \; \langle 0| - |1\rangle \; |1\rangle \tag{2.7}$$

2.4.2 Hadamard Gate

The Hadamard gate is a single-qubit gate that transforms a qubit from the standard basis state ($\langle 0|$ or $|1\rangle$) to a superposition of both states. It is represented by Equation 2.8.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.8}$$

One important property of the Hadamard gate is that it is its own inverse. Therefore, applying the Hadamard gate twice returns the qubit to its original state. This property makes the Hadamard gate useful for implementing quantum circuits, where it can be used to prepare superposition states and to perform measurements in different bases.

2.4.3 CNOT Gate

The CNOT (Controlled-NOT) gate is a conditional gate that flips the target qubit if the control qubit is in the $|1\rangle$ state, and leaves it unchanged otherwise. The CNOT gate is commonly used in quantum circuits for entangling qubits and implementing quantum algorithms. It is represented by Equation 2.9.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.9}$$

The two qubits in a CNOT gate are usually labeled as the control qubit (C) and the target qubit (T). The CNOT gate acts on the two-qubit state —C⟩—T⟩ as follows:

- If the control qubit is in the $\langle 0|$ state, the CNOT gate leaves the two-qubit state unchanged.
- If the control qubit is in the $|1\rangle$ state, the CNOT gate flips the state of the target qubit. Therefore, it applies the Pauli-X gate to the target qubit.

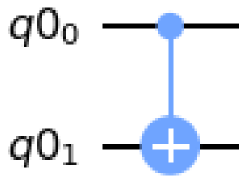


Fig. 2.2: CNOT Gate Circuit Notation

Figure 2.2 represents the circuit notation of a CNOT gate. q_0 is the control qubit and q_1 is the target qubit.

2.4.4 CSWAP Gate

A controlled SWAP gate is a two-qubit quantum gate that swaps the states of two qubits if a control qubit is in the —1⟩ state and leaves them unchanged otherwise. The controlled SWAP gate is an important gate in quantum computing because it allows for the exchange of information between qubits without changing the overall state of the system. It is represented by Equation 2.10.

$$CSWAP(q_0, q_1, q_2) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

The two qubits in a controlled SWAP gate are usually labeled as the control qubit (C) and the target qubits (T1 and T2). The controlled SWAP gate acts on the three-qubit state —C⟩—T1⟩—T2⟩ as follows:

- If the control qubit is in the $|0\rangle$ state, the controlled SWAP gate leaves the three-qubit state unchanged.

- If the control qubit is in the $|1\rangle$ state, the controlled SWAP gate swaps the states of the two target qubits if they are different, and leaves them unchanged if they are the same.

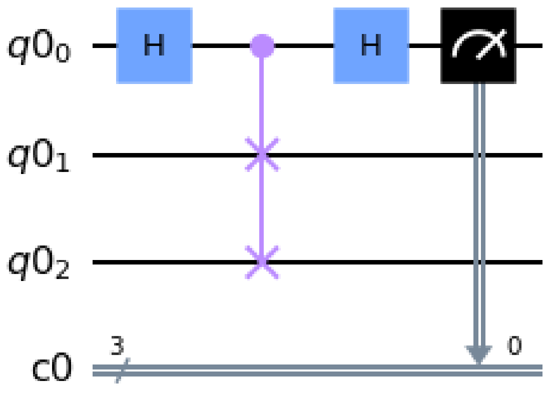


Fig. 2.3: Swap Test Circuit Notation

Figure 2.4 displays the swap test being performed. During the swap test, the ancilla qubit, q_0, is placed in superposition via a Hadamard gate. The swap test is then performed between q_1 and q_2 and measured onto q_0. Then, another Hadamard gate is performed on q_0. Lastly, the ancilla qubit is measured onto a classical bit.

2.5 Quantum Entanglement

Quantum entanglement is a phenomenon that occurs in quantum computing, where two or more qubits can be in a superposition state that is entangled. This means that if a measurement is made on one of the qubits, it will instantaneously affect the state of the other entangled qubits, regardless of their separation distance.

The simplest form of quantum entanglement is the Bell state. The Bell state is a superposition of two qubits that are entangled, and it is defined in Equation 2.11.

$$\langle\Psi| = \frac{\langle 00| + |11\rangle}{\sqrt{2}} \tag{2.11}$$

In this state, both qubits are in a superposition of being either in the state $|0\rangle$ or $|1\rangle$. However, the states of the two qubits are entangled, meaning that if one qubit is measured to be in the state $|0\rangle$, the other qubit must be in the state $|1\rangle$, and vice versa. This correlation between the qubits remains even if they are separated by a large distance.

2.6 Quantum Circuits

A quantum circuit is a set of instructions that involves performing a series of coherent quantum operations on quantum data, specifically qubits, and simultaneous real-time classical computation. The circuit is composed of a sequence of quantum gates, measurements, and resets, and these operations can be dependent on and use data from classical computation. An example of a quantum circuit can be seen in Figure 2.4.

The sample circuit contains three qubits. All have Hadamard gates applied to them which ensures they are in superposition. They have an equal chance of being measured as a 0 or 1. The input data is then encoded using R_y rotation gates. A series of CNOT gates are then applied to the qubits so that they become entangled. Their measurements now depend on the other qubits in superposition, which increases the representational power of the circuit. R_x, and R_z gates are then applied to each of the qubits, which creates trainable parameters that enable the manipulation of the probability distributions of qubit measurements. Finally, q_0, q_1,, and q_2 are all measured onto a classical bit. This measurement is represented by the block boxes at the far right of the circuit diagram. Once measured onto a classical bit, a qubit collapses to a single value and is represented by a binary bit (either 0 or 1). It can no longer be in superposition when measured. This circuit has a total of 9 trainable parameters (3 for each qubit).

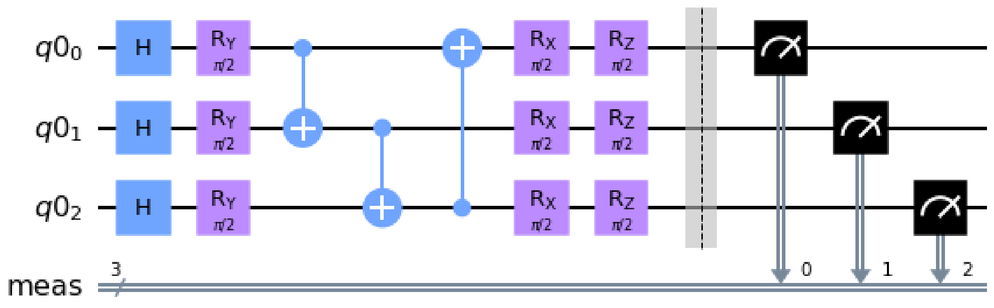


Fig. 2.4: Sample Quantum Circuit

There are four main components of a quantum circuit:

- **Initialization and reset:** Establishes a clearly defined quantum state. This is accomplished through the use of initialization and reset operations. Resets can be carried out through a combination of single-qubit gates and concurrent classical computation, which monitors measurements to confirm whether the desired state has been created successfully. Once this is achieved, the initialization of qubit q_0 into a specific state $|\Psi\rangle$ can be performed by applying single-qubit gates.
- **Quantum gates:** Utilize a series of quantum gates to modify the qubits in a manner necessary for the algorithm
- **Measurements:** Measure the qubits. The measurements of each qubit are interpreted as classical outcomes, which can only be either 0 or 1. These outcomes are then stored in two classical bits. Qubits in superposition collapse to a single value when measured.
- **Classically conditioned quantum gates:** Perform single-qubit operations using Z and X gates on the third qubit. These gates are dependent on the outcomes of measurements

that have been stored in two classical bits. The classical computation results are used simultaneously and in real-time in the same quantum circuit.

2.7 Distributed Systems

A distributed system is a type of computer system that is composed of multiple interconnected devices or nodes, which work together to achieve a common goal. Each node in a distributed system is independent and can perform its own computation and storage tasks. These nodes are connected by a central computer system that contains the software that coordinates activities and shares resources among the nodes. The data collected by this software is stored in the databases housed by the system. This architecture can be seen in Figure 2.5.

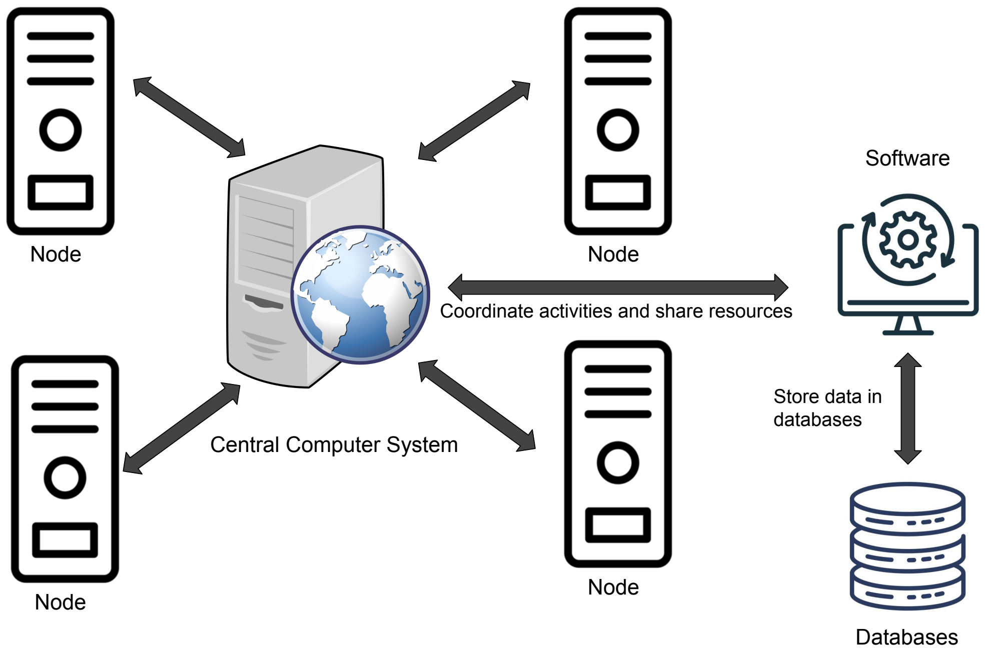


Fig. 2.5: Distributed System Architecture

Some of the advantages of a distributed system are:

- **Scalability:** The system's ability to scale up or down to meet the changing demands of its users. This is because the workload is distributed among multiple nodes, and new nodes can be added to the system to handle increased traffic or data.
- **Fault tolerance:** The system's ability to be resilient in the face of hardware or software failures. If one node fails, the system can continue to function using the remaining nodes.
- **High availability:** The system's ability to provide high availability by replicating data across multiple nodes. This ensures that if one node goes offline, the data is still available from other nodes.
- **Improved performance:** The system's ability to achieve better performance than a centralized system. This is because the nodes can work in parallel to complete tasks faster.
- **Geographic distribution:** The system's ability to be deployed across multiple geographic locations, which can improve performance for users who are geographically dispersed.
- **Reduced cost:** The system's ability to be less expensive than a centralized system, as it can use commodity hardware and can be scaled up or down as needed.

Overall, a distributed system offers greater flexibility, resilience, and performance than a centralized system, making it an attractive option for many modern computing applications. However, there are some drawbacks to using distributed systems including complexity, security, latency, and compatibility issues. These disadvantages must be carefully evaluated when attempting to implement a distributed system.

2.8 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a Deep Learning algorithm that can take an input image, assign importance to various characteristics of the image, and be able to differentiate one from the other. The preprocessing for a CNN is much lower than other classical algorithms. A CNN is made up of three main layers: the convolutional layer, pooling layer, and fully-connected (FC) layer. An example of a CNN with these layers, classifying the traits of vehicles, can be seen in Figure 2.6.

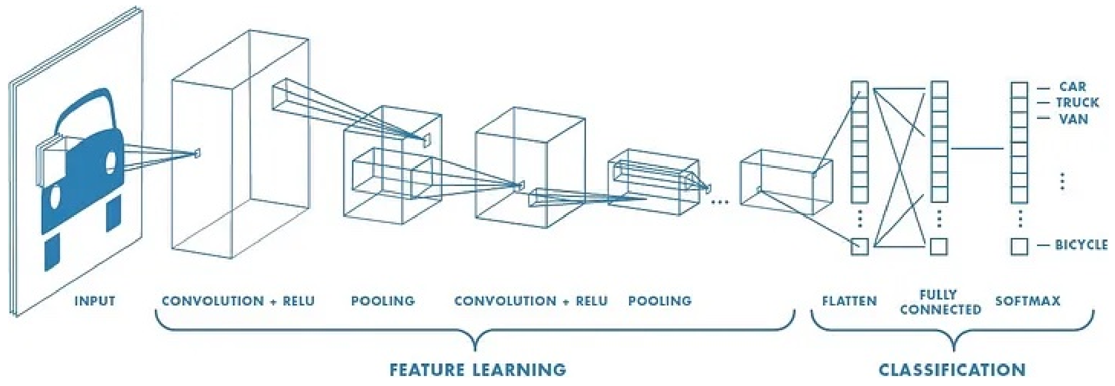


Fig. 2.6: Example of a CNN Extracting Features of Vehicles [62]

2.8.1 Convolutional Layer

The convolutional layer is the main building block of a CNN, where most of the computation occurs. It requires three components: input data, a filter, and a feature map. There is also a feature detector, known as a kernel, that moves across the receptive fields of the image, checking if the feature is present. This process is called convolution. The layers in a CNN are typically arranged in a hierarchical structure, with the earlier layers learning simple patterns such as edges and corners, and the later layers learning more complex patterns such as objects and scenes.

The kernel is a two-dimensional array of weights that represent part of the image. The filter of specified size determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product between the input pixels and the filter is calculated. This result is then fed into an output array. The filter then shifts by a certain number of pixels, known as the stride, repeating the process until the kernel has made its way across the whole image. The final output of the series of dot products is known as a feature map. After each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, which introduces nonlinearity to the model.

2.8.2 Pooling Layer

Pooling layers, also known as downsampling, conduct dimensionality reduction. This process reduces the number of parameters in the input. Similar to the convolution operation, the pooling operation sweeps a filter across the entire input. However, this time the filter does not have any weights. The kernel instead applies an aggregation function to the values within the receptive field, which populates the output array. A lot of information can be lost in the pooling layer, but its benefits include reducing complexity, improving efficiency, and limiting the risk of overfitting.

There are two types of pooling:

- **Max pooling:** When the filter moves over the input data, it chooses the pixel that has the highest value and sends it to the output array. This method is more commonly used than average pooling.
- **Average pooling:** When the filter slides over the input data, it computes the mean value of the pixels within its receptive field and transmits it to the output array.

2.8.3 Fully-Connected Layer

In the fully-connected (FC) layer, each node in the output layer directly connects to a node in the previous layer. This layer performs the task of classifications based on the features that have been extracted from the processes of the previous layers. FC layers usually use a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

CNNs have several advantages over traditional computer vision techniques. They can automatically learn features from the data, which can save time and effort in feature engineering. They can also handle variations in scale, rotation, and orientation, making them robust to changes in the input data. Additionally, CNNs can be trained using backpropagation, which is an efficient algorithm for optimizing the network weights. While these advantages are common in the field of Machine Learning, utilizing these advantages within a distributed quantum system allows for a more efficient use of publicly available quantum resources like never before.

However, CNNs also have some limitations. They require a large amount of training data to perform well, and they can be computationally intensive, making them difficult to deploy on resource-constrained devices. Additionally, they are not well-suited for tasks that require understanding of the context or semantics of the input data. It is important to realize these possible drawbacks when implementing a CNN within a larger system such as a distributed quantum system that's proposed.

Chapter 3

System Architecture and Algorithm Design

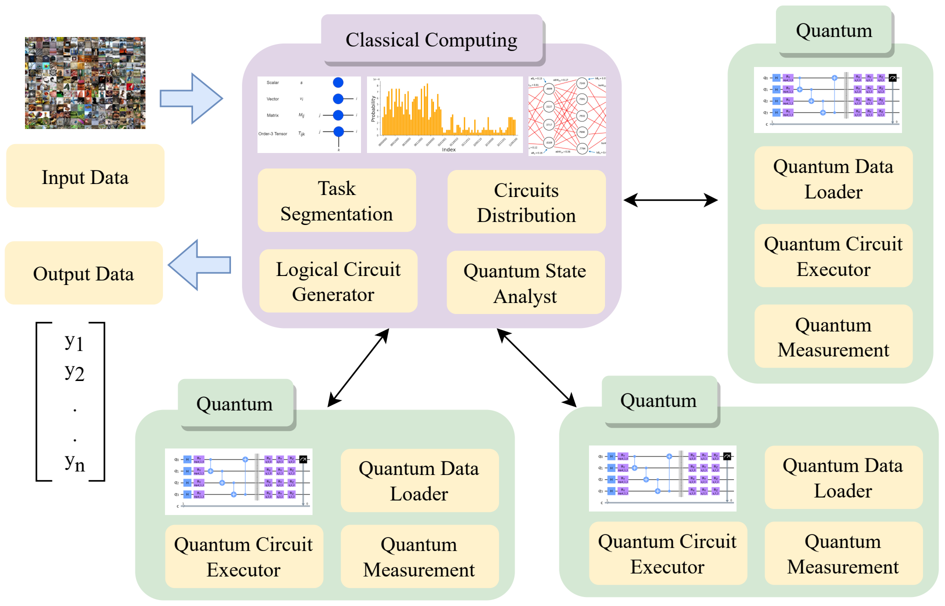


Fig. 3.1: System Architecture

The proposed system architecture functions as a closed feedback loop between a classical computer and a set of quantum computers, as illustrated in Figure 3.1. The data undergoes an initial cleaning process that includes the removal of significant outliers and other necessary data cleaning procedures. The cleaned classical data is then passed to related modules in the system.

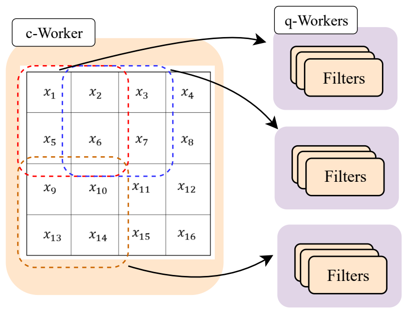


Fig. 3.2: Parallel Filters

The **Task Segmentation** is a key module in the system that aims to segment the large input into smaller pieces. Based on the predefined subtask unit, such as convolutional filter size, the Task Segmentation module decomposes the original data into multiple smaller sections in a continued fashion. The data points in the same section can be grouped and treated as individual subtasks. Figure 3.2 illustrates a conceptual example of a convolutional neural network (CNN) for image classification. An image on a classical computer is divided into smaller sections and there might be padding between the sections. While the concept of using a CNN to split up large pieces of data is not new, the concept of using it as the backbone of a distributed quantum system is what makes this technique unique.

The generated smaller data sections are transferred to the **Logical Circuit Generator** module. The main task in this module is to encode the classical data points from the CNN filters onto the quantum qubits. Different encoding methods can be adopted in this step and we utilize X and Y rotations to encode our data. After the data transformation, it generates a logic circuit for each section and passes the circuit to the next module. Due to limited quantum resources, the original image may be too large to be encoded on q-Workers, while sub-images are small enough. For our experiments, we used a filter stride of two, width of four and four for the number of filters. This allowed for us to have enough filters of appropriate size to send to one, two, and four workers.

The Circuit distribution module accepts logical circuits from the generator. It executes the scheduling algorithms to monitor a set of quantum computers and select the best host for each circuit. Once a quantum worker is selected, it distributes the circuit to the targeted worker for execution.

On a specific quantum worker, it employs three modules to interact with a classical computer. Firstly, the the **Quantum Data Loader** is used to load the logical circuits with initial data encoding. It maps logical qubits that are defined by the given circuits onto physical qubits on this specific quantum machine. Next, the **Quantum Circuit Executor** conducts the experiment to execute the gates on the circuits. Finally, the **Quantum Measurement** calculates the quantum fidelity from one anicilla qubit which is used to calculate model loss, and sends this metric back to the classical computer.

When the results loop back at classical computer, the **Quantum State Analyst** module actively analyzes the data based on the predefined cost function. The module uses this analysis to update the trainable parameters in the quantum circuit in attempts to minimize the cost function.

In our system, the decomposed data goes through smaller pieces on quantum Workers, and the results are transferred back to the classical computer for further processing. This hybrid quantum-classical design allows computation tasks to be distributed collaboratively. With data stored on multiple classical computers and processed on multiple quantum workers iteratively. Consequently, our system enables a quantum application to be distributed across multiple quantum and classical computers, working concurrently.

The training process of the system is summarized by Algorithm 1. First, the data is loaded as shown in Line 1. (**Disclaimer:** Different implementation libraries use different normalization methods. Some use [0,1] and some use [-1,1]. That is why some of the equations may lead to a value of -1) Line 2 represents the introduction of the training parameters set by the user at run time. The learning rate, α, indicates how large the updates to the system parameters should be during training. The network weights are initialized randomly. The number of epochs, ϵ, indicates how many times the network will be trained on the data set, X. The user then indicates the filter stride and width. The number of filters and number of workers are also initialized.

Line 4 is where the circuit bank that stores all of the circuits is initialized. Then, in line 5 the for loop of epochs begins. At the start of each epoch an epoch timer is started (Line 5). In each epoch, for the number of filters (Line 6), Lines 7-27 must be completed. For each data point in the X (Line 7), the data is encoded into a unitary matrix using $log_n encoding$ (Line 8). From there, a filter shape is determined by the width times the width is fed into the CNN. This data is then flattened and run through a classical dense layer (Line 10). Line 11 represents the equation for propagating through the classical dense layer.

Lines 12-23 represent the quantum backpropagation that occurs for each parameter θ. First, the output of the classical layer and all the trainable quantum circuit parameters θd are loaded (Lines 13-14). Then, one forward shifted (Lines 15-17) and one backward shifted (Lines 19-21) circuit are added to the circuit bank. The circuit bank is then divided by the number of workers to split it into w sections (Line 22). Each section is then assigned to a worker, prioritizing the least busy (Line 23). This job then sends circuits to its assigned worker, receiving job results back (Lines 24-26).

Lastly, the epoch timer is stopped (Line 27) and the time for that epoch is recorded (Line 28). The accuracy results are then retrieved and are used to calculate gradients and update the quantum parameters (Line 29).

Algorithm 1 DQuLearn Algorithm

```
1:  Data set Loading Dataset: $(X|Class : Mixed)$
2:  Parameter Initialization:
    Learning Rate : $\alpha = 10^{-4}$
    Network Weights : $\theta_d =$ [Rand Num between $0 - 1 \times \pi$]
    epochs : $\epsilon = 40$
    stride : s = int
    filter width : w = int
    number of filters : nF = int
    number of workers : w = int
    Dataset: $(X|Class = \omega)$

3:  Define circuit compiler with circuit bank cB
4:  **for** $\zeta \in \epsilon$ **do**
5:      Start epoch timer
6:      **for** $nF$ **do**
7:          **for** $x_k \in X$ **do**
8:              Encode data into unitary matrices $(log_n encoding)$
9:              Feed filter of size $(w * w)$ to CNN
10:             Flatten data and run through a classical dense layer
11:             $y(x) = W^{(N)T} h^{(N-1)} + b^{(N)}$
12:             **for** $\theta \in \theta_d$ **do**
13:                 Load $x_k \xrightarrow[\text{DataEncoding}]{\text{Quantum}} Q_{Q_1} \rightarrow Q_{count}$
14:                 Load $\theta_d \xrightarrow[\text{DataEncoding}]{\text{Quantum}} Q_{\frac{Q_{count}}{2}+1} + 1 \rightarrow \frac{Q_{count}}{2} + 1$
15:                 Add $\frac{\pi}{2} \rightarrow \theta$
16:                 $\Delta_{fwd} = (E_{Q_0} f(\theta_d))$
17:                 Add circuit to cB
18:                 AND
19:                 Subtract $\frac{\pi}{2} \rightarrow \theta$
20:                 $\Delta_{bck} = (E_{Q_0} f(\theta_d))$
21:                 Add circuit to cB
22:             $cB/w = jobs : j$
23:             Assign j to workers, prioritizing least busy
24:             **for** $j_k \in j$ **do**
25:                 Send $j_k$ to a worker
26:                 Receive results from worker
27:     Stop epoch timer
28: Record times per epoch
29: Record accuracy results per epoch
```

Chapter 4

EVALUATION

4.1 Accuracy

Python 3.9 and the IBM Qiskit Quantum Computing simulator package were used to implement our system. The circuits were trained on NSF Cloudlab M510 nodes housed at the University of Utah data center. Our experiment results are compared with multiple cutting-edge solutions listed below.

- PCA-QuClassi [43]: The predecessor of `DQuLearn` . It is a quantum-classical system that uses principal component analysis (PCA) to reduce the dimensions of the dataset to make it easier to train. In our evaluations, we use PCA-5, PCA-7 and PCA-17 to represent its 5-qubit, 7-qubit and 17-qubit settings.
- QuantumFlow [63] (QF-pNet): A co-design framework including quantum neural networks and the use of downsampling to reduce the dimensions of the data as well as an amplitude encoding method. The co-design framework, namely contains five

sub-components (QF-pNet, QF-hNet, QF-FB, QF-Circ, and QF-Map) and they work collaboratively to design neural networks and implement them to quantum computers. Based on three layer types (P-LYR, U-LYR, and N-LYR), QF-Nets contains two types of neural networks QF-pNet and QF-hNet, where QF-pNet applies P-LYR for neural computation while QF-hNet applies hybrid P-LYR and U-LYR. The QF-FB is designed to train the QF-Nets on classical computer. The trained QF-Nets can be deployed to quantum circuit QF-Circ for the inference, and then optimized by the QF-Map with the consideration of QF-Nets' property. Furthermore, QF-Map maps the virtual qubits to the physical qubits with the consideration of qubits' error rates.

- TensorFlow Quantum [50] (TFQ): The tutorial codes provided by the Tensorflow Quantum library that use Cirq circuits and standard layer designs. In this solution, once the data is preprocessed and downsized, it is encoded onto qubits that are then loaded into quantum circuits. A two-layer model is then built based on the data-circuit size and wrapped in a TFQ-Keras model. The model is fed the quantum data that encodes the classical data and uses a Parameterized Quantum Circuit layer, to train the model circuit on the quantum data.

- DNN-Fair [64]: A classical deep neural network (DNN) for training MNIST data usually has about 1.2M parameters. To provide a more fair comparison, we construct a DNN with 3145 parameters.

When comparing `DQuLearn` to the solutions above, the MNIST dataset was used. This is a frequently used dataset as a benchmark in the literature of quantum deep learning. The dataset contains hand-written digits with a resolution of 28x28, which means there are 784 dimensions in total. However, performing the data-coding technique on the full dimensions of the dataset is not feasible due to the lack of available qubits on accessible quantum computers and the computational complexity of the experiment. To combat this, the dimensionality of the dataset needs to be reduced. In our experiments, the dimensions were reduced to 4 for binary classification and 6 for multi-class classification. In addition to the MNIST dataset, the derived datasets of Fashion MNIST and Extended MNIST were used. The binary and multi-class experiments for these datasets were evaluated using simulators and some real IBM-Q quantum computers.

4.1.1 Quantum Binary Classification

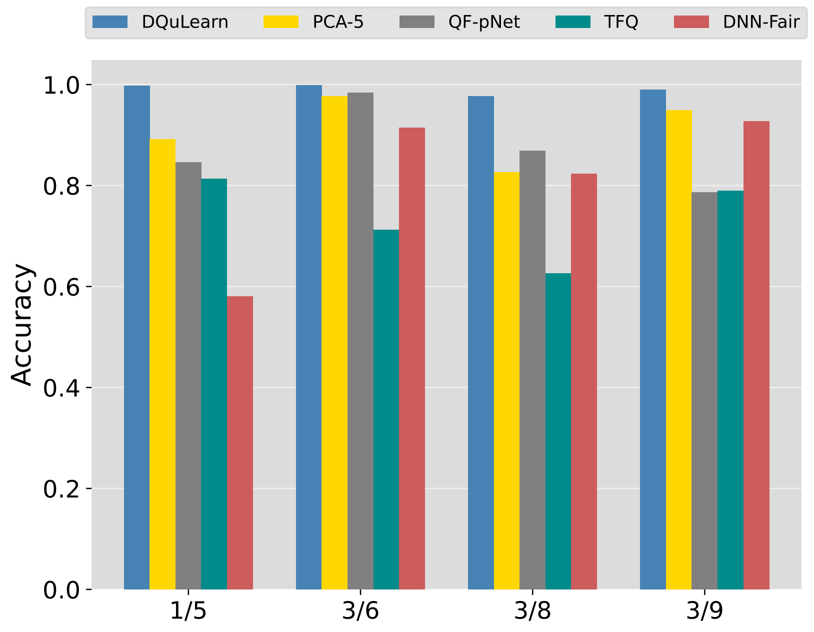


Fig. 4.1: Binary Classification on MNIST Data

For binary classification, the popular digit combinations of (1,5), (3,6), (3,8), and (3,9) were used. As seen in Figure 4.1, `DQuLearn` repeatedly outperforms the other solutions. The largest improvement over classical deep neural networks can be seen in the (1,5) experiment. `DQuLearn` was 41.72% better than DNN-Fair (3145 parameters) with a recorded accuracy of 99.79%. Even though the classical DNN can train perfect accuracies on the MNIST dataset, it requires a much larger parameter size than `DQuLearn`. By using five qubits, `DQuLearn` achieves a better or similar performance with 49.54% fewer parameters.

Moving onto the comparison to quantum-based solutions with the MNIST dataset, `DQuLearn` outperforms the alternatives. The largest margin of improvement can be seen with (3,8) and (3,9) classifications where there's an increase of 35.07% and 30.71% over Tensorflow Quantum and QF-pNet. `DQuLearn` also outperformed its predecessor, PCA-QuClassi when training MNIST.

4.1.2 Quantum Multi-Class Classification

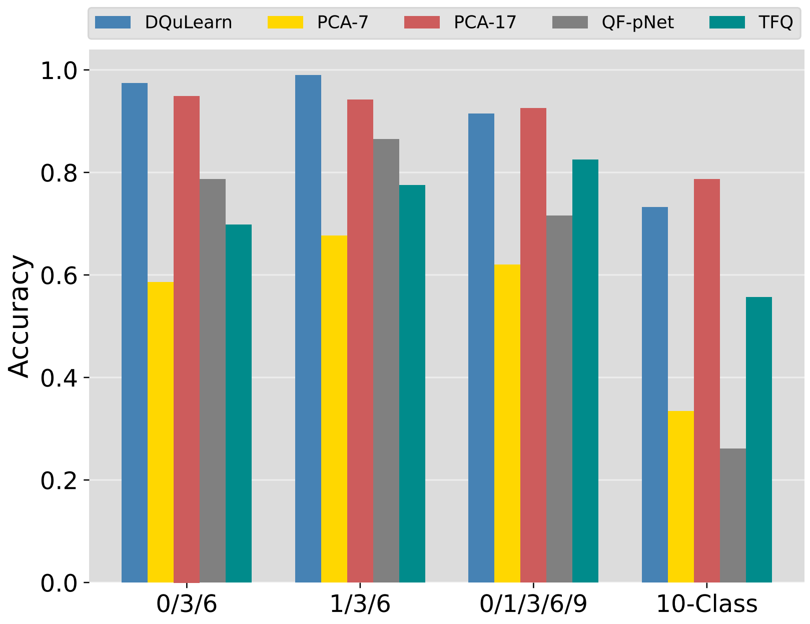


Fig. 4.2: Multi-Class Classification on MNIST Data

Next, `DQuLearn` is evaluated using multi-class classification with a 7-qubit setting. The digit combinations of (0,3,6), (1,3,6), (0,1,3,6,9), and (0,1,2,3,4,5,6,7,8,9) were used. As seen in Figure 4.2, `DQuLearn` greatly outperforms the alternatives. For the first three multi-class digit combinations, `DQuLearn` achieves 97.39%, 98.94%, and 91.48% accuracies compared to 58.55%, 67.68%, and 62.02% for the same 7-qubit setting for PCA-QuClassi. Therefore, `DQuLearn` is an improvement over PCA-QuClassi by up to 66.3% because of its use of the quantum-classical collaborative training architecture.

Compared to QF-pNet, `DQuLearn` performs better in all the experiments. An example of this is `DQuLearn` recording accuracies of 97.39% and 98.94% for (0,3,6) and (1,3,6) as opposed to 78.70% and 86.50% recorded by QF-pNet. This is an accuracy increase of 23.75% and 14.38%. The trend continues in 5-class classification where `DQuLearn` performs 19.92% better (91.48% vs 71.56%). Lastly, the biggest improvement can be seen in the 10-class experiments where `DQuLearn` outperforms QF-pNet by more than 181.90% (73.21% vs 25.97%). The difference between the two solutions is that in QF-pNet, most of the training is completed on a classical computer, and a traditional loss function is used. In `DQuLearn`, a quantum-based loss function is used so that the qubits can be fully utilized within the collaborative training architecture.

4.1.3 IBM-Q Platform Experiments

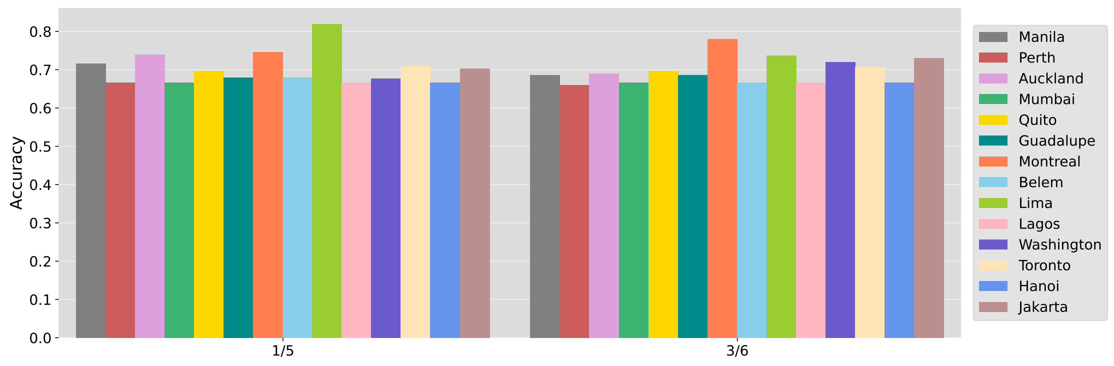


Fig. 4.3: (1,5) and (3,6) MNIST Binary Classifications on IBM-Q Quantum Computers

For further analysis, `DQuLearn` was evaluated on real quantum computers via the

IBM-Q platform. 300 data points from (1.5) and (3,6) MNIST classification were submitted to 14 of IBM-Q's superconducting quantum computers.

The circuits were generated from a network trained using `DQuLearn`. 300 circuits were submitted per machine in one job at 8192 shots each and the results can be seen in Figure 4.3. 8 of the 14 machines returned a 66.67% accuracy, which is the accuracy of an experiment that assumes all 0s. This is known as the ground state. The reasoning behind this is that variational parameters from a simulation may often perform poorly on real machines. This is the result of issues such as temporal drift and machine specific bias during induction issues [65]. The best results were achieved by IBMQ-Lima with an accuracy of 82.10%. This machine has a Quantum Volume of 8 which is one of the lowest among IBM machines. This reveals the complexity involved in predicting the machine performance of quantum routines and the effect that temporal drift has on learned parameters. The results could be improved going forward by optimizing the trained network on a local machine and then finalizing that training on the processor to account for machine-specific bias.

4.2 Runtime

Another piece of the project was to determine if distributing jobs over multiple quantum machines would lead to an improvement in the runtime of each epoch. The data was trained on the MNIST dataset and split into pieces by using filters. For the experiments below, the stride of the filter was set to two pixels, the width to four pixels and the number of filters was four. These jobs were then run on IBM-Q simulators. Experiments consisted of using five and seven qubits settings with one, two and three layers in each circuit, respectively. The runtime of these jobs was then compared when being run on one, two and four workers. **Disclaimer:** Experiments were run locally on a personal computer that is on the older side

(2015 Macbook Air). That is why the runtimes, in general, may seem longer than would be expected. The difference in the data between workers is what's most meaningful.

4.2.1 5-Qubit Setting Experiments

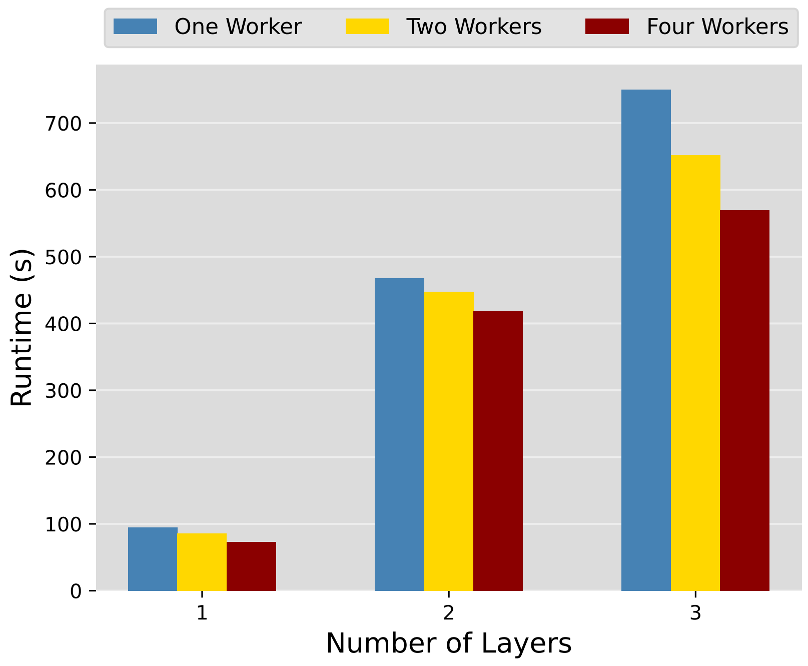


Fig. 4.4: 5-Qubit Experiment Runtimes

The data for five-qubit experiments on circuits with one, two, and three layers can be seen in Figure 4.4. For all layer counts, the trend is the same. As the number of workers increases, the runtime of the epoch decreases.

In the one layer experiments, the runtime starts at 94.71 seconds when running on one worker and improves to only 73.14 seconds when using four workers. When using two layers, the recorded runtime for using one simulator is 467.9 seconds and decreases to 418.63 seconds when using four workers. The biggest improvement in terms of the amount of seconds being eliminated is seen in the three layer experiments. When running on one worker the runtime is 749.75 seconds. The time then decreases to 651.71 seconds for two workers and 569.77 seconds for four workers. This is an improvement in runtime of 98.04 and 81.94 seconds, respectively.

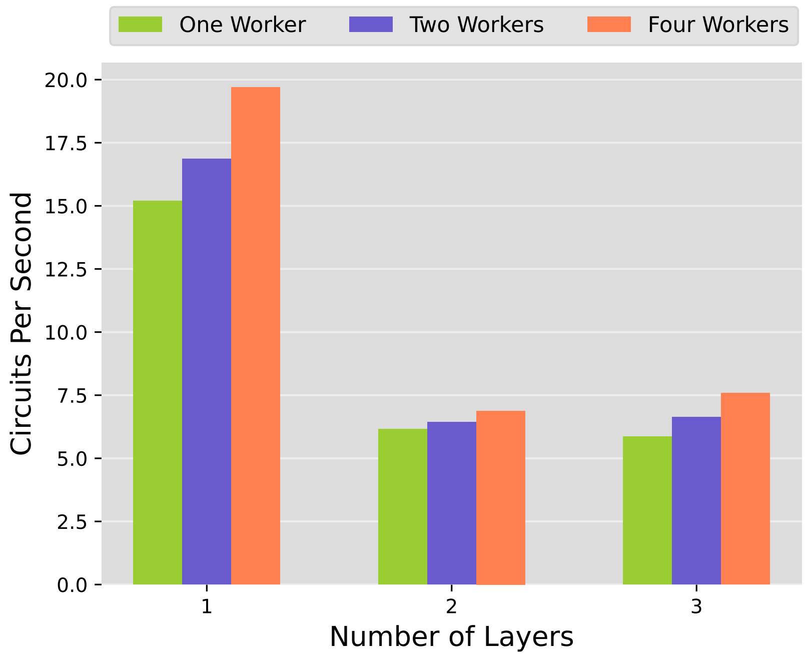


Fig. 4.5: 5-Qubit Experiment Circuits Per Second

Another form of evaluation is taking a look at the number of circuits being run per second based on the number of workers being used. For the various experiments, the number of circuits run are 1440 for one layer, 2880 for two layers, and 4320 for three layers. As seen in Figure 4.5, across all experiments, as the number of workers increases, the number of circuits simulated per second increases.

The one layer experiment displays the biggest increase as the number of circuits per second jumps from 15.2 when using one worker to 16.86 when using two workers. This number increases to 19.69 when using four workers. That's an overall increase of 4.49 circuits per second when using four workers. The one layer circuits are the least complex circuits used in the experiments so it makes sense that many of them can be simulated per second and the amount per second would increase significantly as the number of workers increases.

For the more complex circuits of two and three layers, the improvement is not as drastic as the one layer experiments, but there is still a steady improvement as the number of workers is increased. During the two layer experiments, the number of circuits per second increases from 6.16 when using one worker to 6.44 when using two workers. This number increases to 6.63 when using four workers. Similar improvement can be seen with the three layer experiments as the number of circuits per second increased from 5.86 with one worker to 6.63 with two workers to 7.58 with four workers. That's an overall increase of 1.72 circuits per second.

# of Layers	1 to 2	2 to 4	1 to 4
1	9.84%	14.35%	22.78%
2	4.38%	6.43%	10.53%
3	13.08%	12.57%	24.01%

Table 4.1: 5-Qubit Setting Percentage Time Decrease

Above is the percentage decrease in time for all five-qubit experiments. In the table, "1 to 2," for example, means the percentage decrease in runtime when moving from one worker to two workers. Here, the largest percentage decrease in seconds between one experiment setting occurred in the one layer experiments jumping from two simulators to four simulators at 14.35%. That means that the jobs took 14.35% less time when running on 4 simulators as opposed to two simulators.

Following the same pattern as the seconds data, the three layer experiments have the largest percentage decrease from one simulator to four simulators at 24.01%. However, it is the one layer experiments that had the next highest decrease at 22.78%. Overall, the experiments saw an improvement of 10.53% or better which is a significant decrease in time spent.

4.2.2 7-Qubit Setting Experiments

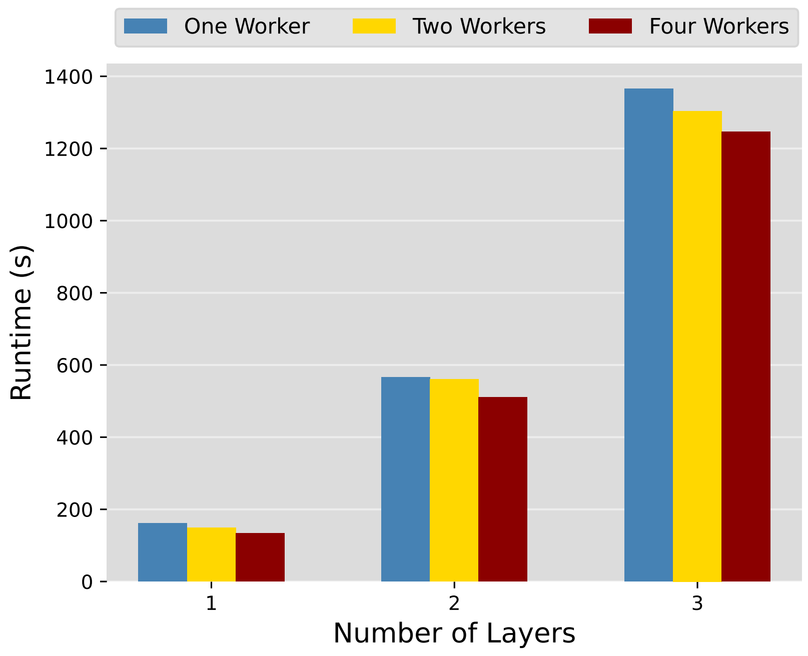


Fig. 4.6: 7-Qubit Experiment Runtimes

The data for seven-qubit experiments on circuits with one, two, and three layers can be seen in Figure 4.6. The results are in line with those of the five-qubit experiments. Each time, more workers were used, the less time it would take for each epoch to run.

In the one layer experiments, the runtime starts at 163.03 seconds when running on one worker and improves to just 134.31 seconds when using four workers. When using two circuit layers, the recorded runtime for using one worker is 566.46 seconds and decreases to 510.78 seconds when using four workers. The biggest improvement for number of seconds removed once again is seen in the three layer experiments. When running on one worker, the runtime is 1366.1 seconds. The time then decreases to 1303.94 seconds for two workers and

1246.49 seconds for four workers. This is an improvement in runtime of 62.16 and 57.45 seconds, respectively.

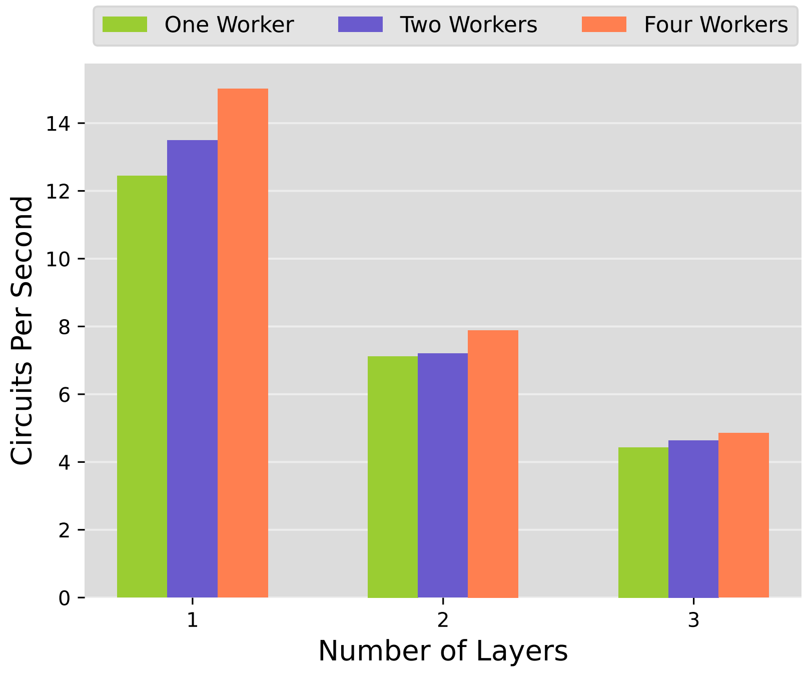


Fig. 4.7: 7-Qubit Experiment Circuits Per Second

For the seven-qubit experiments, the number of circuits run are 2016 for one layer, 4032 for two layers, and 6048 for three layers. As seen in Figure 4.7, the same trend as the 5-qubit experiments occurs. Across all experiments, as the number of workers increases, the number of circuits simulated per second increases.

The one layer experiment once again has the largest increase as the number of circuits per second jumps from 12.44 when using one worker to 13.49 when using two workers. This number then increases to 15.01 when using four workers. This is an overall increase of 2.57 circuits per second.

During the two layer experiments, the number of circuits per second slightly increases from 7.12 when using one worker to 7.20 when using two workers. This number increases to 7.89 when using four workers. Improvement can also be seen with the three layer experiments as the number of circuits per second increased from 4.43 with one worker to 4.64 with two workers to 4.85 with four workers.

# of Layers	1 to 2	2 to 4	1 to 4
1	7.76%	10.14%	17.11%
2	1.08%	8.84%	9.83%
3	4.55%	4.41%	8.76%

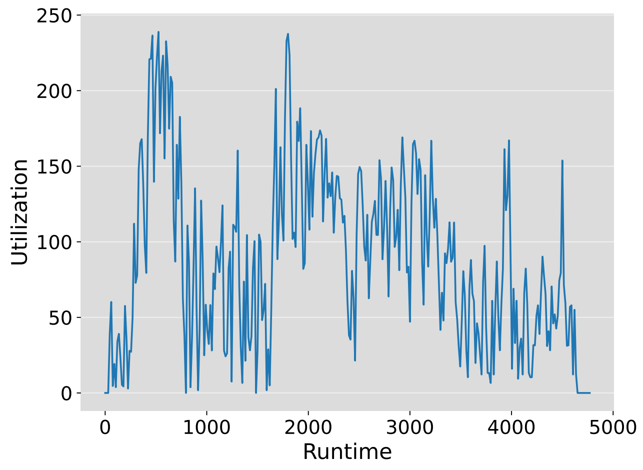
Table 4.2: 7-Qubit Setting Percentage Time Decrease

Above is the percentage decrease in time for all seven-qubit experiments. Here, the largest percentage decrease in seconds between one experiment setting again occurred in the one layer experiments jumping from two workers to four workers at a 10.14% decrease in runtime.

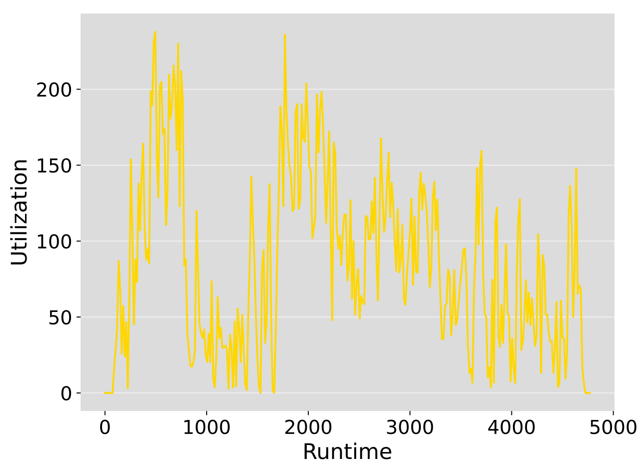
The data for the seven-qubit experiments differs from the five-qubit experiments. It is the one layer data that has the largest percentage seconds decrease at 17.11%. The two layer experiments have the next highest decrease, followed by the three layer experiments. Overall, the experiments saw an improvement of 8.76% less runtime or better.

4.3 Utilization

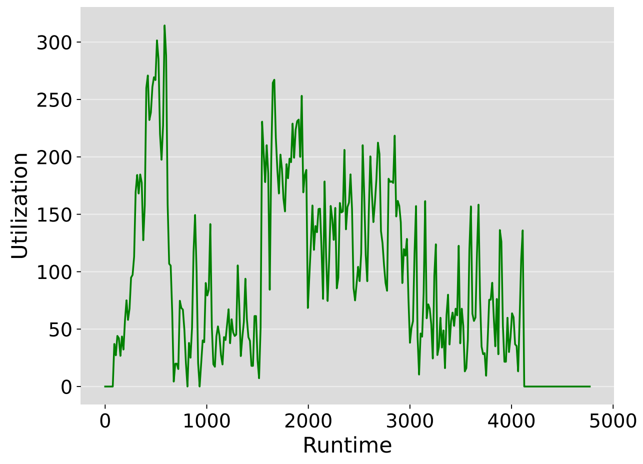
The last part of our research was monitoring the utilization to ensure an equal distribution in the workload of the workers. Experiments for all 5-qubit and 7-qubit settings were run simultaneously (6 experiments in total) on four workers. The results for the utilization of the four workers can be seen in Figure 4.8. The workload is consistent among the four workers. Therefore, the algorithm achieved its goal of using our quantum resources as efficiently as possible. None of the workers are ever sitting idle while others do the work.



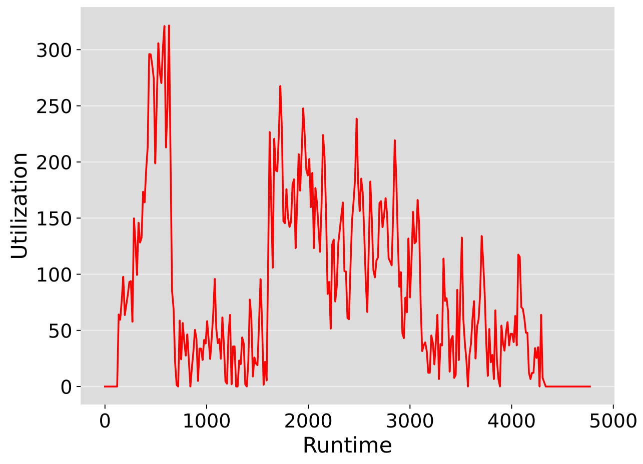
(a) Worker 1



(b) Worker 2



(c) Worker 3



(d) Worker 4

Fig. 4.8: Utilization of the Four Workers

CONCLUSION

In this thesis, `DQuLearn`, a distributed quantum computing system, is proposed. The system takes the data and breaks it up into chunks called filters through the use of a convolutional neural network. These pieces of data are then distributed to various workers in parallel to speed up the runtime of performing quantum experiments. While CNNs are a common tool used to tackle large chunks of data, the use of a CNN to help coordinate a distributed quantum system is something that is not as common. Using a CNN allows us to create a distributed workload among our workers in parallel which leads to more efficient runtimes.

As seen in the results, a distributed architecture does lead to a lower overall runtime for the jobs. `DQuLearn`, when using four workers, performs up to 24.01% better than one worker. This lower runtime allows for a more efficient use of quantum resources which are scarce to the public.

`DQuLearn` is a notable advancement in the field of quantum computing, but there is definitely more work to be done. While 24.01% less runtime is impressive, not all experiments had as strong of a result. Future work would focus on getting those numbers up across the board, possibly approaching a 50% decrease.

Future research will concentrate on expanding our experiments to include more workers in order to fully utilize publicly available quantum resources. It will also involve creating a better scheduling algorithm so that circuits are being sent to workers in the most efficient way possible.

Bibliography

[1] R. P. Feynman, “Simulating physics with computers,” *International journal of theoretical physics*, vol. 21, no. 6/7, pp. 467–488, 1982.

[2] `https://quantum-computing.ibm.com/`.

[3] F. Arute, K. Arya, R. Babbush, D. Bacon, *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, 505–510, 2019. [Online]. Available: `https://www.nature.com/articles/s41586-019-1666-5`.

[4] P. Kaye, R. Laflamme, and M. M. al., “An introduction to quantum computing,” *Oxford university press*, 2007.

[5] C. C. Aggarwal *et al.*, “Neural networks and deep learning,” *Springer*, vol. 10, no. 978, p. 3, 2018.

[6] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain, “Machine learning for resource management in cellular and iot networks: Potentials, current solutions, and open challenges,” *IEEE communications surveys & tutorials*, vol. 22, no. 2, pp. 1251–1275, 2020.

[7] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, “Flowcon: Elastic flow configuration for containerized deep learning applications,” in *ACM the 48th International Conference on Parallel Processing (ICPP ’19)*, 2019.

[8] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, “Deep learning for hybrid 5g services in mobile edge computing systems: Learn from a digital twin,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4692–4707, 2019.

[9] W. Zheng, Y. Song, Z. Guo, Y. Cui, S. Gu, Y. Mao, and L. Cheng, “Target-based resource allocation for deep learning applications in a multi-tenancy system,” in *2019 IEEE High Performance Extreme Computing Conference(HPEC ’19)*, 2019.

[10] J. Ren, G. Yu, and G. Ding, “Accelerating dnn training in wireless federated edge learning systems,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 219–232, 2020.

[11] Y. Mao, V. Green, J. Wang, H. Xiong, and Z. Guo, “Dress: Dynamic resource-reservation scheme for congested data-intensive computing platforms,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, 2018, pp. 694–701.

[12] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, “Deep-learning-based joint resource scheduling algorithms for hybrid mec networks,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6252–6265, 2019.

[13] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, “Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster,” in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2017, pp. 1–8.

[14] Y. Mao, Y. Fu, S. Gu, S. Vhaduri, L. Cheng, and Q. Liu, "Resource management schemes for cloud-native platforms with computing containers of docker and kubernetes," *arXiv preprint arXiv:2010.10350*, 2020.

[15] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

[16] Y. Mao, Y. Fu, W. Zheng, L. Cheng, Q. Liu, and D. Tao, "Speculative container scheduling for deep learning applications in a kubernetes cluster," *IEEE Systems Journal*, vol. 16, no. 3, pp. 3770–3781, 2021.

[17] K. I. Ahmed, H. Tabassum, and E. Hossain, "Deep learning for radio resource allocation in multi-cell networks," *IEEE Network*, vol. 33, no. 6, pp. 188–195, 2019.

[18] Y. Mao, V. Sharma, W. Zheng, L. Cheng, Q. Guan, and A. Li, "Elastic resource management for deep learning applications in a container cluster," *IEEE Transactions on Cloud Computing*, 2022.

[19] Y. Mao, W. Yan, Y. Song, Y. Zeng, M. Chen, L. Cheng, and Q. Liu, "Differentiate quality of experience scheduling for deep learning inferences with docker containers in the cloud," *IEEE Transactions on Cloud Computing*, 2022.

[20] M. Maier, D. Elsner, C. Marouane, M. Zehnle, and C. Fuchs, "Deepflow: Detecting optimal user experience from physiological data using deep neural networks.," in *AAMAS*, 2019, pp. 2108–2110.

[21] K. Holstein, J. Wortman Vaughan, H. Daumé III, M. Dudik, and H. Wallach, "Improving fairness in machine learning systems: What do industry practitioners need?" In *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–16.

[22] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*, IEEE, 2019, pp. 331–344.

[23] Y. Fu, S. Zhang, J. Terrero, Y. Mao, G. Liu, S. Li, and D. Tao, "Progress-based container scheduling for short-lived applications in a kubernetes cluster," in *2019 IEEE International Conference on Big Data (BigData'19)*, 2019.

[24] X. Chen, F. Cheng, C. Liu, L. Cheng, and Y. Mao, "An improved wolf pack algorithm for optimization problems: Design and evaluation," *Plos one*, vol. 16, no. 8, e0254239, 2021.

[25] L. Cheng, Y. Wang, Q. Liu, D. H. Epema, C. Liu, Y. Mao, and J. Murphy, "Network-aware locality scheduling for distributed data operators in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1494–1510, 2021.

[26] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 924–935, 2019.

[27] Q. Liu, T. Xia, L. Cheng, M. Van Eijk, T. Ozcelebi, and Y. Mao, "Deep reinforcement learning for load-balancing aware network control in iot edge systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1491–1502, 2021.

[28] L. A. Barroso, U. Hölzle, and P. Ranganathan, *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019.

[29] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A woa-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems journal*, vol. 14, no. 3, pp. 3117–3128, 2020.

[30] A. Yang, J. Wang, Y. Mao, Y. Yao, N. Mi, and B. Sheng, "Optimizing internal overlaps by self-adjusting resource allocation in multi-stage computing systems," *IEEE Access*, vol. 9, pp. 88 805–88 819, 2021.

[31] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 2002–2013, 2019.

[32] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Physical Review A*, vol. 103, no. 3, p. 032 430, 2021.

[33] S. L. Wu and S. Yoo, "Challenges and opportunities in quantum machine learning for high-energy physics," *Nature Reviews Physics*, pp. 1–2, 2022.

[34] R. V. Casaña-Eslava, P. J. Lisboa, S. Ortega-Martorell, I. H. Jarman, and J. D. Martín-Guerrero, "Probabilistic quantum clustering," *Knowledge-Based Systems*, p. 105 567, 2020.

[35] D. Chen, Y. Xu, B. Baheri, S. A. Stein, C. Bi, Y. Mao, Q. Quan, and S. Xu, "Quantum-inspired classical algorithm for slow feature analysis," *arXiv preprint arXiv:2012.00824*, 2020.

[36] B. Baheri, Z. Xu, V. Chaudhary, Y. Mao, B. Fang, S. Xu, and Q. Guan, "Pinpointing the system reliability degradation in nisq machines," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2022, pp. 646–652.

[37] L. Xue, L. Cheng, Y. Li, and Y. Mao, “Quantum machine learning for electricity theft detection: An initial investigation,” in *2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, IEEE, 2021, pp. 204–208.

[38] S. Y.-C. Chen, C.-M. Huang, C.-W. Hsing, and Y.-J. Kao, “An end-to-end trainable hybrid classical-quantum classifier,” *Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045 021, 2021.

[39] S. A. Stein, R. L’Abbate, W. Mu, Y. Liu, B. Baheri, Y. Mao, G. Qiang, A. Li, and B. Fang, “A hybrid system for learning classical data in quantum states,” in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, IEEE, 2021, pp. 1–7.

[40] C. Zoufal, A. Lucchi, and S. Woerner, “Quantum generative adversarial networks for learning and loading random distributions,” *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[41] J. Li, R. O. Topaloglu, and S. Ghosh, “Quantum generative models for small molecule drug discovery,” *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–8, 2021.

[42] D. Chen, Y. Xu, B. Baheri, C. Bi, Y. Mao, Q. Quan, and S. Xu, “Quantum-inspired classical algorithm for principal component regression,” *arXiv preprint arXiv:2010.08626*, 2020.

[43] S. A. Stein, B. Baheri, D. Chen, Y. Mao, Q. Guan, A. Li, S. Xu, and C. Ding, “Quclassi: A hybrid deep neural network architecture based on quantum state fidelity,” *Proceedings of Machine Learning and Systems*, vol. 4, pp. 251–264, 2022.

[44] S. A. Stein, B. Baheri, D. Chen, Y. Mao, Q. Guan, A. Li, B. Fang, and S. Xu, "Qugan: A quantum state fidelity based generative adversarial network," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2021, pp. 71–81.

[45] S. Ruan, Y. Wang, W. Jiang, Y. Mao, and Q. Guan, "Vacsen: A visualization approach for noise awareness in quantum computing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 462–472, 2023. DOI: `10.1109/TVCG.2022.3209455`.

[46] W. Mu, Y. Mao, L. Cheng, Q. Wang, W. Jiang, and P.-Y. Chen, "Iterative qubits management for quantum index searching in a hybrid system," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, IEEE, 2022, pp. 283–289.

[47] S. Stein, Y. Mao, J. Ang, and A. Li, "Qucnn: A quantum convolutional neural network with entanglement based backpropagation," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, IEEE, 2022, pp. 368–374.

[48] S. Ruan, R. Yuan, Y. Wang, Y. Lin, Y. Mao, W. Jiang, Z. Wang, W. Xu, and Q. Guan, "Venus: A geometrical representation for quantum state visualization," *arXiv preprint arXiv:2303.08366*, 2023.

[49] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen, *et al.*, "Qiskit: An open-source framework for quantum computing," *Accessed on: Mar*, vol. 16, 2019.

[50] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, *et al.*, "Tensorflow quantum: A software framework for quantum machine learning," *arXiv preprint arXiv:2003.02989*, 2020.

[51] B. Bichsel, M. Baader, T. Gehr, and M. Vechev, "Silq: A high-level quantum language with safe uncomputation and intuitive semantics," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 286–300.

[52] D. S. Steiger, T. Häner, and M. Troyer, "Projectq: An open source software framework for quantum computing," *Quantum*, vol. 2, p. 49, 2018.

[53] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, "Q# enabling scalable quantum computing and development with a high-level dsl," in *Proceedings of the real world domain specific languages workshop 2018*, 2018, pp. 1–10.

[54] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, vol. 549, no. 7671, pp. 180–187, 2017.

[55] J. Liu, L. Bello, and H. Zhou, "Relaxed peephole optimization: A novel compiler optimization for quantum circuits," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2021, pp. 301–314.

[56] L. Gyongyosi and S. Imre, "Scalable distributed gate-model quantum computers," *Scientific reports*, vol. 11, no. 1, pp. 1–28, 2021.

[57] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti, "Towards a distributed quantum computing ecosystem," *IET Quantum Communication*, vol. 1, no. 1, pp. 3–8, 2020.

[58] Y. Feng, S. Li, and M. Ying, "Verification of distributed quantum programs," *ACM Transactions on Computational Logic (TOCL)*, vol. 23, no. 3, pp. 1–40, 2022.

[59] T. Häner, D. S. Steiger, T. Hoefler, and M. Troyer, "Distributed quantum computing with qmpi," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.

[60] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. Sawaya, “Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits,” *Quantum Science and Technology*, vol. 5, no. 3, p. 034 007, 2020.

[61] G. G. Guerreschi, “Fast simulation of quantum algorithms using circuit optimization,” *Quantum*, vol. 6, p. 706, 2022.

[62] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`, [Online; accessed 20-April-2023].

[63] W. Jiang, J. Xiong, and Y. Shi, “A co-design framework of neural networks and quantum circuits towards quantum advantage,” *Nature communications*, vol. 12, no. 1, p. 579, 2021.

[64] *Tensorflow Quantum Fair Comparison*, `https://www.tensorflow.org/quantum/tutorials/mnist`, [Online; accessed 07-March-2023].

[65] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, “Eqc: Ensembled quantum computing for variational quantum algorithms,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 59–71.

Anthony D'Onofrio Jr.

M.S., Fordham University

Distributed Learning in a Quantum-Classical Hybrid System

Dissertation directed by Ying Mao, Ph.D.

Machine learning and deep learning have made significant advances in recent years, transforming the way we approach and solve complex problems. However, the viability of machine learning and deep learning on classical computers has began to come into question as the need for complex problem solving has increased since classical computers are limited in terms of their processsing power.

Quantum computing has emerged as a new form of computing that uses quantum mechanics to process and manipulate information. Quantum computers use quantum bits, or qubits, which can exist in multiple states simultaneously. This property allows quantum computers to perform computations more efficiently than classical computers.

However, publicly available quantum resources are limited to five or seven qubits per machine which limits the complexity of the problems that can be solved. A potential solution to resource limitation challenges is the use of a distributed system which is a network of interconnected computers that work together to achieve a common goal.

In this thesis, a distributed quantum system is introduced. Our system allows for the efficient management of qubits on individual machines while expanding the qubits available by managing a distributed system of quantum computers. This solution will lead to the ability to solve increasingly complex problems while maximizing the use of limited quantum resources.

VITA

Anthony D'Onofrio Jr., son of Anthony and Toni Ann, was born on April 23, 2001, in New York, New York. After graduating in 2019 from Ridgefield High School in Ridgefield, CT, he entered Fordham University. In 2022, he received the Bachelor of Science degree in Computer Science.

He then continued at Fordham University through their Accelerated Master's Program in 2022, and earned his Master of Science degree in Computer Science in 2023. This ended a period of four years where he completed both his Bachelor's and Master's degrees. During his time at Fordham, he was inducted into Sigma Xi, The Scientific Research Honor Society, as an associate member for his research on quantum computing under the mentorship of Dr. Ying Mao.