3-2021

Performance of Various Low-level Decoder for Surface Codes in the Presence of Measurement Error

Claire E. Badger

Performance of Various Low-Level Decoders for Surface Codes in the Presence of Measurement Error

THESIS

Claire E Badger, B.S.C.S., 2d Lieutenant, USAF

AFIT-ENG-MS-21-M-008

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-008

Performance of Various Low-Level Decoders for Surface Codes in the Presence of Measurement Error

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Claire E Badger, B.S.C.S., B.S.E.E.

2d Lieutenant, USAF

March 26, 2021

AFIT-ENG-MS-21-M-008

Performance of Various Low-Level Decoders for Surface Codes in the Presence of Measurement Error

THESIS

Claire E Badger, B.S.C.S., B.S.E.E.
2d Lieutenant, USAF

Committee Membership:

Laurence D. Merkle, Ph.D
Chair

Brett J. Borghetti, Ph.D
Member

David E. Weeks, Ph.D
Member

Abstract

Quantum error correction is a research speciality within the area of quantum computing that constructs quantum circuits that correct for errors. Decoding is the process of using measurements from an error correcting code, known as error syndrome, to decide corrective operations to perform on the circuit. High-level decoding is the process of using the error syndrome to perform corrective logical operations, while low-level decoding uses the error syndrome to correct individual data qubits. Research on machine learning-based decoders is increasingly popular, but has not been thoroughly researched for low-level decoders. The type of error correcting code used is called surface code. A neural network-based decoder is developed and compared to a partial lookup table decoder and a graph algorithm-based decoder. The effects of increasing error correcting code size and increasing measurement errors on the error syndrome are analyzed for the decoders. The results demonstrate that there are advantages in terms of average execution time and resistance to increasing measurement error with the neural network-based decoder when compared to the two other decoders.

Table of Contents

Page

Page

Page

List of Figures

List of Tables

Performance of Various Low-Level Decoders for Surface Codes in the Presence of Measurement Error

I. Introduction

1.1 Motivation

Quantum computing is a new and increasingly important area of research that combines computer science, mathematics, and physics [55]. Quantum mechanics and the use of quantum information have the potential to use new computational abilities to create efficient algorithms for problems that are computationally difficult for classical computers. Quantum computing research has expanded significantly in the last few decades due to breakthroughs in quantum algorithms, improvements in quantum hardware, and developments in quantum computing software stacks, such as IBM's Qiskit [4]. Among other areas, full-scale quantum computing could radically change communications, cryptography and security, complex simulations [21], and algorithms that handle big data [20, 25].

Although the implications of full-scale quantum computation are numerous and highly important to technology and science professionals across the globe, getting to that point is incredibly difficult. Current quantum computers have high error rates and cannot perform accurate computations on anything but small, simple quantum circuits. Quantum hardware needs to improve drastically before theoretical quantum algorithms can be executed. The hardware needs to be improved, but another way to make practical quantum computation possible is to use error correcting codes to perform quantum computation. In classical computation, if there is a certain

probability of error that a classical bit can flip erroneously then one bit can be encoded with many of the same bits with repetition. However, in quantum computation such a simple solution cannot be implemented. This is due to quantum's No Cloning Theorem, which states that an arbitrary quantum state cannot be copied without altering the original [2]. Although the No Cloning Theorem is a unique property that can lead to innovations in secure communication methods, it also makes the process of implementing error correcting codes on quantum systems much more complicated. However, the research into quantum error correction is one of the main reasons that scientists believe that more exploration in quantum computing will be rewarding [42].

Error correcting codes have been an important part of the research in quantum computing. One particularly active area of research is in topological error correcting codes. These types of codes have a repeated geometry and configuration that is useful to error correction [32]. The qubits of surface code are constructed to encode logical qubits. The states of these logical qubits are changed by performing logical operations, which changes whole groups of qubits on the surface code to alter the logical state. This is expanded in Section 2.3.3. The type of topological code used in this research is called the surface code, which uses a geometry similar to a grid.

In error correction, decoding is the process of using measurements from these codes to select corrective operations to execute on the circuit [50]. Decoders are categorized as either high-level or low-level. High-level decoders perform logical corrective operations, changing a whole row or column of qubits on surface codes using one of four operators: the logical X, Z, Y, and identity operators. In contrast, low-level decoders work to correct individual faulty qubits on surface codes. There has been far more research in high-level decoders since the output for any classification algorithm would only be those four operators regardless of surface code size. Existing research in low-level decoding is more limited.

This research analyzes deep neural network-based low-level decoder performance, and is the first such effort to do so while considering varying rates of measurements error. Although the logical error rate of various decoders has been compared based on increasing error on both measurement and data qubits, there have not yet been experiments to find decoders that are more resilient to measurement errors specifically. This is an important motivation and contribution of this research.

This is also a good opportunity to demonstrate the strengths and weaknesses of various types of decoders while there is a need for development in a commonly used Python package called Qiskit. This software allows users to access IBM's quantum computers and simulators through the Qiskit package so that more people can be exposed to quantum computing. The `topological_codes` Qiskit module is configured to run error correcting codes such as the surface code. This module is currently limited in its applicability because it has not been developed as much as other modules [4]. However, like many packages in Qiskit, it is currently being updated and developed. This includes diversifying the decoders included in the module. More research and support in neural network decoding would support updating this module to have machine learning-based decoders, which would give researchers, amateurs, students, and all Qiskit users exposure to the tradeoffs between various decoders and further an overall understanding of decoders that would be best to use in practical topological codes.

1.2 Research Objectives

The objective of this research is to analyze the strengths and weaknesses of machine learning-based decoders for low-level decoding of the surface code. The following research questions guide the effort to investigate this primary research objective:

- What tradeoffs exist between the use of neural network-based low-level decoders

compared to low-level decoders based on graph algorithms?

- How do various factors such as code depth and measurement noise affect decoding performance for neural network-based decoders and decoders based on graph algorithms?

The first research question specifically aims to address neural network-based low-level decoding. The low-level decoders evaluated in the existing literature are compared to high-level decoders, rather than evaluating how well low-level decoding works when just compared to graph algorithm-based decoders. There is evidence that high-level decoders based on neural networks offer distinct advantages in their flexibility and execution time [50] when compared to graph algorithm-based decoders. Little research has been done that specifically analyzes the extent to which low-level neural network decoders offer the same benefits as high-level decoders.

The hypotheses regarding this research question are:

- **Hypothesis 1:** The effectiveness of predicting corrective operations of neural network-based decoders decreases more slowly than that of algorithmic decoders with increasing noise levels on measurements.

- **Hypothesis 2:** Neural network-based decoders have execution times that increase more slowly by a constant factor than algorithmic decoders with increasing code depths.

The second research question aims to address how noise models more complicated than the commonly used depolarizing noise model impact the predictive performance of the decoders tested. Realistic noise affects the data qubits of an error correcting code, but it also impacts measurements, corrective operations, and even entanglement operations used to create the structure of the code. An error correcting code and decoding scheme that can maintain all realistic types of error under a certain threshold

to protect the logical qubit is known as fault tolerant. This research investigates how measurement errors, also called read out errors, can affect how accurately a decoder performs. This gives more information about how well decoders can do under more realistic noise, which can support more research in those decoding schemes that show more promise to be fault tolerant.

The hypotheses regarding this research question are:

- **Hypothesis 3:** A neural network-based decoder can predict corrective operations on surface codes of depth 3, 5, and 7 better than a partial lookup table at 0, 1, 3, and 5% probability of measurement error.

- **Hypothesis 4:** For graph algorithm decoders and neural network-based decoders, increasing code depth does not change effectiveness of predicting corrective operations.

- **Hypothesis 5:** Increasing measurement noise levels decreases effectiveness of predicting corrective operations in all decoders.

These research questions are important to guiding this investigation in low-level decoding. There has not yet been an analysis in low-level neural network decoding besides comparing it to a high-level decoder. Analyzing various low-level decoders compared to each other, rather than a high-level decoder, adds to the body of knowledge about the characteristics of low-level decoding as a stand alone method. The research questions intend to achieve this goal.

1.3 Assumptions and Limitations

This research makes several assumptions and is subject to several limitations. The dataset used for training machine learning algorithms is simulated since current

quantum hardware does not exist that can execute the surface code. This necessary simulation is the root cause of the main assumptions and limitations of this research.

One assumption of this research is the noise model used for simulation. Noise models are generally much simpler than noise observed on quantum computers. However, even simple noise models provide a way to analyze how well error correcting codes and decoders generally work [50]. The noise model used in this research assumes the errors on qubits are independent, with equal probability of the various types of errors to happen on a single qubit, whereas in actual quantum circuits the errors on various qubits may be correlated and may occur with different frequencies. The details of noise models and the depolarizing noise models used for data simulation are fully explained in Section 2.3.1.

One limitation of this research is the computation time required to simulate error syndromes and data qubit errors with increasing code depths and data qubit errors. The number of combinations of physical qubit errors that can occur in an error correcting circuit scales exponentially with the number of individual data qubit errors possible. This in turn is proportional to the number of data qubits and therefore to the square of the code depth. The available computation time did not permit the simulation of errors for large code depths, nor for the full set of possible combinations of error for the code depths examined. Thus, the scope of the research is limited to small code depths with a fixed maximum number of data qubit errors. As such, caution should be used in generalizing the results of this research to larger code depths and arbitrary noise levels.

1.4 Summary and Document Overview

In this Chapter, Section 1.1 outlines the motivation behind this thesis. Section 1.2 states the research questions and hypotheses on which the experiments of this thesis

are based. Lastly, Section 1.3 describes the assumptions underlying this research and limitations on its scope.

In the remainder of this document, Chapter II presents relevant background on the surface code and machine learning concepts used in this thesis, as well as a summary of previous work on machine learning techniques used for surface code decoding. Chapter III defines the methodology used to implement low level decoders and the experiments designed to evaluate the effectiveness and efficiency of those decoders. Finally, Chapter V draws conclusions from those experiments, summarizes the contributions of this research, and offers some areas of future work.

II. Background and Literature Review

2.1 Overview

This chapter introduces the fundamentals of quantum computation used for error correction, quantum error correction itself, and the fundamentals of decoding surface codes. Section 2.2 gives an overview of what makes quantum computation different from classical computation and why that is important. Section Section 2.3 discusses quantum bits, also known as qubits, and types of operations used on qubits. Section 2.3 also presents error correcting codes, specifically a well known type of topological code called the surface code. Section 2.4 overviews decoding and types of decoders, which is followed by Section 2.5 discussing background knowledge needed to understand the machine learning techniques used in this thesis. Section 2.6 describes the statistical significance tests considered for this research. Previous work that is pertinent to this research is summarized in Section 2.7 as well. Finally, Section 2.8 summarizes the chapter.

2.2 Quantum Computation

Quantum computing has been increasingly a subject of research over the past several decades for many reasons, one of them being the potentially incredible computational power that is possible through this new computing paradigm. At a basic level, both quantum computers and classical computers solve problems, but quantum computers manipulate data differently and are fundamentally governed by the rules of quantum mechanics [20]. One of the reasons quantum computing is very different than classical computing because quantum theory is cast in the language of complex vector spaces [55]. Quantum computing is also subject to some unique rules, such as the No Cloning Theorem, which is mentioned in Section 1.1, as well as entanglement

and superposition, which are explained in Section 2.2.1 and Section 2.2.2. These unique features and complex states make quantum computing notoriously difficult, but very powerful given reliable hardware.

Quantum hardware is still in its infancy, which makes quantum computers subject to high levels of noise. This noise makes the quantum algorithms that can solve useful algorithms impossible to execute on the quantum computers that exist today. Noise occurs at all stages of quantum computation, whether it be on operations, measurements, or qubit states during circuit execution. This makes quantum computing a promising field, but this potential can only be fully realized with improvements in quantum hardware and the development of processes that allow quantum algorithms to successfully execute with the presence of errors.

2.2.1 Qubits

One fundamental difference between quantum computing and classical computing is the way information is encoded. In classical computing, information is encoded as a bit that can either take the state 1 or 0. The state of a classical computer is composed of a vector of these 1s and 0s, with 2^n possible states for n bits. Quantum computing differs from classical computing by allowing an n-qubit system to be represented by a linear combination of 2^n states. This is because of superposition, the feature of quantum systems that allows individual qubits to be in a linear combination of two states, as well as entanglement, which is the condition in which the states of two or more qubits cannot be decomposed into independent representations of each individual qubit's state [55].

Specifically, the state of an n-qubit quantum system corresponds to a vector in a complex vector space $\mathbb{C}^n$. An arbitrary quantum state can be written as $|\psi\rangle$. This notation comes from Dirac notation (also known as bra-ket or just ket notation). A

single qubit has the standard basis vectors of $|0\rangle$ and $|1\rangle$. The kets $|0\rangle$ and $|1\rangle$ are represented in the computational basis by the vectors $\binom{1}{0}$ and $\binom{0}{1}$ respectively. An arbitrary single qubit state can then be defined as $|0\rangle + |1\rangle$ where α and β are complex numbers and $\alpha^2 + \beta^2 = 1$ [39].

Although qubits can exist in superposition, when they are measured, they collapse into a state of either $|0\rangle$ or $|1\rangle$ [39]. The probability that an arbitrary qubit will collapse to $|0\rangle$ is α^2 and the probability that an arbitrary qubit will collapse to $|1\rangle$ is β^2 [39]. Because of this characteristic of qubits, it is very important not to measure a quantum system until the quantum operation is over, or the states in superposition will be lost and the quantum information is lost and outputs of the circuit will be wrong.

2.2.2 Basis States

The states of qubits are more complex to describe than their classical counterparts due to the higher dimensionality of their states. One way to visualize the state of a single qubit is to reference the Bloch Sphere, which is shown in Figure 1. Every

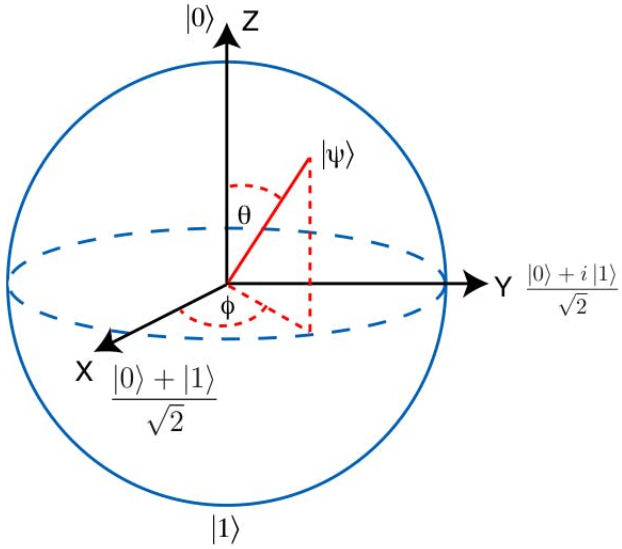


Figure 1: Bloch sphere for qubit visualization. Adapted from de Voorhoede [52].

point on the surface of the Bloch Sphere represents a possible state of a qubit [4]. The location of the point can be written as a vector that is offset from the three axes of the Bloch Sphere. The three axes each represent a different pair of basis states of a qubit. The Z basis is more commonly known as the computational basis [39]. The two orthogonal Z basis states are $|0\rangle$ and $|1\rangle$. The two orthogonal X basis states are $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. The two orthogonal Y basis states are $|R\rangle = \frac{|0\rangle+i|1\rangle}{\sqrt{2}}$ and $|L\rangle = \frac{|0\rangle-i|1\rangle}{\sqrt{2}}$.

This can be seen in the Bloch Sphere, where the north and south poles of the sphere correspond with the computational basis states, $|0\rangle$ and $|1\rangle$. The X basis states are at the end of the vector labeled "X" and the vector orthogonal to it, which correspond with the states $|+\rangle$ and $|-\rangle$. The same is true for the Y and the "Y" vector corresponding to the states $\frac{|0\rangle+i|1\rangle}{\sqrt{2}}$ and $\frac{|0\rangle-i|1\rangle}{\sqrt{2}}$. The pure state of any arbitrary qubit can be represented as the unit vector pointing to a location on the surface of the sphere. The states themselves can then be described as any point on the Bloch Sphere by defining two angles, ϕ and ψ. However, typically states are described as linear combinations of the computational basis vectors $|0\rangle$ and $|1\rangle$.

2.2.3 Single-Qubit Operations

Quantum operations are different from classical operations in a few important ways. In the quantum world, all operations that are not measurements are reversible and represented by unitary matrices [55]. A unitary matrix is one for which the conjugate transpose U^* is also the inverse, i.e. $U^*U = UU^* = I$, where I is the identity matrix [39]. A quantum gate is an operator that acts on qubits, and can be represented by a unitary matrix [39]. Four essential single-qubit operators are called the Pauli matrices shown in Equation 1 represent the Identity, Pauli X, Pauli Y, and

Pauli Z operations [39].

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \; X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \text{ and } Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{1}$$

These operations change the state of a single qubit in the following ways:

- No Change: $I|0\rangle = |0\rangle$, $I|1\rangle = |1\rangle$. In matrix terms, this looks like:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

- X Flip: $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$ In matrix terms, this looks like:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

- Z Flip: $Z|0\rangle = |0\rangle$, $Z|1\rangle = -|1\rangle$ In matrix terms, this looks like:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = - \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

- Y Flip: $Y|0\rangle = -i|1\rangle$, $Y|1\rangle = i|0\rangle$ In matrix terms, this looks like:

$$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ i \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = - \begin{pmatrix} i \\ 0 \end{pmatrix}.$$

Another way to understand how these operations change the state of a single qubit is by referencing the Bloch Sphere. These matrices are ways of rotating the Bloch Sphere about its X, Y, and Z axes by π radians [4]. Using the Bloch Sphere to

visualize the different operations that can be done on a single qubit and how those rotations change the state is a straightforward way to understand how qubit states change in response to single-qubit operations.

The Pauli matrices are both unitary and Hermitian. A Hermitian matrix is a complex square matrix that is equal to its own conjugate transpose. In matrix form, a matrix can be described as Hermitian if $M^\dagger = M$. Due to the Hermitian and unitary properties, the Pauli matrices satisfy a few important identities:

- $X^2 = Y^2 = Z^2 = I$
- $XY = iZ$
- $YX = -iZ$
- $YZ = iX$
- $ZY = -iX$
- $ZX = iY$
- $XZ = -iY$

These can be useful in quantum computation and the applications of the characteristics of the Pauli matrices are a backbone of quantum error correcting codes. For example Pauli matrices are Hermitian, and all eigenvalues of Hermitian matrices are real [55]. Also, the first identity $X^2 = Y^2 = Z^2 = I$ is particularly useful for stabilizer codes. Both of these concepts and the useful properties of Pauli matrices in error correction are expanded further in the chapter.

Another fundamental single qubit gate is the Hadamard gate. In matrix form, the Hadamard gate looks like: $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ In terms of the Bloch Sphere, this gate can change the position of the state vector by moving it away from the poles.

Because the vector is neither at one pole or the other, it is not at fully at one state or the other, so it is in superposition. The Hadamard gate is one gate that can create superposition in a quantum state [39].

2.2.4 Multi Qubit Operations

A couple of types of multi-qubit operations are common in error correcting codes. The first is a quantum CNOT gate. This gate operates on pairs of qubits. One is the control qubit, while the other is the target qubit. If the control qubit is $|x\rangle = |1\rangle$, then the target output will result in $|1\rangle$ if the target qubit is initially $|0\rangle$. Likewise, if the target qubit is initially $|1\rangle$ and the control qubit is $|x\rangle = |1\rangle$, then the target output will change its state to $|0\rangle$. If the control qubit is $|0\rangle$, then no change occurs on the target qubit. More generally, the CNOT gate takes the system from initial quantum state $|a\rangle \otimes |b\rangle$ to $|a\rangle \otimes |a \oplus b\rangle$, where $\oplus$ is addition modulo 2. The quantum CNOT gate can be represented by: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

CNOT gates are important to quantum computation because they have the ability to create entangled states. Entangled states are states of multiple qubits that can be described as one system, but cannot be described separately. This property is fundamental to quantum computation.

2.3 Quantum Error Correction

None of the operations introduced in Section 2.2 are implemented reliably in today's world. No one has created a completely error-free gate, so computations on quantum systems are not entirely reliable. What makes the problem harder is the

fact that qubits themselves are prone to error when idle. These continuous errors that happen in the system are hard to track because they happen slowly, with many qubits at a time. After a short period of time, the whole system decoheres, or loses its intended state needed for computation [44]. This makes practical quantum computation impossible without either error free hardware or an implementation of fault tolerant error correcting codes along with hardware for which the errors lie below a certain threshold. However, since error correcting codes were discovered only recently [8], there's still a lot of work to be done in this field.

Fault tolerance in quantum systems means that a system can correctly encode a logical qubit with many data qubits that can undergo errors and erroneous operations [45]. One theorem that brings hope to practical computation in the face of faulty hardware is the Quantum Threshold Theorem, which shows that if the error to perform a gate operation is a small enough constant, then one can perform arbitrarily long and precise quantum operations [24]. Formally, the Quantum Threshold Theorem is defined below [24]:

> **Theorem:** *There is a threshold error rate p_T. Suppose there is a local stochastic error model with $p_i \leq p \leq p_T$. Then for any ideal circuit C, and any $\epsilon > 0$, there exists a fault-tolerant circuit C' which, when it undergoes the error model, produces an output which has statistical distance at most ϵ from the output of C. C' has a number of qubits and a number of timesteps which are at most $(|C|/\epsilon)$ times bigger than the number of qubits and timesteps in C, where $|C|$ is the number of locations in C.*

Hardware improvement is moving at a steady pace, but it will still not be feasible to make fault tolerant hardware that can execute operations without the need of error detection or correction in the next few decades. For this reason, implementing error correcting codes is a key part in practical quantum computers.

The mathematical representation of a qubit is useful to understand how operations are performed and how information is stored, but it is just one way to understand the underlying physical particles that make up a qubit. Quantum computers can be constructed with subatomic particles, whose physical states can be used for the state of a qubit, such the spin of a particle like an electron [55]. These subatomic systems are extremely delicate, and for that reason are very error-prone. The environment can induce unwanted changes in the system through vibrations, temperature fluctuations, and electromagnetic changes, among other types of interference [55]. The qubit itself can be made of different materials, including ion traps, superconductors, quantum dots, and more. Depending on the implementation of the qubits, whether it be transmon qubits made with quantum superconductors, or trapped atomic ions [35], hardware can vary in decoherence times, error rates, and connectivity between qubits.

2.3.1 Noise Models

Noise models for quantum systems are classified by the types of qubits and operations that have error simulated, and whether or not the errors that act on the qubits are correlated with other errors or independent. The depolarizing noise model is a simplified noise model that randomly chooses one of the Pauli operators, X, Y, or Z, to act on an arbitrary data qubit [54]. This model makes a few important assumptions. The first is that each error acts on a single qubit, independently of other errors. The next is an X, Y, or Z error can act on an arbitrary data qubit with equal probability. This is in contrast real quantum systems, which tend to experience the phenomenon known as decoherence, in which many qubits of the system drift slightly from their intended state. When this happens, the intended state of the system is lost [39]. Qubits also tend to decohere to their ground state of $|0\rangle$, rather than to an arbitrary state. This means that the type of errors acting on the qubits do not

occur with equal probability. Thus, noise models must be understood as simplified versions of how noise acts on quantum computers. Since hardware is a limiting factor with many experiments in quantum computation, simulations and noise models are are necessary parts of carrying out those experiments.

Other types of noise models are either simpler or more complicated than the depolarizing noise model, depending on the application. An even simpler one performs an error on an arbitrary data qubit with probability p and either an X or Z flip, but not a Y flip, occurs on the data qubit with equal probability of an X or Z error [34]. Noise models can include errors on the corrections and measurements themselves, which happens on quantum computers. Noise models that capture decoherence are much more complex, and require significant effort towards understanding the physics and mathematics to classically simulate [26]. Many of these noise models are used in current efforts to experimentally evaluate decoders, although efficient and accurate quantum noise modeling is an active area of research in and of itself [29, 21].

Although the depolarizing noise model has disadvantages, it is suitable for experiments that evaluate surface codes and the algorithms to decode them. It is used to compare surface codes to other topological codes and develop decoding algorithms for surface code. Another use of the depolarizing noise model is to find theoretical thresholds of error for topological codes that contribute to the current body of research in topological quantum error correcting codes. One way this simple noise model can be expanded is to apply errors with equal probability on both ancilla qubits and data qubits. The case in which there are errors on measurement qubits, also referred to as ancilla qubits in the context of surface codes, is known as the fault tolerant case [14, 53].

Measurement error rates are easily accessible through the data collected by IBM quantum computers. These are referred to as read-out errors, and the read-out error

is calibrated periodically for each qubit on each IBM machine.

2.3.2 Stabilizer Formalism

A stabilizer code for quantum error correction is described in terms of the number of stabilizers, each of which is an operator, and for which the collected measurements give information about what errors have affected an encoded state [6, 23].The key idea of the stabilizer formalism is to represent a quantum state $|\psi\rangle$ not as a vector of amplitudes, but by a stabilizer group, which consists of unitary operators, such as the Pauli operators, that stabilize $|\psi\rangle$ [3]. Unitary U stabilizes a pure state $|\psi\rangle$ if $U|\psi\rangle = |\psi\rangle$. Note that if U and V both stabilize $|\psi\rangle$, then any product of U, V, U^{-1}, and V^{-1} also stabilizes $|\psi\rangle$. The identity operator I also stabilizes every quantum state [3].

The Pauli matrices, which are single-qubit operations, can be used to stabilize certain quantum states. A summation of the states that Pauli matrices stabilize:

- I stabilizes all quantum states
- $-I$ stabilizes no quantum states
- X stabilizes $|+\rangle$
- $-X$ stabilizes $|-\rangle$
- Z stabilizes $|0\rangle$
- $-Z$ stabilizes $|1\rangle$
- Y stabilizes $|i\rangle$
- $-Y$ stabilizes $|-i\rangle$

In a stabilizer code, the encoded states are chosen to be in the mutual +1 eigenspace of a set of multi-qubit Pauli operators [23]. This means that the stabilizers of the code are made of Pauli operators that all mutually commute if no data qubit in the encoded state has undergone an error. Stabilizer codes are described by sets of data qubits and the operators that act on them, which are called stabilizer generators. Measuring these generators gives information about the parity of the states of the connecting physical qubits. If the stabilizer generators commute, the stabilizer has an eigenvalue of 1, and if they anticommute, it has an eigenvalue of -1. Measuring all the stabilizers generates the error syndrome of the stabilizer code [54]. These stabilizers can be measured without disturbing the states of the data qubits, but the outcomes of these measurements provide information about the errors occurring in the system.

Each stabilizer code is characterized by parameters $[n, k, d]$ where n is the number of physical qubits, k is the number of logical qubits, and d is the minimum distance of the code [32, 43]. The distance of a stabilizer code is the minimum number of data qubits with nontrivial errors such that the code subspace is still preserved [43]. If an encoded subspace is damaged by X, Y, or Z error chains, then it can be recovered successfully by measuring the stabilizers and applying corrective operations on the predicted erroneous data qubits [32].

2.3.3 Surface Code

Although the repetition code is a useful tool for understanding error correction and the underlying concepts, it does not have a low enough error rate to implement algorithms that can work using real-world quantum computers. However, another class of quantum error correcting codes called topological codes have greater practical use. Historically, this class of codes grew out of a specific type of stabilizer code, called

the toric code [32, 7]. Various types of topological codes are used almost exclusively when it comes to research in large-scale error correction. The advantages of the toric code and other topological codes include high thresholds for error as well as simple, repeated geometry that makes error correction codes of large depth more straightforward to construct than other types of large error correcting codes [7]. The type of topological code used in this research is called the surface code, which is similar to the toric code, but with a planar rather than toroidal geometry [18]. A visual representation of the surface code is presented in Figure 2.

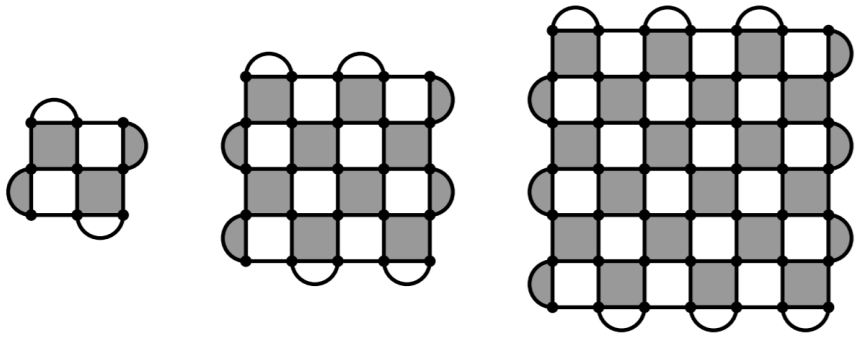


Figure 2: Surface Code Examples of Depths 3, 5, and 7. Dots represent data qubits. "Depth" is the number of data qubits along one edge of the grid. Bordered regions (squares and semicircles) represent ancilla qubits. Gray regions represent X-type ancilla that measure X stabilizers on two (semicircle case) or four (square case) neighboring data qubits. White regions represent Z-type ancilla that measure Z stabilizers on two or four neighboring data qubits.

Surface code is a type of stabilizer code with stabilizer generators made from neighboring qubits. The stabilizers of the surface code are measured by ancilla qubits. The stabilizer operations are performed on the data qubits that lie on the vertices of the ancilla in the grid representation. In the grid representation, each data qubit neighbors two X-type ancilla and two Z-type ancilla, unless the data qubit lies on the boundaries of the grid. If it lies on the boundaries, then the data qubit neighbors only a total of three ancilla qubits. The data qubits on the corners only neighbor two ancillas. A Z-type ancilla can put its four neighboring data qubits, a, b, c, and

d, into an eigenstate of the operator product $\hat{Z}_a\hat{Z}_b\hat{Z}_c\hat{Z}_d$ [18]. Each Z-type ancilla therefore measures a Z-stabilizer. The neighboring qubits for X-type ancilla are in an eigenstate of $\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d$, and measure X-stabilizers [18].The ancilla qubits of the surface code measure the parity of the data qubits that belong to the stabilizer. This means that if an odd number of data qubits are not in the same state as the others, then the eigenstate of the stabilizer is -1 and the ancilla qubit detects an error.

Several steps are required to create the stabilizers of the surface code. Each ancilla qubit is initialized in its ground state and then undergoes a sequence of CNOT operations with its neighboring data qubits followed by a projective measurement [18]. For X-type ancilla, the four neighboring data qubits are the targets of four CNOT operations with the ancilla qubit being the control qubit. The projective measurement of these operations yields the eigenstate of $\hat{Z}_a\hat{Z}_b\hat{Z}_c\hat{Z}_d$. The Z-type ancilla qubits are entangled in the same way to their neighbors, except that a Hadamard gate is applied to the ancilla qubit before and after the CNOT operations. The Hadamard gate effectively changes the basis of the measurements so that changes in parity in the X-basis of data qubits can be detected by the X-type ancilla. The Hadamard gate is performed before and after the CNOT gates in order to measure the parity in the X-basis. The projective measurement then yields an eigenstate of $\hat{X}_a\hat{X}_b\hat{X}_c\hat{X}_d$. The error syndrome is the results of all the projective measurements of all the ancilla qubits. Figure 3 shows how X-type and Z-type ancilla are set up and entangled with the neighboring data qubits.

Error chains on the surface code refer to the presence of multiple data qubit errors. Multiple data qubit errors with the same type of error form error chains if they are neighboring sets of data qubits. An example is shown in Figure 4. The middle portions of error chains that only flip two data qubits of an ancilla erroneously are not detected by the surface code. Since $X^2 = Y^2 = Z^2 = I$, two pairs of parity check

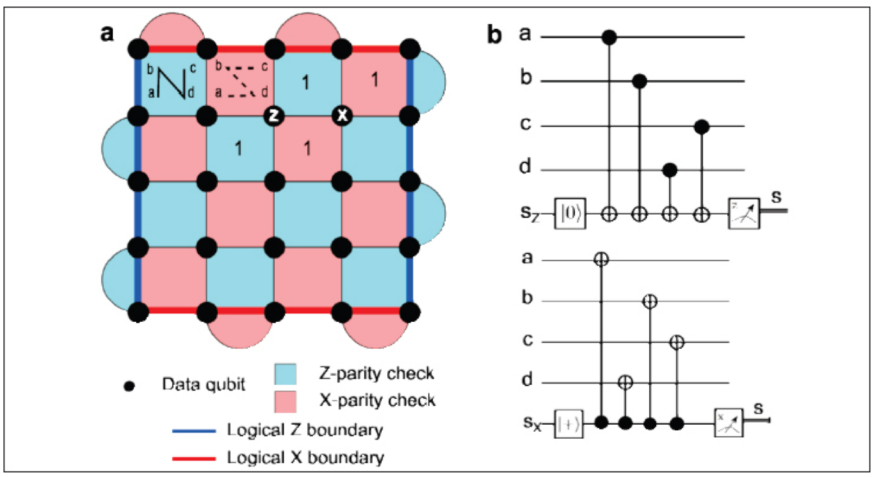


Figure 3: Example of surface code with visual of ancilla quantum circuit connections. The circuit describing the CNOTs performed on the data qubits and ancilla is presented for both X and Z ancilla. The vertical and horizontal boundaries that correspond with logical operators are also shown. Adapted from NAE report *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2018 Symposium* [40].

operators that are the same results in $I^2 = I$, which has a $+1$ eigenvalue. Since the middle of the error chains change the states of two qubits in a stabilizer, under the commutation properties discussed in Section 2.2.3 the eigenvalue of the stabilizer is $+1$. Figure 5 shows examples of detection events resulting from the parity checks of the stabilizers.

The logical states of the surface code are created by initializing all of the data qubits to the intended logical state, and then maintaining the intended state through error correction [27]. Logical operations can be performed by changing the state of all data qubits along the boundaries of surface code [27]. Generally the logical X operator is the top or bottom horizontal boundary and the logical Z operator is the left or right boundary.

An error chain can escape detection if it connects back with itself to form a loop in the circuit. Non-trivial error loops are error chains that reach from one boundary of the code to the other, either vertically or horizontally. This causes an unintentional change in the logical state. Preventing these non-trivial error chains is necessary to

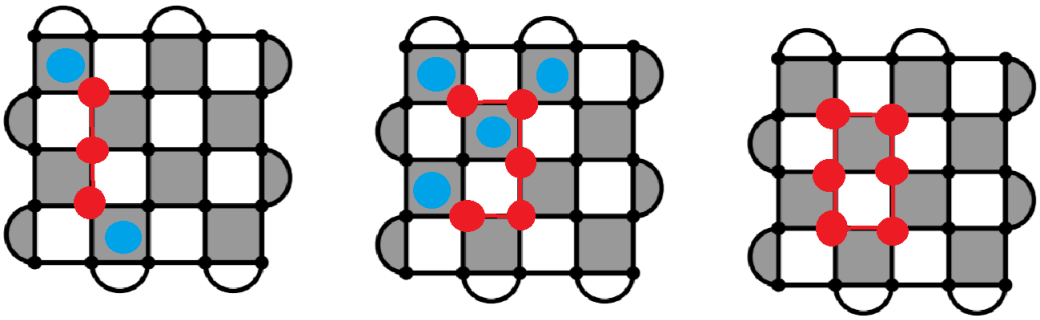


Figure 4: Example of depth 5 surface code with X-type error chains. Each surface code denotes and X-type error on a data qubit in red. The blue dots are the X-type ancilla qubits detecting an error. The neighboring data qubits form error chains, with the last chain being one that loops around and results in no detection events from the ancilla qubits. Adapted from [50]

prevent logical errors in the surface code. Preventing this type of logical error is achieved through decoding, which is the process of using error syndrome to identify corrective operations to perform on data qubits.

Surface codes are widely used due to a few distinct advantages. They can withstand relatively large levels of error when compared to other error correcting codes. Another advantage is that the parity checks done by the ancilla qubits require only nearest neighbor interactions. The nearest neighbor interactions allow surface code

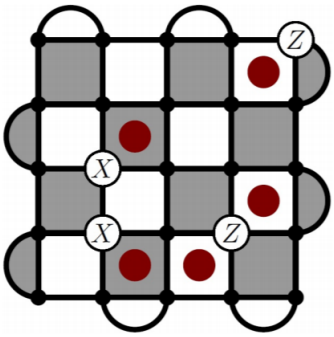


Figure 5: Example of depth 5 surface code. X's and Z's denote data qubit errors. Red dots denote detection events. Adapted from [50]

to be realized with quantum hardware implementations such as ion traps or superconducting qubits. The surface code also has a repeated pattern, making larger code depths easy to understand. This helps surface code be scalable, which is vital to quantum error correction since large code depths are associated with more reliable logical qubits.

2.4 Decoding

Understanding why and how stabilizers create error syndromes for surface codes is important, but the key component of QEC is using a method to determine which operations to perform on the physical qubits in order to preserve the intended logical state of the code. Decoding is computed classically, and must be integrated with quantum computers to provide corrective operations. Several decoders have been developed over the years, and this is an ongoing area of research. The main goal of research in decoders is to find a suitably accurate and fast decoder that is flexible enough to work with various noise models and large code depths.

2.4.1 Decoder Characteristics

Decoding algorithms must be improved to reach the full potential of quantum error correcting codes. The current best threshold of maximum error on data qubits for surface codes and toric codes with accurate decoders is 10-11% of the data qubits on the code undergoing an error. As long as the probability of error on the data qubits is below this threshold, then the surface code can maintain the logical qubit state indefinitely [43]. The threshold value is based on experiments on the surface code that don't include either noisy measurement or erroneous corrective operations [14, 43, 53]. This threshold assumes measurements and corrections are noiseless, and is based on the depolarizing noise model. As such, the simulation of error on the quantum

computer is simplified. The current threshold for the fault tolerant case for surface codes is 1% [45, 53]. The threshold that only includes data qubit errors is optimistic, and so more extensive research into various types of decoders is necessary to find one that can best work under realistic noise from a quantum computer.

Fast decoding is necessary to implement large-scale quantum computers with error correcting codes such as surface code. Decoding itself is an NP-complete problem [28], but there has been work into finding approximation algorithms and parallel programs to speed up the decoding process [17]. When a decoder is fast, more error correction measurements are made, which detects errors faster on the code, and prevents a high incidence of errors which leads to the intended logical state of the code being lost. Although small error correcting codes have fast decoding times, the run time of decoders increases with the depth. This is because increasing the depth of the code increases the number of physical and ancilla qubits and the number of possible syndrome measurements is 2^n, where n is the number of ancilla qubits. It is predicted that hundreds, or even thousands of physical qubits are needed to implement surface codes for practical quantum computation [43], which means that scalable decoding algorithms are integral to implementing topological codes.

One important consideration in designing a decoder is whether a decoder is a high-level or a low-level [49] decoder. High-level decoders perform corrective operations on surface code through logical operations. This allows for some small data qubit errors to occur and not be corrected as long as they do not corrupt the intended logical state of the code. High-level decoders only need to choose between four logical operations, a logical X, Y, Z, and identity operation [49]. By contrast, low-level decoders work to apply corrective operations to individual data qubits. Low-level decoders can help prevent error chains from propagating and becoming unmanageable in surface code, but they also run the risk of false predictions causing more errors on the circuit than

there were previously.

2.4.2 Examples of Decoders

There are several ways to determine which corrective Pauli operators to perform on data qubits from a given error syndrome. However, better decoders are always desired since decoding time is one of the limiting factors to practical error correcting codes. The simplest decoder that can be implemented is a full lookup table. For each given syndrome, a full lookup table returns the most probable set of data qubit errors that could generate that syndrome. The most accurate decoder possible is a lookup table that accurately represents the distribution of data qubit errors and their probabilities given an arbitrary syndrome, which always returns the most probable set of errors. However, the number of entries of a lookup table decoder quickly increases, since the error syndromes increase as a function of depth, where the $syndromes = 2^{depth^2-1}$. This results in exponential growth rate of the lookup table, making large code depth unfeasible for large scale computation. The best that could be done would be a partial lookup table (PLUT), which is an incomplete lookup table which is only populated with some syndromes and their corresponding corrective operations, but not all. A PLUT would not be able to predict corrective operations for syndromes that are not in the lookup table.

The most common type of decoder is an algorithmic-based decoder that uses the Minimum Weight Perfect Matching (MWPM) algorithm [11]. This is the standard decoder used with surface code because this decoder has the highest accuracy of any known decoder, other than the complete lookup table, and is the most researched decoder of topological codes. This algorithm works by finding the most likely path between two endpoints. This is useful for surface code decoding since the ancilla measurements that read out -1 are the endpoints of error chains on the surface code.

The MWPM decoder transforms the error syndrome of an observation into a syndrome graph, where the vertices are the ancilla qubits, and every edge represents a physical qubit. The algorithm is run on the syndrome graph, and returns the edges of the syndrome graph that corresponds to the physical qubits with errors. This syndrome graph is constructed and run for the X and Z syndrome measurements separately.

Although the MWPM decoder is considered the standard when it comes to decoding surface code due to its advantageous accuracy and run time trade off, it has some disadvantages as well. This decoder only works with the depolarizing noise model, where errors are independent and of equal probability. In real world quantum computing, these assumptions are unlikely to be true. More complex and realistic noise models exist for quantum systems and so the main disadvantage of MWPM is the applicability to real-world quantum circuits. Figure 6 illustrates a syndrome graph overlaid on top of surface code.

Once both the X and Z syndrome graphs are constructed from a given syndrome for a code, they are fed into the MWPM algorithm. In graph theory, this algorithm works by finding a perfect matching of a given graph G with vertices V and edges E. A matching is a set of non-adjacent edges, which also can be thought of a set of edges in a graph that do not share any vertices. A maximum matching is a graph for which the cardinality of the matching is maximized for the graph. A perfect matching in graph theory is when the cardinality of the matching is $|E|/2$, which is always a maximum matching for a graph. The MWPM algorithm also takes into account the weights of edges in a graph, and returns a perfect matching with the minimum weighted sum of edges. This algorithm runs in $O(n^2)$ [11],where n is the number of detection events [17], although parallel optimizations are being developed that run in $O(1)$ [17]. It can be applied to both X syndrome and Z syndrome graphs for which the edges are weighted by the length of the shortest error chain between

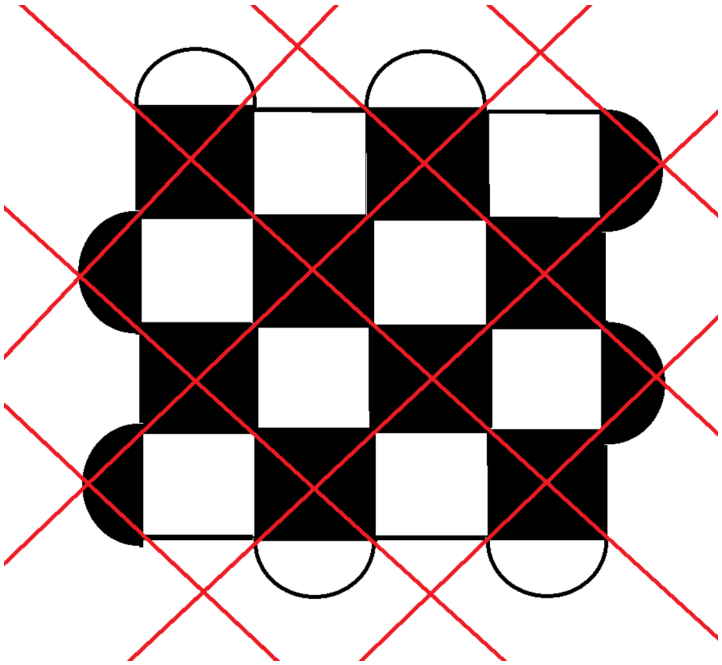


Figure 6: Syndrome graph used for Minimum Weigh Perfect Matching algorithm. The black squares and semicircles denote X-type ancilla. The white squares and semicircles denote Z-type ancilla. The red lines indicate an X syndrome graph that is created for the MWPM algorithm. The vertices of the graph are on the X-type ancilla, while the edges are overlayed on the data qubits.

the syndrome nodes as determined by the syndrome graph.

2.5 Machine Learning

Machine learning is a common technique by which systems can acquire knowledge and learn patterns from raw data instead of relying on hard-coded knowledge [22]. This allows computers to make predictions on new data by drawing on the expected values of patterns learned and knowledge acquired. Machines can learn these patterns more effectively if the data presented to the computer has a useful representation [22]. Each piece of information included in the representation of the data is known as a feature. A machine can learn the correlation between certain features and outcomes, and use this learned correlation to predict the most likely outcome when the features are presented to it in another observation.

In general machine learning problems fall into various categories. One useful designation is whether a machine learning problem is supervised or unsupervised. Unsupervised learning learns the properties and patterns of a large set of data, and can apply the learned structure of the dataset to various problems. Some examples include clustering and denoising [22]. Supervised learning also learns the properties of a set of data, but in contrast to unsupervised learning, each observation is assigned a label. Another way of categorizing machine learning problems is as either regression or classification problems. A regression problem is a problem where the output is a continuous value, whereas a classification problem's output is categorical. Classification problems can be either binary or multi-class. A binary classification problem uses the features of an observation to determine to which one of two classes the observation belongs. A multi-class classification problem uses features to determine an observation's membership in a single class out of more than two classes. A machine learning problem in which an observation can be assigned one or more class labels out of many is called a multi-label classification problem.

For training, there are general principles that are important in the learning pro-

cess. Once the features and the labels of a supervised learning problem are identified and the data processed correctly, then the dataset is partitioned into multiple subsets. Most of the data is put into a training dataset, while a minority of the dataset is set aside for testing [31]. This test dataset is not used for data exploration and is held out until the model is fully trained[31]. This is done in order to esnure the test accuracy is an accurate reflection of how well the model can predict unseen data. During training, the machine learning algorithm finds patterns in the training dataset and learns how to predict observations based on the training data. The other way that a dataset is partitioned is a subset within the training set. A validation set is taken from the training set and is used to evaluate a model during training for parameter tuning.

There are a few metrics that are used to evaluate the quality of a given model. The first is accuracy, which is the ratio of correctly predicted observations to total observations. This is an evaluation metric that is widely used and most people are familiar with. This metric is most suited for datasets that have balanced class labels and when it is desirable to have results that are easy to interpret.

Accuracy is a useful performance metric, but there is more to performance measurement than an accuracy report. Given a binary classification problem, where a class label can either be positive or negative, often represented by [0,1], there are four ways a model can classify or missclassify a class label:

- True Positive: The predicted class value is positive and the actual class value is positive
- False Positive: The predicted class values is positive and the actual class value is negative
- True Negative: The predicted class value is negative and the actual class value

is negative

- False Negative: The predicted class value is negative and the actual class value is positive

These parameters are used to describe certain evaluation metrics. In terms of the parameters described above, accuracy is the sum the counts of the True Positives and True Negatives over the count of entire observations. Additional important metrics are F1 score, precision, and recall. Precision is the ratio of true positive over the total number of observations predicted to be positive (TP / TP + FP). A high precision close to 1 indicates a low false positive rate. Recall is the ratio of observations correctly predicting positive divided by the total number of observations with an actual class value of positive (TP/ TP + FN). By themselves, these are valuable metrics, but F1 score is a metric that gives the weighted average of precision and recall. F1 score is particularly useful given an uneven class distribution. F1 Score = 2*(Recall * Precision) / (Recall + Precision).

2.5.1 ROC Curves

ROC curves are also used to determine a model's quality. ROC curves plot true positive rate on the Y axis of a graph, and the false positive rate on the X-axis. These rates are plotted for classification thresholds. The point of optimal classification threshold is located in the upper left corner of the graph, where the best tradeoff of false positive rate and true positive rate occurs. The area under the ROC curve (AUROC) is also typically higher for better models. Figure 7 and Figure 8 show examples of ROC curves and the meaning of the area under the ROC curve.

Since there are many labels, both macro-average and mirco-average ROC curves could be used to determine how good the model is. The macro-average computes the ROC independently for all labels and takes the average [37]. This metric treats all

classes equally. By contrast, the micro-average uses the total values of all the classes and averages it based on the total number of observations [37]. These metrics are the same in the case of balanced classes for classification problems, but vary greatly in the case of imbalanced datasets.

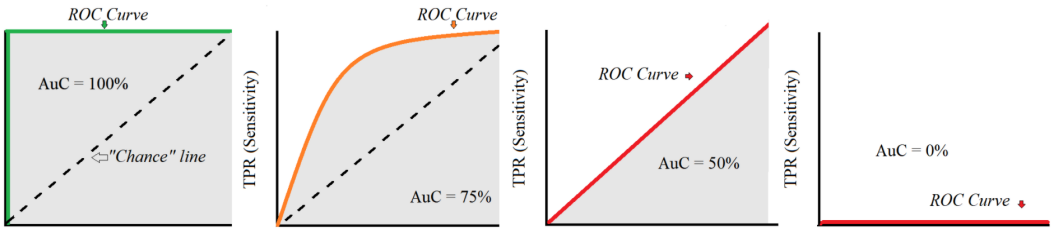


Figure 7: Example of AUROC adapted from [33]

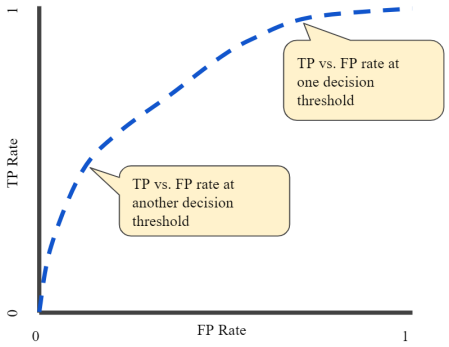


Figure 8: Example of ROC adapted from [1]

2.5.2 Cross Validation

Cross validation is a technique in machine learning that is used to provide an understanding of model performance that has less variance than a simple train test split. In a train test split, the dataset is split with most of the data being used for training and validation, while the rest is not looked at or trained on until final model evaluation. While this is a sufficient technique to use with large datasets and models that require long training times, the performance of the model is dependent on the

data in the training set and test set. If the dataset is small, then this results in high variance of model performance depending on the train test split.

K-Fold cross validation is a process that remedies this by splitting the dataset into k folds, where k is 5 or 10 in most cases. Each fold is used as a test set while the rest of the dataset is used for training the model. Therefore, the model is trained k times and tested k times with a different test set. The average and standard deviation of performance metrics are evaluated at the end of the process. This produces lower variance than the standard train test split, but it takes longer.

Cross validation can also be used for hyperparameter tuning in machine learning models as well. Cross validation for hyperparameter tuning is often done alongside a train test split, and is only used on training data. The process is done by creating a list of possible hyperparameters to use in a machine learning model. The combinations of all of these hyperparameters are evaluated on a training set using the average performance from K-fold cross validation to determine the best set of hyperparameters. The final model is then evaluated on the test set. A visualization of how cross validation is performed is shown in Figure 9

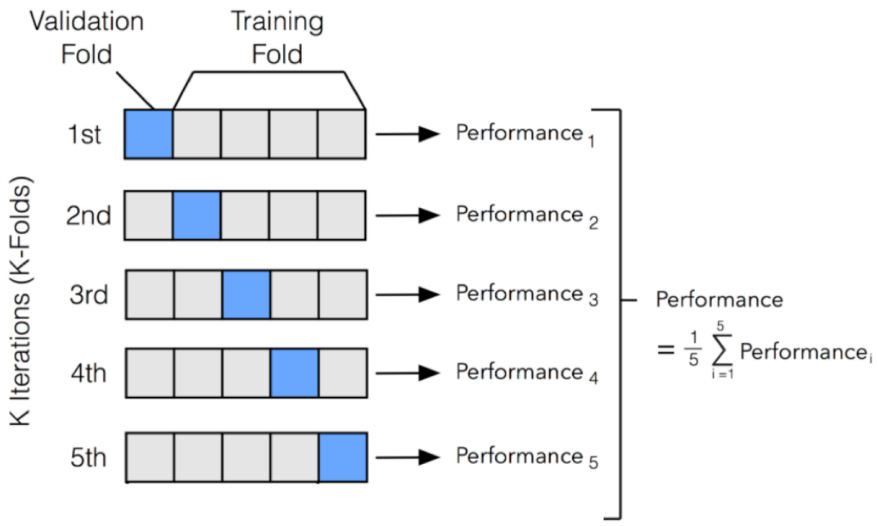


Figure 9: Visualization of K-fold cross validation adapted from [41]

2.5.3 Random Forest

Random forest classifiers are classifiers that are made up of decision trees. Decision trees are are classification method that takes features as input and splits branches of the tree based on the values of each feature. Random forest is an ensemble classifier, which uses an ensemble of classifiers to vote for whether a class label belongs to a set of features. Each classifier in the ensemble is a decision tree. This has less variance than most single classifiers by themselves, which is one reason that random forest classifiers are popular. A visualization of how the random forest algorithm is performed is shown in Figure 10

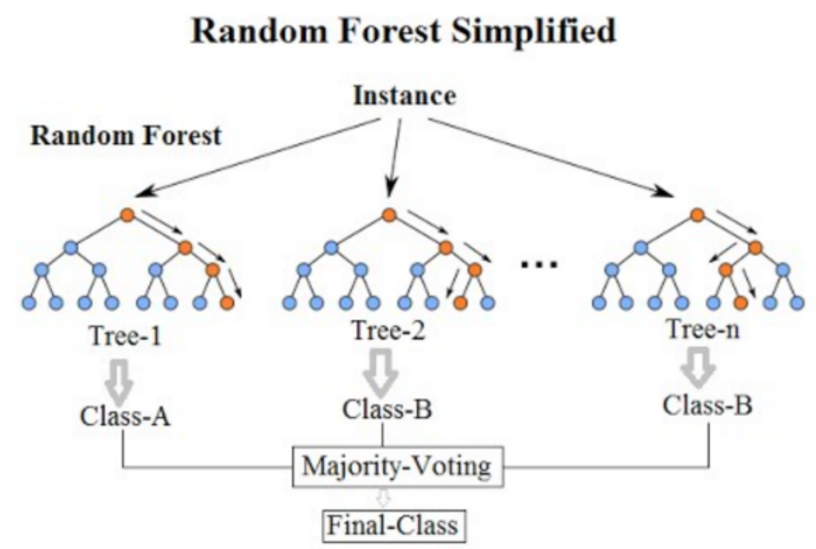


Figure 10: Visualization of random forest classification, adapted from [30]

2.5.4 kNN

K-Nearest Neighbors(KNN) is a simple and widely used algorithm in machine learning. The distance between feature vectors of the data are found using algorithms such as euclidean distance, and are a way to compare the similarities between observations. The probability that a new observation belongs to a particular class depends on the distance between the new observation and all the observations of each class. A visualization of how the K-Nearest Neighbors algorithm is performed is shown in Figure 11.

Multi-K Nearest Neighbors (MLkNN) adapts the KNN algorithm for use with multi-label classification. The MLkNN algorithm is implemented in Python in the scikit-multilearn package [56]. MLkNN uses KNN to find the nearest neighbor of the example, and then uses Bayesian inference to assign a maximum a posteriori principle is used to determine the full label set to assign to the unseen observation [56].

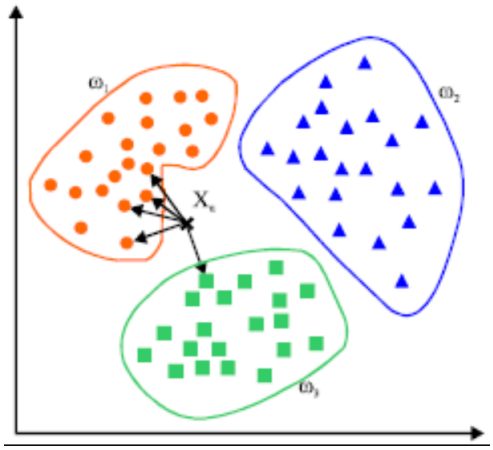


Figure 11: Visualization of KNN, reproduced from [57]. Each point represents an observation, and each observation is assigned one of three class labels. The various classes are denote by different colors. A new observation is assigned a class label based on the distance between it and all the observations of the various classes.

2.5.5 Multi-Output Classifier

Not all machine learning models are suitable for multi-class or multi-label classification. One strategy to use these methods is with a `MultiOutput` Classifier supported by Sklearn. This is a simple method where a classifier that is not natively multi-label or multi-class can be used for muli-label and multi-class classification. In the case of multi-class classification, only one class can be assigned a positive class label at a time. In multi-label classification, more than one class can be assigned a positive class label. This is a much more difficult task, since there are 2^n possible outputs, given a problem with n classes. The way to construct a multi-label classifier is more difficult as well, since one classifier is fit per class label. In testing, each of the classifiers use the features as input to predict the outcome of each class label.

2.5.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a popular machine learning architecture that is inspired by neurobiology to represent functional patterns between inputs and

outputs. These are networks that are made up of neurons, or perceptrons, which when stacked together and connected can learn complex functions [22]. A fully connected neural network is made up of an input layer, an output layer, and one or more hidden layers. Each neuron in this type of model is connected to each subsequent neuron in the next layer. In this case, "connected"means the output of a neuron is part of the input of a connected neuron. Each layer in an ANN receives inputs, multiplies these inputs by a vector of weights that were updated through training, and then passes the weighted sum of the inputs through an activation layer. This activation layer normalizes the output of each layer, and can determine whether the neuron is active or not by evaluating if the output value is high enough to be relevant. Activation functions can normalize values to between -1 and 1, 1 and 0, or even just act as a step function that returns values of [1,0]. Various activation functions can affect model performance, and help the network to learn complex functions. The output of a layer of a fully connected network is $f(Wx + b)$, where f is an activation function, W is a matrix of weights that is multiplied by the input vector, and b is the vector of biases that is added to the product of the weights and the input.

Stacking layeres of perceptrons can lead to powerful networks that can accurately predict a variety of problems, depending on the architecture. The way to define the architecture of a neural network is by describing it in terms of the configurations of the nodes, connections between them, and the functions used. An example of a simple network is shown in Figure 12

There are a few parameters that are important to a model's performance. Some important parameters are summed up below:

- Nodes per layer: More node increases capacity of network
- Number of Hidden Layers: More layers increases capacity of network

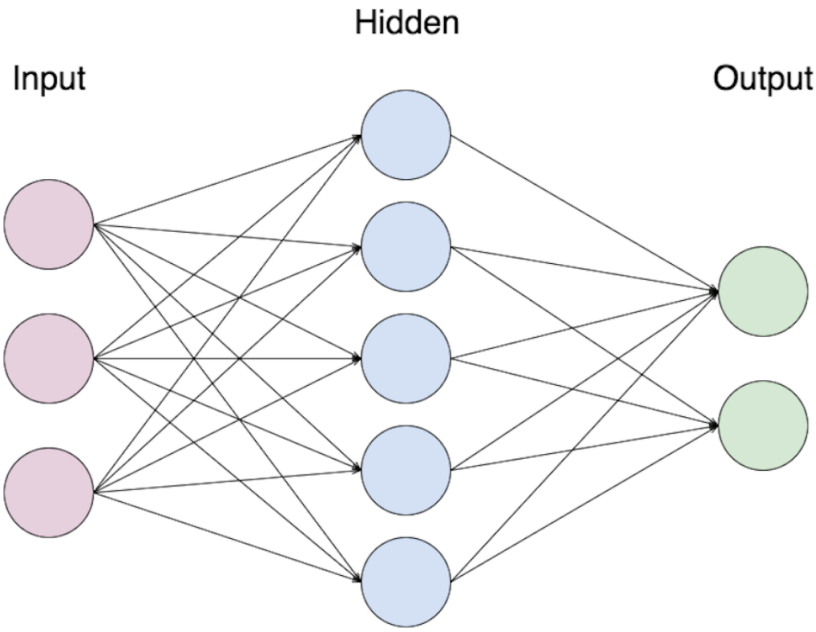


Figure 12: Example of a simple fully connected neural network with one hidden layer, adapted from [12]

- Learning Rate: Controls how much to change to the weights of the network in response to the training error
- Activation function: Allows model to learn nonlinear functions by normalizing and determining if the output of each perception is activated or not
- Optimizer: Determines how to change weights to reduce the error between the predicted and actual values of the output of a neural network

In neural network training, if the model cannot reach a high training accuracy, then the capacity of the network needs to increase, assuming there is a meaningful relationship between the input observations and their output labels. Once the training accuracy is high and the validation accuracy is close to the training accuracy, the model can be used to predict the test set. If the validation set accuracy is significantly less than the training accuracy then overfitting has occurred, which is when a model

memorizes the training set but cannot generalize for unseen data. Regularization methods are used to decrease model capacity for better generalization, and the model can be evaluated again.

2.6 Hypothesis Testing

Statistical methods of determining if results are contrary to or in support of hypotheses is known as hypothesis testing. Statistical tests help determine if results are meaningful or if they happened by chance.

2.6.1 Paired T-Test

A paired t-test is a statistical test to find if the mean difference between two sets of observations is significant. This test has the following assumptions:

- The dependent variable is continuous
- The observations are independent
- The dependent variable is approximately normally distributed
- The dependent variable does not contain outliers

For comparing classification methods, the mean difference in accuracy between classifiers can be used as the dependent variable. This mean is produced from the accuracy of each classifier on the test set in k-fold cross validation.

2.6.2 5x2 Cross Validation

5x2 Cross validation is a procedure to perform a paired t-test on two classifiers, but is a variant of k-fold cross validation. Instead of performing k-fold cross validation with 5 or 10 folds and recording the difference between the accuracy of each classifier,

cross validation is repeated 5 times and k=2 for each iteration. After the difference values are captured for each fold, then a t-test is performed to determine if the performance is significant. This is the same t-test discussed in Section 2.6.1, but the process to obtain the difference values used in the test is different.

2.6.3 Matthew's Correlation Coefficient

Matthew's correlation coefficient (MCC) is a metric used to determine the value of a classifier. This metric is closely related to the chi-square statistic for a 2x2 contingency table. This equation describes the relationship between MCC and chi-square, where N = number of observations:

$|MCC| = \sqrt{\frac{\chi^2}{N}}$

This coefficient is generally considered one of the best ways to describe a confusion matrix of true and false positives and negatives in just a single value [9]. The way to calculate MCC given all the true positives, true negatives, false positives, and false negatives is: $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

2.6.4 McNemar's Test

McNemar's test is a statistical test for paired nominal data. This test is applied to paired outcomes of classification tests. A contingency table is a table that describes when two classifiers both are correct, incorrect, or when one is correct while the other is not correct. Below is an example of a contingency table comparing the outcome of two classifiers.

Both classifier 1 and classifier 2 are correct	classifier 1 is correct and classifier 2 is incorrect
classifier 1 is incorrect and classifier 2 is correct	Both classifier 1 and classifier 2 are incorrect

These cells are denoted as:

n_{11}	n_{10}
n_{01}	n_{00}

n_{01} and n_{10} capture the error rates of the two classifiers, and so the test statistic from McNemar's measures if there is a significant difference between these two values. The null hypothesis is that the two error rates of the classifiers are the same, where $n_{01} = n_{10}$. McNemar's test is based on χ^2 test for goodness of fit with one degree of freedom [15]. The test statistic is calculated by: $\frac{(|n_{01}+n_{10}|-1)^2}{n_{01}+n_{10}}$ If the χ^2 is significant, the null hypothesis can be rejected in favor of the alternative hypothesis that the relative proportion of errors on the test set made by the two classifiers are significantly different. This would imply that the classifiers have significantly different performance on the test set relative to one another [15]. This test works best if the variability in results due to the training set is small, and the test set is large [15].

2.7 Neural Network Decoders (Previous Work)

In the past few years, researchers have been looking more heavily into using machine learning-based decoders. Several different machine learning techniques have been used, including reinforcement learning based decoders and belief propagation networks. These various decoders offered advantages and disadvantages, which depends on the size of surface code being tested, the noise model being used, and the dataset being trained on. The most consistent advantage of machine learning based decoders compared to the standard partial lookup table decoder or MWPM decoder is the flexibility to use a network with no modifications to the architecture on various topological codes and noise models. In just the past 8 years, the research in neural network decoders has expanded rapidly and will likely not slow down in the future due to evidence of their advantages.

One technique that has been researched is the use of a restricted boltzmann machine to decoder stabilizer codes. A restricted boltzmann machine is a generative stochastic neural network that is made up of two layers, a visible layer and a hidden layer. The visible nodes are only connected to hidden nodes and vice versa. This type of network is useful for dimensionality reduction, classification, feature learning, and topic modelling, among other topics. For error correction, researchers developed a network that is optimized to fit a dataset where the inputs are the error syndrome and their error chains [47]. The resulting network was trained to model the underlying probability distribution, with the goal to predict recovery operations to perform on unseen error syndromes. The errors simulated only came from a phase-flip channel, and so the errors simulated included syndromes generated from various data qubit error rates. To achieve this for surface code, many networks had to be produced to create the predictions of recovery that depend on a given error probability p. The results showed that this method produced a similar performance to Minimum Weight Perfect Matching decoding for simple error models. While the accuracy of decoding is similar, the boltzmann machine decoder has the advantage to be easily trained with different noise models, and the flexibility to be applied to other topological codes with only minor adjustments [47].

There have been several proposed machine learning programs that have managed to balance accuracy and execution time well for surface codes of small depths. One of them is a model developed by Baireuther, O'Brien, Tarasinski, and Beenakker, which used two neural networks that had Long Short Term Memory layers [5]. These layers were determined to be suitable for the neural network architecture because the training data was made up of error correction cycles. Error correction cycles measure the error correction multiple times in a short time frame before applying corrective operations [19]. This method is used to mitigate measurement errors on

ancilla qubits. In the research, these cycles were labeled with different time steps, and the error model could introduce new errors between cycles due to the degenerative nature of qubits. The cycles are advantageous because they can better help the model to learn how to account for measurement errors on the ancilla qubits. The surface code itself is prepared in a known logical state that is held for a certain amount of error cycles before readout. Two separate neural network decoders with similar architectures are created for both X type errors and Z type errors. This architecture was only tested on the surface code of depth 3 and compared the performance to the Minimum Weight Perfect Matching decoder. The fidelity of the logical qubit was more reliable in the neural network decoder, as well as the ability for the model to correct correlated errors as compared to the Minimum Weight Perfect Matching decoder. This decoder also has the possibility of extension to other topological codes without modifications to design, which supports the evidence of more flexibility of applications in machine learning based decoders[5].

One well known neural network decoding research was done by Varsamopoulos in 2017. Their research uses surface codes of depths 3, 5, and 7 and applied a high-level neural network decoder to decode these code depths. The high-level decoder has high dimensional inputs that are the error syndromes of the datasets, as well as low dimensional outputs that are the possible logical operators for surface code. These logical operators are the logical X, Y, Z, and Identity operator to the surface code's logical state. The depolarizing noise model is used to generate two separate datasets. One dataset did not generate errors on measurement qubits and the other dataset generated errors on measurement qubits with the same probability as those on the data qubits. The performance is analyzed by looking at the run time of decoders and the highest error rate the decoder can decoder with before failure to maintain the logical qubit state [50]. The threshold for error rate on the data and ancilla qubits is

the main metric for performance in this research. The neural network is also compared to the Minimum Weight Perfect Matching decoder and achieved similar logical qubit error rates for any given physical qubit error rate. Because of these positive results, this research overall supports the research into machine learning for decoding.

In 2019, Varsamopoulos expanded on original research in decoding small surface codes. The contributions of the research are the comparisons of high-level decoders and low-level decoders, the motivation to find the smallest execution time possible for the neural network-based decoders, and the analysis of how various training datasets affect the learning of the neural network models [49]. Given that multiple error correction cycles are part of the decoding process of this research, the input of a network is multiple error syndromes and their timestamps. Because of this, both fully connected and recurrent neural networks are tested for both high-level and low-level decoders. These decoders are compared with the Minimum Weight Perfect Matching algorithm and each other. The metrics for comparison were the execution times of all the decoders and the threshold of physical qubit errors the decoder would run with and still preserve the logical qubit state of the code. They found that the high-level decoder can achieve a higher threshold for error and has a constant execution time for a given code depth, and that execution time increases linearly with respect to code distance [48]. Then, for the neural network architecture, these researchers observed smaller execution time in fully connected neural network models, but recurrent neural networks were trained easier and have better overall decoding performance due to better training. However, since the initial experiments showed an advantage with high-level decoding, Varsamopoulos et al. further analyzed and elaborated on their high-level decoder, its training, and their implementation, leaving the analysis of low-level decoding somewhat limited.

This research also has conclusions about the best way to choose a dataset to train

neural networks. The best sampling method found in these experiments was using datasets that included various rates of physical qubit error for both training and testing. This was compared to using a dataset with only one rate of physical qubit errors and testing it against datasets with various probabilities of physical qubit errors.

The last piece of important research in neural network decoding was done by Maskara in 2019 [38]. This research had the goal of demonstrating the versatility of neural network decoding in two different topological error correcting codes under three different noise models. Decoding models were created for each of the two types of topological codes at various depths, and with various noise models. The error threshold was then tested and compared to the error threshold of the Minimum Weight Perfect Matching decoder. One distinct advantage of neural network decoding that is demonstrated in this research is the ability of neural networks to have good decoding performance, which was measured in terms of maximum error threshold, while also not needing prior knowledge of the underlying noise model [38]. It is concluded in this research that using neural networks simplifies the process of decoding for various topological codes for use with various noise models.

2.8 Summary

This chapter presented the background information necessary to understand this thesis and the scientific literature that inspired it. Section 2.2 provides the basics of quantum computation and Section 2.3 describes error correcting codes, specifically surface codes, which leads to a discussion of the process of decoding in Section 2.4. Then there is a description of machine learning and various machine learning algorithms in Section 2.5, which can be used for the process of decoding topological error correcting codes. In the literature summarized in Section 2.7, there is a description of

the current relevant research in neural networks used in topological error correcting codes. Although there has been much research in high-level decoders, the research in low-level neural network decoders is sparse. Along with this, either the depolarizing noise model is used, or a noise model that simulates error on ancilla that happen with the same rate as the data qubits. There has not yet been an evaluation of the affect of various levels of ancilla qubits noise on decoding performance. Along with this, most of the evaluation of decoders is based on logical error rate, which does not give much detail in the actual classifying ability of decoders with corrective operations.

III. Methodology

3.1 Overview

Chapter 3 describes the methodology used to address the research questions of Section 1.2 with various low-level decoders of surface codes of depths 3, 5, and 7. First in Section 3.2 the chapter describes research questions and hypotheses. Section 3.3 a description of the evaluation metrics and their importance in comparing decoders. Section 3.4 describes how the problem of decoding is translated to a problem for a neural network. Section 3.5 describes the simulation method used to create the data, along with how the data from the simulation represents surface codes, and how it is transformed into a usable data representation for experiments. Next in Section 3.6 the chapter presents a description of the data simulation and mapping of surface code characteristics to features and labels used for training and testing of decoders. Section 3.6 also includes the details of choosing a machine learning model and how that model is evaluated. In Section 3.7 the design of the experiment with surface code decoders is described. The chapter ends with a summary in Section 3.8.

3.2 Approach

This research aims to address several goals. The first is to demonstrate neural network decoding for surface codes, address some of its challenges and limitations, and assess how well models can perform. The second is to analyze the tradeoffs of neural network decoding compared to a MWPM decoder and a PLUT decoder. The evaluation metrics include number of parameters of algorithm, execution time, accuracy, and F1 score. The final goal is to inject various levels of noise on surface code observations to simulate measurement errors from the ancilla qubits, and compare the effects of the resulting noisy syndrome measurements on the accuracy of decoding of

the various decoders.

3.3 Evaluation Metrics

The metrics for the evaluation of surface code decoders are summarized in this section. These metrics are considered in the experimental design of this thesis, and their use to determine the quality of a decoder is outlined here.

3.3.1 Accuracy and F1

In this research, the "performance" of a decoder refers to how well it classifies, i.e. predicts labels (corrective operations) given the features (error syndromes) of unseen observations. A decoder with low precision would tend to indicate unnecessary, and therefore harmful, "corrective" operations on error-free data qubits. A decoder with low recall would tend to fail to indicate necessary corrective operations on erroneous data qubits. Either of these scenarios would result in the loss of the logical qubit state. In the worst case, a poorly-performing decoder might do both, while a well-performing decoder would do neither.

Either low precision (large false positive count relative to true positive count) or low recall (large false negative count relative to true positive count) can result in low accuracy (large false negative and positive count relative to true positive and negative count). As such, high accuracy is a necessary condition for a decoder to be considered well-performing. However, it is not a sufficient condition, because the context of significant class imbalance such as that present in the decoding problem, accuracy may fail to indicate low precision or low recall with respect to underrepresented classes due to the overrepresentation of the negative class and likely an inflated score due to large amount of True Negatives. Since high accuracy is not a sufficient condition for a well-performing decoder, a performance evaluation that also takes into account F1

is necessary to determine the quality of a decoder.

As discussed in Section 2.5, the F1 metric captures the balance between precision and accuracy of a given model. F1 score is useful in the context of decoding since for a given class label, which represents corrective operations, the more dominant class is 0, which means not to use that corrective operation. Since F1 places emphasis on True Positives, this metric is useful to understand how well a decoder can correctly identify the need for a corrective operations. This makes the F1 score suitable for determining how well a given decoder predicts corrective operations for the test set.

Accuracy is also considered as a means to understand the quality of a decoder. Accuracy takes into account the labels that are correctly and incorrectly predicted in a single observation. One method to obtain an overall accuracy score would be to designate each observation in the test set as correct (all labels predicted correctly) or incorrect (at least one label predicted incorrectly) and then calculate overall accuracy as simply the fraction of observations designated as correct. However, this would not, for example, give different weights to predictions that get only one label wrong for an observation and predictions that get all the labels wrong.

Instead, each observation is given an accuracy value equal to the fraction of labels correctly predicted for that observation. The accuracy of each observation is calculated by comparing the vector of true values with the vector of predicted values by summing the predicted values that agree with the true values and dividing it by the length of the vector. The overall accuracy of a decoder for a given test dataset is then the average accuracy value of that decoder across all observations. In other words if m is the total number of observations in the test set, then

$$\text{accuracy} = \frac{1}{m}\left(\sum_{i=1}^{m} \frac{\text{number of correctly predicted labels of observation } i}{\text{number of labels}}\right).$$

3.3.2 Speed

Quantum hardware cannot ensure that qubits will stay in their intended state as they have a natural tendency to revert back to their ground state, undergo phase shifts, or otherwise drift away from their intended state. Due to this property of qubits, the speed of a decoder is crucial to its practical use. Error correction cycles must be run quickly enough to prevent new errors from appearing alongside the ones that already exist. Qubits used in quanutm computers from IBM lose their intended state in a matter of microseconds [46]. Ancilla measurements must be performed and the error syndrome decoded in less time than it takes other data qubits to lose their intended state so that indicated corrections can be implemented. If decoding is performed accurately, and error correction cycles are run quickly enough, then data qubit errors are corrected fast enough to maintain an intended logical qubit state. Although quantum hardware is improving, leading to qubits that stay in their intended state longer, fast decoders should be created to improve reliability as well. The execution time of the decoder is the limiting factor in the error correcting process, which means that fast decoding is essential to realizing fault tolerant quantum computation.

3.3.3 Scalability

Fast execution time of decoders is necessary to correct errors on surface codes before more appear. However, many decoders achieve a fast execution time for surface codes of depth 3, but not for depth 7 or more. Existing decoders with accuracy sufficient for practical use are $O(n^2)$. Some decoders have achieved $O(n)$ under certain noise models [13], but none have been able to scale well under the fault tolerant case. Any decoder that can decode in linear time is desirable, although constant time approximation algorithms have been proposed as possible decoders. Since large code depths results in a more reliable logical qubit state, both performance and execution

time that scale well are desirable traits of decoders.

3.3.4 Fault Tolerance

The ability of a decoder to maintain the intended logical qubit state can drastically change based on the noise present on the ancilla qubits at the time they are measured. Because of this, the highest possible probability of error that can occur on an arbitrary measurement while still decoding accurately is an important metric to know about any decoding algorithm. The value of a decoder is increased when it can be accurate and fast in decoding when subject to more noise on ancilla qubits, which leads to measurement error. A simulation that is comprehensive in all types of errors that can occur on the surface code would include erroneous corrective operations as well, but this research is limited to injecting noise on the measurements taken by the ancilla qubits.

3.4 Neural Network Task

In terms of a machine learning problem, the problem of decoding is a supervised multi-label classification problem. A multi-label classifier can predict one or many class labels to an observation out of a set of possible labels. The goal is to create a successful model for depth 3, depth 5, and depth 7 surface codes, with an accuracy and F1 score similar to that of the MWPM decoder. Since decoding is a one-to-many problem in which a given error syndrome (feature set) could be caused by multiple possible sets of data qubit errors (label configurations), a training accuracy of 100% is not to be expected. The best any algorithm can do in this problem, even in the context of no measurement errors, is to return the most probable set of necessary corrective error operations. With higher levels of measurement errors, training accuracy is expected to decrease further. In the literature, an accuracy that is

as good as or better than the MWPM algorithm is an acceptable accuracy to achieve before considering other factors, such as execution time [50]. Once that threshold is reached for accuracy, it is known that the capacity of the neural network is at least where it should be and there needs to be no more hidden layers or nodes added.

There is inherent difficulty in multi-label classification due to the class imbalances that are often present. The vector created that represents the labels of the observations has a 0 to represent the absence of a label for the observation. All labels in this dataset are overrepresented by 0 for any given observation, which is common in multi-label classification [51]. Sampling from the dataset to create a training dataset without class imbalances can be done for multi-class problems, but becomes more difficult with multi-label classification [51]. The biggest difficulty to overcome in this case is to prevent a classifier from predicting no labels at all.

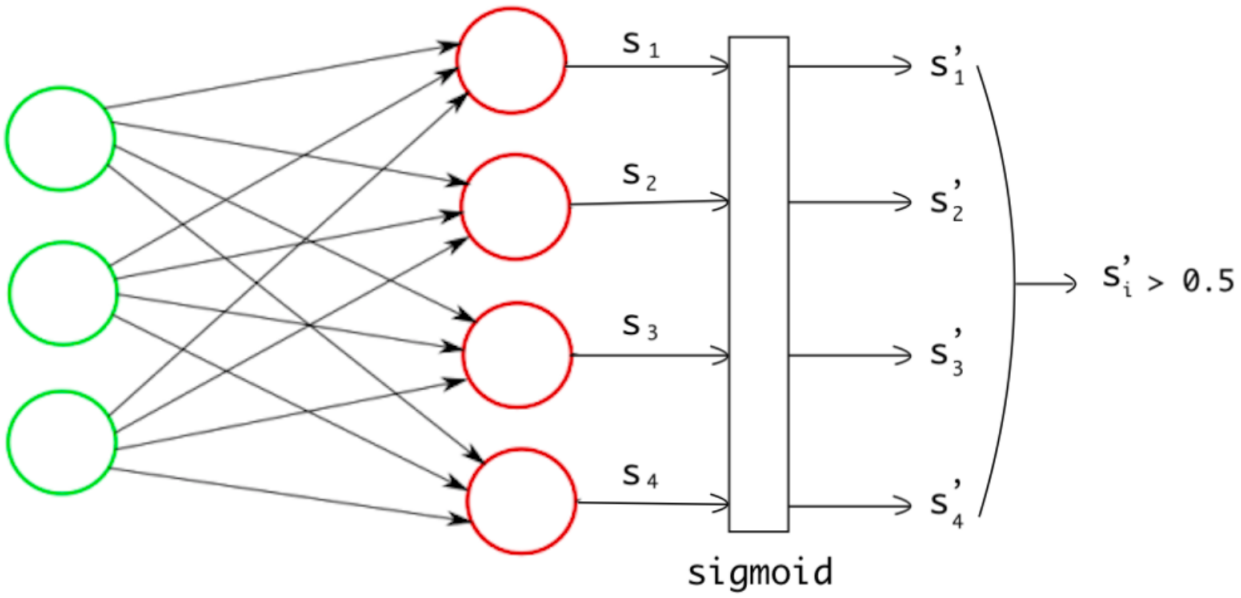


Figure 13: Example of simple multi-label neural network, adapted from [51]

To understand the difficulty of the neural network task, an analysis of the data

is performed. The task of the neural network is to return the corrective action appropriate for the most probable set of qubit and measurement errors given an error syndrome. The relationship between data qubit error configurations and the error syndrome is a many-to-one relationship, as well as the set of most probable corrective operations and a given error syndrome. Generally, the most probable corrective operations is a subset of the data qubit configurations that cause an error syndrome. However, in the case of this research, there is a limited number of data qubit errors that happen on the surface code of depth 3, 5, and 7. In general, the set of data qubit configurations and most probable corrective operations are the same set, except for a case of data qubit errors on the boundary of surface code, where two data qubit errors of the same type occur on the semicircle ancilla qubit and result in two undetectable errors.

To gain insight into the difficulty of the task as a consequence of multiple class label sets producing the same error syndrome, the distribution of possible data qubit errors and their frequencies for each error syndrome is examined. The errors on the data qubits are relatively limited for each surface code depth in this experiment, with depth 3 only having 1 data qubit error, depth 5 having 2 data qubit errors, and depth 7 having 3 data qubit errors. Given these limitations, the observations generated for the surface code of depth 3 have between 0 and 2 corrective operations, those for depth 5 have between 0 and 4 corrective operations, and those for depth 7 have between 0 and 6 corrective operations. Many possible error syndromes are only caused by one combination of data qubit errors.

Since an error syndrome can be caused by more than one configuration of erroneous data qubits, or labels, an analysis of the mapping of syndromes to data qubit configurations is useful to understanding the data. An understanding of these configurations helps to understand the difficulty of the neural network task. A one-to-one

mapping of error syndrome to data qubit configuration is an easy task for the network to learn, while larger numbers of configurations associated with an error syndrome is more difficult. For each code depth, each error syndrome is stored in a lookup table, with one or more data qubit configurations. The counts for how many error syndromes have one, two, three, etc. data qubit configurations is tallied for each code depth in order to generate the data about number of data qubit configurations. The summary of these numbers is displayed in Figures 14, 15, and 16. The number of error syndromes with the given amount. For all of the code depths, error syndromes that map to only one data qubit configuration is the most common scenario. As code distance increases, the fraction of such cases decreases, and the average number of possible data qubit configurations per error syndrome increases.

```
Number of syndromes with 1 data qubit configurations : 20
Number of syndromes with 2 data qubit configurations : 4
```

Figure 14: Number of error syndromes with 1 or more data qubit configurations for depth 3

```
Number of syndromes with 1 data qubit configurations : 1583
Number of syndromes with 2 data qubit configurations : 480
Number of syndromes with 3 data qubit configurations : 20
Number of syndromes with 4 data qubit configurations : 36
Number of syndromes with 5 data qubit configurations : 4
Number of syndromes with 9 data qubit configurations : 1
```

Figure 15: Number of error syndromes with 1 or more data qubit configurations for depth 5

```
Number of syndromes with 1 data qubit configurations : 263645
Number of syndromes with 2 data qubit configurations : 88186
Number of syndromes with 3 data qubit configurations : 4634
Number of syndromes with 4 data qubit configurations : 10090
Number of syndromes with 5 data qubit configurations : 1097
Number of syndromes with 6 data qubit configurations : 465
Number of syndromes with 7 data qubit configurations : 39
Number of syndromes with 8 data qubit configurations : 368
Number of syndromes with 9 data qubit configurations : 8
Number of syndromes with 10 data qubit configurations : 88
Number of syndromes with 12 data qubit configurations : 4
Number of syndromes with 13 data qubit configurations : 25
Number of syndromes with 14 data qubit configurations : 33
Number of syndromes with 15 data qubit configurations : 33
Number of syndromes with 16 data qubit configurations : 13
Number of syndromes with 18 data qubit configurations : 16
Number of syndromes with 19 data qubit configurations : 4
Number of syndromes with 26 data qubit configurations : 4
Number of syndromes with 28 data qubit configurations : 8
```

Figure 16: Number of error syndromes with 1 or more data qubit configurations for depth 7

These configurations are generated by creating a full lookup table for each code depth in surface code,with data qubit limitations of 1 data qubit error on depth 3, 2 data qubit errors on depth 5, and 3 data qubit errors on depth 7. In which each unique error syndrome is an entry in the lookup table, and the contents are a list of the data qubit configurations that create that error syndrome. Given a full lookup table, the number of error syndromes with one or more data qubit configurations for each code depth is generated in Algorithm 1:

Although some error syndromes on the surface code are caused by different configurations of data qubit errors, the majority of the syndromes are only caused by one such configuration. However, decoding remains a many-to-one problem, so the goal of the neural network is to learn to return the most probable error chain for a given set of features. Although this is a one-to-many problem, the neural network's task can instead be thought of as a function approximator to return the most probable set of labels that can correct the faulty surface code. Because of this one-to-many

Algorithm 1 Prints Number of Error Syndromes with 1 or More Data Qubit Configurations

Result: Returns Number of Error Syndromes with 1 or More Data Qubit Configurations

Input: Full lookup table L

```
 Initialize list syndromeCounts
 for syndrome in L do
 |  syndromeCounts += number of entries in syndrome
end
for i in range(100): do
 |  if i in syndromeCounts then
 |  |  print(count of i in syndromeCounts)
 |  end
end
```

problem, 100% accuracy is not achievable for a decoder.

Another factor to consider in training and testing is the high level of specificity required to correct the individual data qubits while also considering that erroneously failing to correct a qubit is just as bad as erroneously "correcting" an error-free data qubit. A main advantage of this type of decoding is that the operations happening on the surface code are specific and understandable. However, because there are many possible configurations of data qubit errors that can generate the same error syndrome, getting a prediction wrong by chance can simply create more errors on the surface code. Another disadvantage is that many possible label combinations exist. These combinations increase very quickly when the size of the surface code scales up, which means that data generation and training requires substantially more computation time for larger codes and may not be practical, depending on the implementation.

3.5 Data Simulation

As of now, the hardware does not exist to support thorough experiments on surface codes. It is common to use theoretical computation or to simulate data using

noise models instead. For the task of decoding, a custom made simulator was made to generate the error syndrome measurements and the labels needed for the machine learning tasks. The simulator itself can simulate data for any arbitrary depth of surface code. The simulator can generate error syndromes given user defined probabilities of error on the data qubits and on the ancilla qubits. Changing variables and generating data for given error rates is very easy. The flexibility provided was the main advantage driving the decision to use a custom made simulator rather than IBM's Qiskit library.

Various tests are used with the simulation to validate that it works as intended. These tests were done for all surface code depths used in this research. The tests were run by inputting data qubit errors on surface code, and validating that the output is correct from the simulation. The truth values of the tests were made by hand, and the correctness of the simulator is determined by comparing the true results and the simulator results in a test program. All configurations of a single data qubit error on depth 3 code could be validated since there are only 28 configurations with a single data qubit error. These 28 data qubit configurations include each of the three operators, X, Y, and Z happening on every one of the nine data qubits, plus the case of now data qubit errors on the code. Depth 5 and depth 7 surface codes included tests that validated the simulation by choosing a sampling of generated correct error syndromes for X, Y, and Z errors and their combinations, as well as syndromes generated from boundary and corner data qubits. The simulator passed all these tests.

The simulation is run to ensure that for each depth of surface code there is a syndrome that represents all combinations of data qubit errors. The number of data qubit errors on the system is kept below 10% of the total data qubits, since the threshold of probabiliy of data qubit errors is around 11% [54], as explained in Section 2.3.1.

Depth 3 code has up to 1 error, depth 5 code has up to 2 errors, and depth 7 code has up to 3 errors. To test the effect on depth on decoder effectiveness these maximum number of data qubits errors are chosen so that the training of the decoders is similar and the results at various depths are comparable. The dataset is made by generating all possible configurations of X, Y, and Z error for a fixed number of maximum data qubit errors on surface code and the corresponding error syndrome.

The generation of this data is used with the custom-made simulator that initializes a surface code circuit of a given depth. To generate the combinations of maximum data qubit errors, the maximum number of errors (n) is inputted. The possible errors are always X, Y, and Z errors. From 1 to n, the possible combinations of X, Y, and Z errors with replacement is generated. For each of these possible combinations, the possible locations of the data qubits is generated. These locations are first generated for each type of error, since if there are two X errors on the code, then the number of configurations given m data qubits of just these two X errors is $\binom{m}{2}$. This process is encapsulated in the functions $xLocations$, $zLocations$, and $yLoactions$, which can find those configurations for a specific type of error on a surface code circuit. Once these list of locations of X errors, Y errors, and Z errors is complete, then the list of the various errors are multiplied together since X, Y, and Z errors act independently. These location configurations are stored in the variable $TotalCombos$. For each configuration in $TotalCombos$, the errors are injected on the surface code circuit and the syndrome is retrieved with a GetSyndrome function. These syndromes, along with their corresponding data qubit configuration, are all returned at the end of the simulation.

Algorithm 2 Data Simulation to Return all Data Qubit Error Combinations and Corresponding Error Syndrome

Result: Returns syndromes for all data qubit configurations of number of errors n and under

Initialize surface code circuit = c

Initialize number of data qubits = n

Initialize list of errors combos = [x,y,z]

Initialize empty list = TotalSyndromes

for *i in n* **do**
| combos += [x,y,z] combination i

end

for *errors in combos:* **do**
| $\text{combos}_x = circuit.xLocations(errors)$ ▷ list of all locations of data qubits with $\text{n}_x xerrs$
| $combos_z = circuit.zLocations(errors)$ ▷ list of all locations of data qubits with $\text{n}_z zerrs$
| $combos_y = circuit.yLocations(errors)$ ▷ list of all locations of data qubits with $\text{n}_y yerrs$
| $TotalCombos = combos_x * combos_z * combos_y$;
|
| **for** *i in TotalCombos:* **do**
| | circuit.addErrors(i)
| | TotalSyndromes += circuit.GetSyndrome()
| | circuit.eraseErrors()
| |
| **end**

end

return *TotalSyndromes*

One important piece of information to note is that this simulation results in some duplicate data qubit configurations. This is due to Y errors being a combinations of X and Z errors, so if both an X error and Y error happen on a qubit, the affect is the same as a Z error. These duplicates are removed in data preprocessing. A summation of the size of the dataset created without duplicates is belowin Table 1:

	Depth 3	Depth 5	Depth 7
Dataset Size:	28	2776	508180

Table 1: Summary of sizes of datasets

3.6 Machine Learning Pipeline

In order to develop the machine learning model used for decoding, a machine learning pipeline is done. Machine Learning pipelines describe the process of extracting meaningful knowledge from data through a series of sequential steps that include data extraction and feature engineering, model training, and model evaluation.

3.6.1 Feature Selection and Data Prepocessing

For the purpose of machine learning, each observation has a 1-dimensional input vector that represents the error syndrome. Each element of the input vector represents a measurement from an ancilla qubit. These measurements are the features of the data used in machine learning. The value of each feature can be 0,1, where 1 is an error detection event from the ancilla and 0 is the absence of a detection event.

The dimension of the input vector is determined by the number of ancilla qubits on each surface code, which depends on the depth of the code. The number of ancilla qubits, and therefore the dimension of the input vector is $depth^2 - 1$. For experiments on small surface code of depth 3, there are 8 columns of the input vector with each

element having a value of 1 or 0, which represents the reading from an ancilla. When the depth is 5 for the surface code, there are 24 values in the vector. When the depth is 7 for the surface code, there are 48 values in the input vector.

Generally, this can be considered a multi-label classification problem. This means that a given observation can have an output vector with more than one class with a positive class label for a given observation. The classes represent the individual data qubits and the type of error that needs to be corrected on it. The class labels are a 0,1 value that represents a corrective operation to be performed for that class or the absence of a corrective operation. The output is a mutlilabel output vector of binary values, where each element represents a class. Because either X or Z errors have to be corrected on data qubits, and the number of data qubits on the surface code is the $depth^2$, $2 * depth^2$ error states exist for each observation in the machine learning task. Surface code of depth 3 has 18 classes and surface code of depth 5 has 50 classes and surface code of depth 7 has 98 classes. For each observation, the simulator generates a bitstring where each element of the bitstring represents both a data qubit location and type of error to correct for the data qubit. Figure 17 shows the labels of the data qubits for surface code of depth 5.

For example, an X data qubit error in the very top left data qubit would be denoted by 'X00'. This list of data qubits is then made into a vector for each observation which is then ready to be the multilabel output vector used for model training. Each element of the multilabel output vector represents a possible corrective operation to be performed on a data qubit, where a 1 indicates a corrective operation to be performed and a 0 indicates the absence of a corrective operation. An observation of depth 3 surface code with a data qubit error to be corrected on "X00" would be transformed into multilabel output vector "1000000000000000" to use in machine learning. Figure 18 shows how this representation of features and labels translates

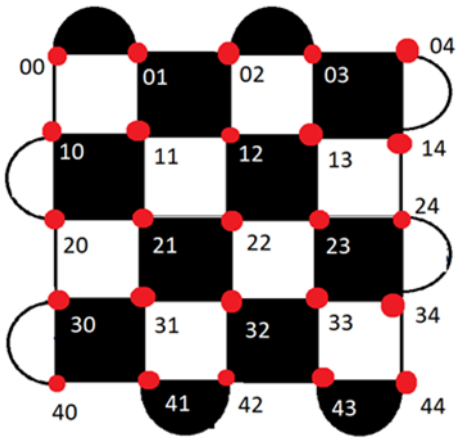


Figure 17: Labels on surface code of depth 5. The red dots indicate the data qubits on the surface code, and the numerical notation represents the location of the data qubit. The first element in the notation is the row, and the second is the column of the data qubit.

from a surface code grid. Table 2 summarizes the label and feature combinations of each surface code depth.

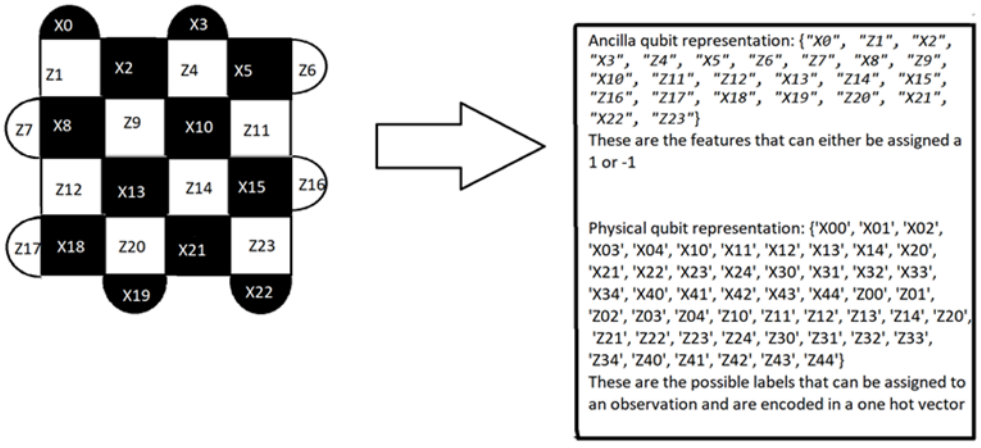


Figure 18: Example surface code of depth 5 and the data representation of the features and labels. The ancilla qubits are shown as the corresponding features. The corrective operations, which are an X or Z operation on a data qubits, are shown as the class labels.

	Depth 3	Depth 5	Depth 7
Input Features:	8	24	48
Number of Possible Input Feature Vectors:	24	2124	368760
Number of Classes:	18	50	98
Number of Possible Ouput Vectors:	28	2776	508953

Table 2: Summary of input and output length and total number of possible vectors given data qubit constraints.

The characteristics of a dataset help to know which evaluation metrics are most suitable for model testing and how to best train a model that can generalize to predict unseen data in a meaningful way. Depending on the depth of the surface code, there exists between 1 and 3 data qubit errors which limits the number of class labels for a given observation. Therefore, zero is represented more often than a one in the multilabel output vector that is made from the data engineering. A major obstacle to overcome is to prevent the machine learning algorithm from not predicting any class labels at all. Since most of the classes are represented by a 0 in the data, indicating its absence, a classifier that does not predict any class labels at all still results in an accuracy score that does better than chance. In order to understand the accuracy score that represents the accuracy if a zero is assigned to all classes, a quick test is run for each test set, which is shown in Figure 3. This simply gathers the accuracy for each test set when the predicted values are all zero. For It is important in each of these cases that the machine learning algorithm has a higher accuracy score than these three values in order to ensure that it is doing better than simply predicting no labels for each observation.

	Depth 3	Depth 5	Depth 7
Features:	0.9259	0.9484	0.9600

Table 3: Accuracy for zero classifier of depths 3, 5, 7

Since there can be multiple classes with positive class labels for a given observation for each code depth, it is important to understand the balance of the positive class labels (1) and negative class labels (0). Although each class label is overrepresented with 0s in the training set, a test is run to see how many times each label is given a 1 within the training set. If some classes have more positive ("1")values than other classes in the training set, then that indicates an imbalance in the classes of the data set. A table with the mean and standard deviation of number of positive class labels for each class is shown in Table 4. The mean and standard deviations are used to estimate the coefficient of variation. In general, a coefficient of variation < 1 indicates a low variance of values, and a coefficient of variation > 1 indicates a high variation in the values [36]. For the positive class labels of surface codes 3, 5, and 7, all of the coefficients of variation are < 1,indicating little variance in the number of classes with a positive label. The class balance is even enough that each label is represented fairly in training and testing, and won't lead to one label being weighted more in evaluation of model performance.

	Depth 3	Depth 5	Depth 7
Mean:	1.555	116.72	16612.7959
Standard Deviation:	0.4969	4.4721	54.5527
Coefficient of Variation Estimate:	0.3206	0.0366	0.0033

Table 4: Summary of class label mean and standard deviation

3.6.2 Model Selection and Design

Although neural networks are used in this research and in the majority of the research in the literature of decoders that are not based on graph algorithms, machine learning models that are not neural networks were considered as well. The models used in these preliminary tests were chosen because there are few machine learning algorithms beside neural networks that can be used for a multi-label classification problem. In choosing a model for testing, several criteria for all code depths are considered. These include:

- Training Time
- Prediction Performance

The model that best fit these criteria is the fully connected neural network design. The MLkNN model needed 10 hours for training the depth 7 decoder, the One vs. Rest with SVM needed over 24 hours of training time for the depth 7 decoder, and the RandomForest model needed over 12 hours of training time, which was stopped after a memory error. In these cases, the large training times make testing and tuning models more time expense, and are a disadvantage for practical use. These run times are also more likely to become larger with code depths beyond 7, which makes these methods less likely to be used in future experiments with code depths beyond 7.

The training time and memory usage is the biggest concern in choosing which method to compare against algorithmic decoders, and due to the large training times for all classifiers except for the neural network, training time became the main factor in deciding which model to use. Another concern is that these decoders did not learn to predict any class labels at all for depth 3 surface code. Instead, these models produced a classifier that did not predict any positive labels for any classes. The summary of the F1 results from depth 5 predictions and the time it takes to train

a model for depth 7 is presented in Table 5. Depth 5 F1 scores indicate similar performance for depth 5 decoding, and is the metric for the predicted performance of that decoder for the rest of the experiments. The training time is presented as well, and is where the neural network decoder has a distinct advantage. This made it the most practical decoder to use in the experiments of this research, and for code depths beyond 7.

	Depth 5 F1 score	**Depth 7 Training Time**
NN	0.692	2.5 hrs
Random Forest	0.615	>12 hrs
One vs Rest	0.630	>24 hrs
MLkNN	0.727	>10 hrs

Table 5: Summary of criteria to pick ML model. NN = neural network. MLkNN = Multi-label k-Nearest Neighbors

The neural network is the machine learning model that best fits the criteria listed, mostly due to training time considerations. MLkNN has the highest F1 for depth 5, the training time needed is not practical for these experiments. Although the performance on depth 5 code indicates that all models can predict corrective error, the neural network has the fastest training time for depth 7 surface codes, and the best depth 5 performance in F1 score. Since the code depths used in this research are small, and larger codes are likely needed in practical quantum computation with surface code, classifiers that can be used with depths 3, 5, and 7 are essential.

Neural networks that are more complex than multilayer perceptrons have several obstacles as well. Convolutional neural networks (CNNs) are a possible option for network design since the grid shape of the surface code could be suitable for CNNs. This could be implemented after altering the surface code to have "dummy" ancillas along

the boundaries that always measure $|0\rangle$, $sothattheshapeisusablebyaCNN.However, thisonlyappliestolo$

The number of hidden layers, nodes per layer, optimizer, and learning rate of the ANNs were chosen based on a Sklearn's GridSearchCV. This module uses list of values to choose from for each parameter and GridSearchCV trains and tests on all combinations of the values of the hyperparameters through cross-validation. The final models were chosen based on their validation accuracy compared to MWPM, which is the decoder that always returns the most probable errors for a given syndrome. The parameters tested in this search are shown in Table **??**. parameters chosen for the final model are shown in Table 7. Along with these parameters, binary cross entropy is used as the loss function because it is the only loss function available for multi-label classification tasks.

	Depth 3	Depth 5	Depth 7
Hidden Layers:	1, 2, 3	2, 3, 4	3, 4, 5, 6
Nodes per layer:	16, 32, 64, 128	100, 200, 250, 300	250, 300, 350, 400
Learning Rate:	.001, .01, .05	.001, .01, .05	.001, .01, .05
Activation Function:	ReLU, tanH	ReLU, tanH	ReLU, tanH
Optimizer:	SGD, Adam	SGD, Adam	SGD, Adam

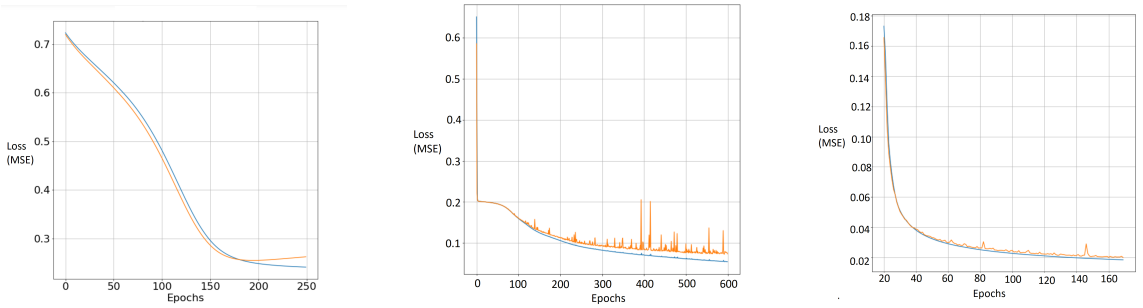
Table 6: Summary of Model parameters. SGD = "Stochastic Gradient Descent." ReLU = Rectified Linear Unit

	Depth 3	Depth 5	Depth 7
Hidden Layers:	2	4	4
Nodes per layer:	32	250	400
Learning Rate:	0.05	0.05	0.05
Activation Function:	ReLU	ReLU	ReLU
Optimizer:	SGD	SGD	SGD

Table 7: Summary of Model parameters. SGD = "Stochastic Gradient Descent." ReLU = Rectified Linear Unit

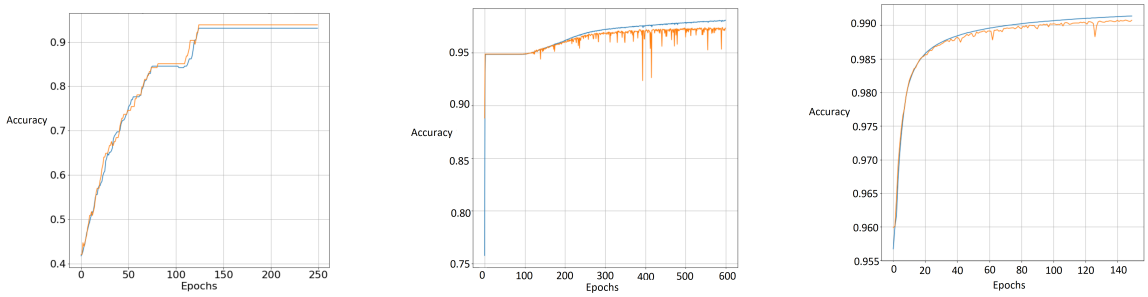
The final layer determines the probability that one of the possible labels is predicted for a given observation. A sigmoid function is used as the activation function for the final layer, which returns a value between 0 or 1, with a class label being closer to 1 if when the model determines that it is more likely to be positive for a given observation. Since each label is given some decimal value, there needs to be a threshold for whether or not to assign the label to the observation or not. For example, this means that if the threshold value is .5, then output node values of over .5 are transformed into a 1 to signify that the label is predicted for the observation. Output node values of under .5 would be assigned a zero. Thresholds for any given model may vary depending on which threshold yields the best performance. Once the output is transformed to this format, it is compared to the labels of the test set using the metrics of F1 score and accuracy. To determine the appropriate threshold for each model, threshold values from 0.0 to 1.0 are tested in .1 increments. The value that has the best F1 score and accuracy score are chosen as the final threshold for the model. Since F1 score represents a balance between precision and accuracy, it can better represent how well the model is predicting each class label for each observation, and is a more important consideration than accuracy in choosing a threshold value.

Along with this, the best training performance is achieved without batches for the networks. The depth 3 network uses 200 epochs for training, depth 5 uses 600 epochs for training, and depth 7 uses 150 epochs for training. The number of epochs used is based on training and validation accuracy and loss curves made in preliminary testing to determine the number of epochs to use before overfitting. Each decoder uses a batch size of 32. During preliminary testing, the loss per epoch and accuracy per epoch is recorded for both training and validation data. An example of a loss curve for each depth are shown in Figure 19 and the accuracy curves are shown in Figure 20. Since both the validation and training accuracy and loss do not diverge with the epochs listed, then model did not overtrain with the epoch used to train the model.



(a) Depth 3 (b) Depth 5 (c) Depth 7

Figure 19: Training and validation loss curves of depth 3, 5, and 7



(a) Depth 3 (b) Depth 5 (c) Depth 7

Figure 20: Training and validation accuracy curves of depth 3, 5, and 7

Regularization is not used with the models. Both dropout and L2 regularization were tested in preliminary stages and both resulted in decreased F1 and accuracy. Overfitting is not a considerable challenge for this dataset and networks. The loss and accuracy curves did not diverge significantly for depth 5 or depth 7 decoders given increased capacity or epochs. Increasing capacity generally resulted in better performance for all models, and so the main goal is to balance training times that are practical for experiments with a large capacity and epoch number to increase model performance.

3.7 Experimental Design

This section describes the experimental design used to evaluate and compare the performance of the various decoders. The intent of these experiments is to test the research hypotheses stated in Chapter I.

3.7.1 Significance Test

Along with the performance (effectiveness as measured by F1 score and accuracy) and efficiency (execution time) gathered from each fold, a test of statistical significance is conducted for the last fold of each code depth at each measurement error level. The test evaluates the difference in performance between decoders under various depth and measurement error configurations. Several statistical tests are potentially applicable, including the paired t-test with k-fold cross validation, the 5x2 cross validation with paired t-test, Matthew's Correlation Coefficient (MCC), and McNemar's test.

The paired t-test with k-fold cross validation has been used in previous machine learning research, but is generally not recommended because the use of this test with k-fold cross validation does not satisfy the assumption of independence of samples. This results in higher Type I error, i.e. the indication of statistical significance when

there actually is none [15].

A variation of typical k-fold cross validation is 5x2 cross validation. This method performs a paired t-test on the results of two classifiers for 2-fold cross validation. 5x2 cross validation has relatively low Type I error, but it is generally used to compare the performance of two machine learning classifiers. The use of only half of the dataset for training would likely decrease performance in a machine learning decoder, but would not affect the performance of an algorithmic decoder such as MWPM. Thus, this test would tend to overestimate the performance advantage of algorithmic decoders over machine learning decoders.

MCC is related to the chi-square statistical test and finds application in machine learning research [9]. However, it is used to describe classification only for one classifier. Because the intent of a significance test in this work is to compare the classification performance between decoders and understand if there is a significant difference in how they are decoding, this metric is not used to compare decoders.

McNemar's test is used in this research because if the test statistic is significant, it provides evidence that two different decoders are performing differently. This is preferred over the paired t-test with k-fold cross validation, which results in high levels of Type I error. McNemar's is also preferred over the 5x2 cross validation because the 5x2 cross validation likely is not a good comparison tool between a neural network model that requires training and the MWPM algorithm. MCC does not directly create a test statistic to compare the performance of two decoders, which is preferred in the test statistic. McNemar's is a statistic that can compare performance based on different error rates of classifiers, and also has acceptable Type I error.

In the experimental run, McNemar's is applied to the last fold in k-fold cross validation. The test set in the last fold of depth 5 and depth 7 surface code provides enough data to perform McNemar's. This test cannot be performed on depth 3 code

due to a small dataset. McNemar's test is applied for each class label and is used to compare the effectiveness and efficiency of the neural network decoder to that of the MWPM decoder and to that of the PLUT. This creates a χ^2 value with one degree of freedom.

3.7.2 Decoder Evaluation

The neural network decoders are compared against the MWPM decoder and a PLUT decoder at each depth. The MWPM decoder runs algorithmically and needs no training. The MWPM algorithmic decoder is based on an implementation that works with IBM's Qiskit package [16]. It always returns the most probable data qubit error chain given an error syndrome with no measurement errors. This is considered to be a "good" decoding, although it does not reach 100% accuracy. If the neural network decoder performs as well as or better than the MWPM decoder in F1 and accuracy, then it can be concluded that the neural network decoder is learning well. If it is doing worse than the MWPM algorithm, then the tradeoffs between execution time and performance should be considered.

The neural network decoder is also compared against a PLUT, which is populated from error syndromes and corresponding corrective operations from the same training set that is given to the machine learning models. The input and output of the PLUT are the same that are used for neural network training. The PLUT decoder is then tested on the test set, which is unseen data. This is a custom-made partial lookup table. Although the smaller depths of surface codes used in these experiments could easily have a full lookup table that performs optimally, in larger codes the data requires simulations that generate data slowly. For the large code depths that would be necessary for real-life surface codes, training of a neural network or populating a lookup table could only be done with some of the combinations of error syndromes and

data qubit errors, but not all. The partial lookup table is a very naïve decoder, and it represents the lower end of decoding performance. If the neural network decoder performs worse than the partial lookup table in F1 and accuracy, then it can be concluded that the neural network decoder is very poor.

Cross validation is used in order to reduce variance in performance results. Five-fold cross validation is used for the depth 3 and depth 5 surface codes, while 3-fold cross validation is used for the depth 7 code due to the long training times with 5-fold cross validation. At each fold, the neural network is trained and the partial lookup table is populated based on the training data of that fold.

All of the decoders are tested on the test set of each fold, and the F1 score, accuracy, and execution time is recorded for each decoder. In the case of the neural network, the Area Under the Receiver Operating Curve is recorded for each label, as well as the micro-average F1 and accuracy for each fold. This process is done for 0, 0.01, 0.03, and 0.05% error on the elements of the ancilla feature vector. Either the micro-average or macro-average could be presented for this problem since the classes are balanced. Micro-average is presented for this problem since this type of averaging weighs each prediction equally in its calculation.

One way to evaluate decoders is to analyze how their performance scales with increasing depth of surface code. Although only small code depths are used in this research, the results of such experiments can indicate trends in the performance of various approaches to decoding surface codes. Since F1 score and accuracy are both metrics for decoding performance, they are both used in this evaluation of scalability.

Experiments with varying levels of measurement error are performed to investigate the role of noise on ancilla qubits on decoder performance. In general, measurement errors may or may not be included in surface code experiments, but it is more realistic to simulate them even though there is a decrease in accuracy. The datasets used in

these experiments are the same used for the experiments without noisy measurements, except that each of the features can have their value flipped with probability p. Accuracy and F1 score are used as the metrics of influence of noise, since execution times for decoders mostly remain the same with varying noise levels on the measurements. The accuracy and F1 is analyzed for each decoder at measurement noise error rates of 1%, 3%, and 5%. These rates are the probability p that feature values are flipped in the dataset. These noise values are the lower end of the read-out error rates of IBM's larger machines. The noise values are chosen to be on the lower end of current real-world read-out values since quantum hardware is likely to improve drastically by the time surface codes could be implemented with real quantum hardware.

McNemar's test is done with each experiment to verify that the decoders have dissimilar performance on the testing data. Since the performance of the neural network decoder compared to algorithmic decoders is the focus of this research, McNemar's test is performed to compare the neural network decoder to the MWPM algorithm and the partial lookup table. The p-values generated from this test determine the level of statistical significance with which the conclusion may be drawn that decoders differ in their classification errors. A decoder with better F1 and accuracy and a p-value that provides evidence that decoder performance is different than other decoders can be confirmed to be better than the other decoders.

To evaluate if the neural network decoder is decoding better than chance, the neural network decoder is executed with the custom-made simulation. This simulation injects 1 random data qubit error on the surface code, which is followed by corrective operations from the neural network decoder. This process is repeated until the number of data qubit errors is equal to the depth of the surface code. Since the number of data qubit errors that result in a logical operation are at least the equal to code depth, ending the simulation once the errors are equal to the code depth is a strict measure.

If the neural network is performing as well as chance, with a decoder doing as well as chance being defined as a decoder that performs one random corrective operation per syndrome measurement, then the neural network would likely only get through *depth* or *depth*+1 number of rounds before the simulation ends. This is because a decoder that is simply guessing for a corrective operation has a $1 \div depth^2$ chance of being correct. If the neural network decoder does better than this, there is evidence that the neural network decoder is performing better than chance.

3.8 Summary

The multi-label nature of this data makes the translation of surface code to classification problem more complex than binary or multi-class classification as explained in Sections 3.4 and 3.5. The metrics chosen in Section 3.7.2 to determine decoder effectiveness biased toward high accuracy values that may be misleading, these considerations are taken into consideration during result analysis. Inherent class imbalance makes it difficult to interpret decoder effectiveness, but the commonly used MWPM algorithm and naïve PLUT decoder are decoders that can be used for comparison when addressing neural network effectiveness from trained neural network models described in Section 3.6. The comparisons of these decoders to each other is confirmed through McNemar's test, which can be used to confirm if performance differences are significant. With the experiments outlined in Section 3.7, these comparisons are used with the carefully chosen evaluation metrics that can be used to test the research hypotheses centered around decoder effectiveness with increasing depth and probability of measurement error. The results can also be used as evidence to support or disprove hypotheses centered around decoder execution time and neural network ability to perform better than chance.

IV. Results and Analysis

4.1 Overview

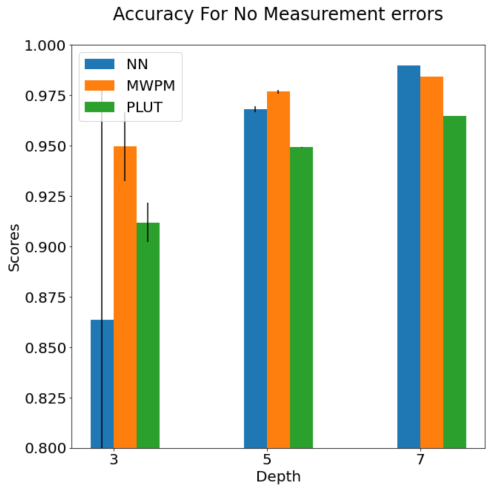
This chapter presents and analyzes the results of the computational experiments described in Chapter III. First, both the accuracy and F1 scores are presented as performance metrics, for all decoder types, with 0, 1, 3, and 5% probability of measurement errors at depths 3, 5, and 7. Specifically, the effects of depth and noise level are presented in Sections 4.2 and 4.3, respectively. The average execution times of the decoders at various depths are discussed in Section 4.4. An analysis of the statistical significance of the experimental results, as measured by p-values from McNemar's test, is presented in Section 4.5. Section 4.6 is a discussion of the neural network-based decoder's performance specifically, including a comparison to the performance of random guessing. The chapter ends with a summary in Section 4.7.

4.2 Performance in Relation to Code Depth

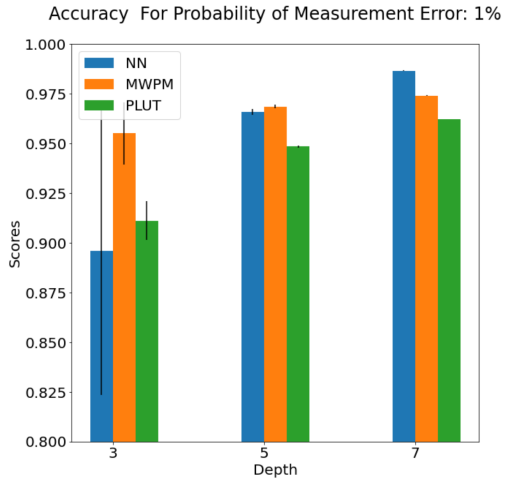
The results are presented here in the form of graphs of F1 and accuracy scores of all three decoders for probabilities of measurement error of 0, 1, 3, and 5%. Tables of the values in the graphs are included in Appendix A.

4.2.1 Accuracy Results and Analysis

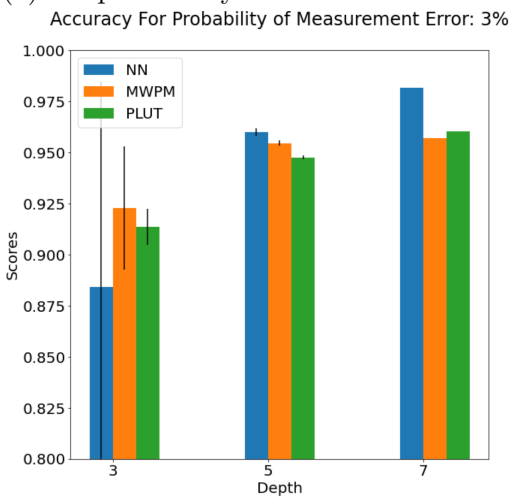
Figure 21 and Tables 8–11 (in Appendix A) summarize the accuracy results for the experiments including the various levels of measurement error. In the absence of measurement error (Figure 21a), all of the decoders increase accuracy for increasing code depths. One explanation for this is that the number of negative class labels increases along with the increasing number of classes that occur with increasing code depths. For depth 3 and depth 5, the MWPM decoder has the highest accuracy



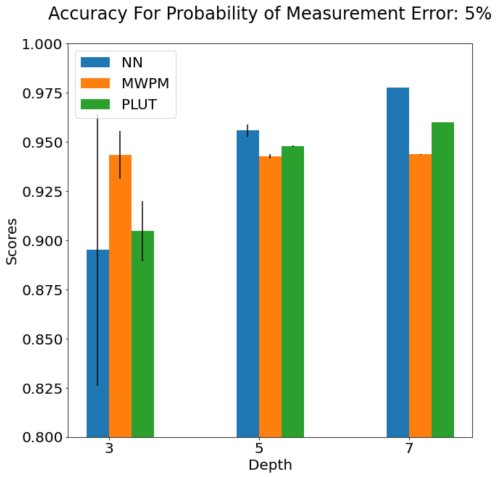
(a) 0% probability of measurement error



(b) 1% probability of measurement error



(c) 3% probability of measurement error



(d) 5% probability of measurement error

Figure 21: Summary of accuracy for increasing code depths. Error bars represent standard deviations.

values. For depth 7, the neural network has the highest accuracy value, although the difference between the means is small. The standard deviation also decreases with increasing code depth, which can be explained by larger code depths having larger datasets. The larger dataset results in a test set that has more observations as well, which results in less variance in the accuracy of each fold.

The last piece of information to note about Figure 21a is that the depth 3 neural network decoder performed much worse than the PLUT decoder. Considering the fact that the PLUT decoder is a very naïve decoder that has a tendency to predict no corrective operations for syndromes that it has not seen before, and the accuracy of the depth 3 neural network decoder is below that of the PLUT, the neural network decoder is performing very poorly. The poor performance of the depth 3 neural network decoder is most likely due to its small training data size, suggesting that a training set made with all the possible syndromes made from 1-qubit error is not sufficient to train a successful decoder.

The results of increasing code depth at noise level of 1% (Figure 21b) have many of the same patterns as the results for the case with no measurement errors. In particular, the accuracy increases for all decoders at increasing code depths, and the variance decreases with increasing code depths.

The results of increasing code depth at noise level of 3% (Figure 21c) have some of the same patterns as the results with no measurement errors. At this noise level, the neural network decoder is the most accurate decoder for depths 5 and 7, rather than just for depth 7. The MWPM decoder is still the most accurate decoder at depth 3, but the neural network decoder is slightly more accurate at depth 5 and significantly more accurate at depth 7 when compared to the MWPM algorithm. At depth 7, the MWPM algorithm is the worst decoder in terms of accuracy.

At 5% probability of error (Figure 21d), the neural network decoder is again

the most accurate decoder for code depths 5 and 7. The neural network and PLUT decoders increase in accuracy with increasing code depths, while the MWPM decoder maintains mostly constant accuracy with increasing code depth. The MWPM decoder is the least accurate decoder for both code depths 5 and 7, and is the most accurate decoder for code depth 3.

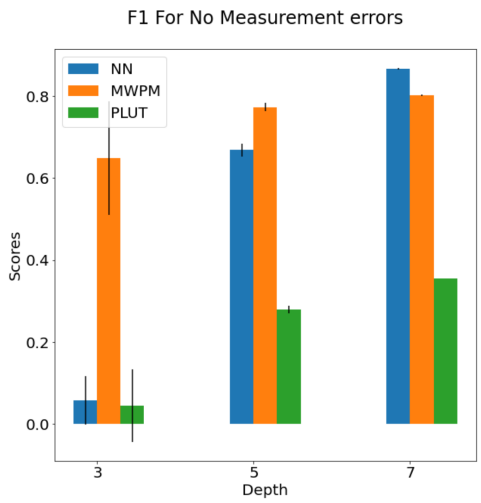
4.2.2 F1 Results and Analysis

Figure 22 and Tables 12–15 (in Appendix A) summarize the F1 results for the experiments including the various levels of measurement error. For the case with no measurement errors (Figure 22a), F1 score increases as code depth increases for all decoders. MWPM performs the best for depths 3 and 5, while the neural network decoder has the highest F1 score for depth 7. The neural network and PLUT have very poor scores for depth 3, indicating poor learning for the neural network.

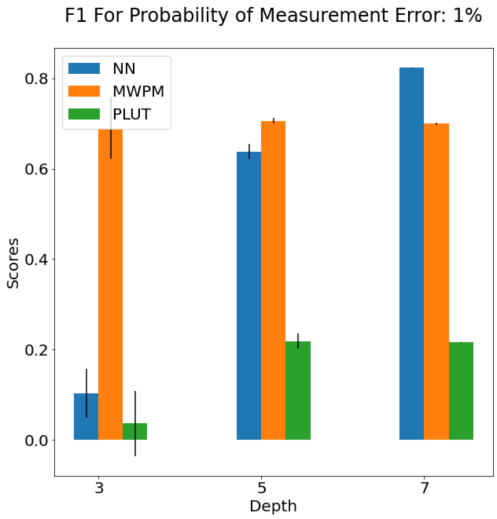
With noise levels at 1% (Figure 22b) the MWPM decoder has the highest F1 accuracy for depth 3 and 5 codes, as it does for the experiment without measurement error. However, the F1 score for MWPM does not increase when there is 1% probability of measurement error, and instead stays constant. The PLUT's F1 score also stays constant for depths 5 and 7. The neural network's F1 score continues to increase with increasing surface code depth, and the neural network decoder's F1 score for depth 7 is significantly above the MWPM decoder.

The results of increasing code depth at noise level of 3% (Figure 22c) have some of the same patterns as the results with 1% probability of measurement error. However, PLUT decoders increase for depth 5, and then decrease for depth 7. This may be because the error chains that these decoders are predicting are longer for depth 7 code, and results in larger errors for individual incorrect observations.

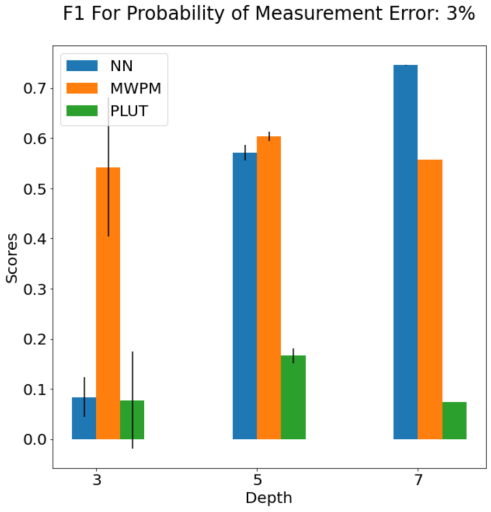
The results for 5% probability of measurement error (Figure 22d) are similar to



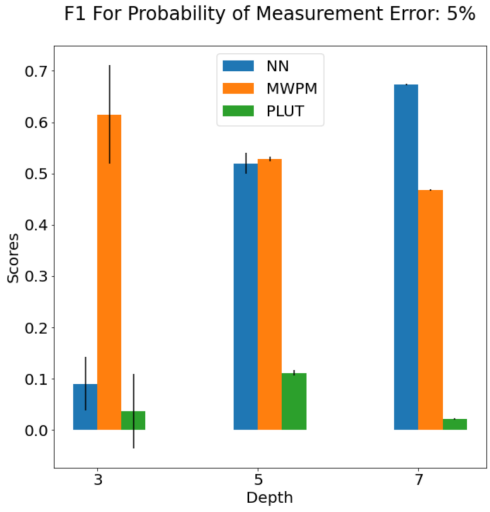
(a) 0% probability of measurement error



(b) 1% probability of measurement error



(c) 3% probability of measurement error



(d) 5% probability of measurement error

Figure 22: Summary of F1 for increasing code depths. Error bars represent standard deviations.

the results of 3% probability of measurement error for increasing code depths. The difference in means between the depth 5 neural network and MWPM algorithms are similar, but the depth 7 neural network and MWPM decoders have a large difference in F1 for probability of measurement error of 5%.

4.2.3 Summary of Performance for Increasing Code Depths

The results in this section help to test the following hypotheses:

- A machine learning based decoder can predict corrective operations on surface codes of depth 3, 5, and 7 with better than chance accuracy at all noise levels.
- Increasing code depth does not change the performance of all decoders

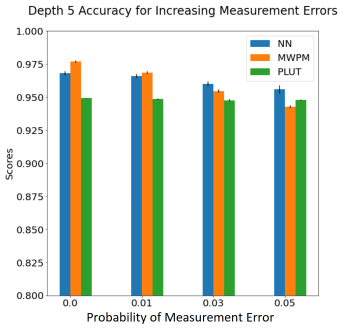
The neural network decoder performed poorly for depth 3, showing that the decoder cannot predict corrective operations with better than chance accuracy at any noise level. This is likely because there is not enough data for the model to learn how to predict corrections for unseen data. The total dataset for depth 3 code is only 28 observations, so the poor performance and high variance due to small training size can be explained by the small dataset. Depth 5 neural network decoding is much better than the depth 3 decoding. To understand if the models are predicting with better than chance accuracy, F1 score is used due to the imbalance of positive and negative class values. The mean F1 score for no measurement errors is 0.669, which is high enough to indicate that the model is predicting with better than chance accuracy. At 5% probability of measurement error, the F1 score is lowest among all the noise levels, and it is 0.5195. This is not a high F1 score, but it is better than chance since the F1 value for MWPM at this noise level is 0.5283. Depth 7 is the best performing decoder, and performs better than chance since the lowest F1 score is 0.6736, which is a fair F1 score.

There is mostly evidence against the second hypothesis. The only case in which there does not seem to be a change in F1 or accuracy in response to increasing code depth is that of the MWPM decoder. The accuracy does not change at 5% measurement error and the F1 score does not change at 1 and 3% probability of measurement error. In general, the neural network decoder increases in accuracy and F1 with increasing code depths, and in some cases the MWPM decoder and PLUT decoder do so as well. There is not enough evidence overall to support this hypothesis. There is more evidence in these results to contradict it.

4.3 Noise Results and Analysis

Figures 24 and 23, as well as Tables 12–15 and Tables 8–11 (in Appendix A) summarize the accuracy and F1 scores with varying levels of noise at surface code depths of 3, 5, and 7.

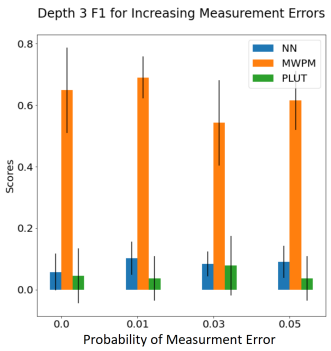
(a) Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 3 with various levels of noisy measurements



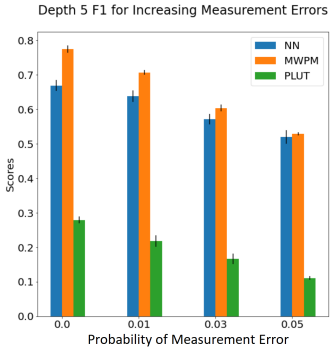
(b) Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 5 with various levels of noisy measurements

(c) Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 7 with various levels of noisy measurements

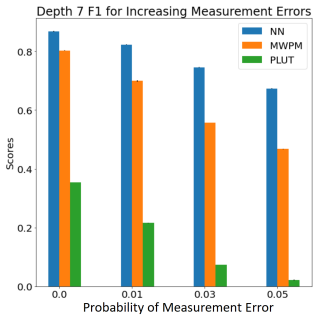
Figure 23: Accuracy for increasing probability of measurement error of depths 3, 5, and 7



(a) F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 3 with various levels of noisy measurements



(b) F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 5 with various levels of noisy measurements



(c) F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder at depth 7 with various levels of noisy measurements

Figure 24: F1 for increasing probability of measurement error of depths 3, 5, and 7

4.3.1 Increasing Measurement Error on Depth 3

Overall, the results presented in Figures 24a and 23a show that for depth 3 the neural network and PLUT decoders are very poor decoders. There is not a clear relation between F1 and accuracy score and code depth for the PLUT and neural network. The performance of the MWPM decoder is lower for error levels of 0.03 and 0.05 compared to levels of 0.00 and 0.01. However, it cannot be concluded that the performance metrics are increasing or decreasing in relation to measurement error for any decoder at depth 3.

4.3.2 Increasing Measurement Error on Depth 5

The F1 and accuracy scores that compare the decoders of depth 5 (Figures 24b and 23b) with increasing measurement error are more meaningful than the scores from the depth 3 code. Overall, the decoders decreased in F1 score with increasing measurement error. The difference between the F1 score of the neural network and MWPM decoder appears to decrease with increasing code depth as well. This can

be observed by looking at the difference between the neural network and MWPM decoder at noise level of 0%, and also observing that the mean of the MWPM F1 score at noise level of 5% is within one standard deviation of the mean of the neural network decoder.

The neural network and MWPM decoder decreased in accuracy for increasing measurement error, while the PLUT's accuracy stayed constant. This may be because many of the predicted observations from the PLUT do not include any positive class labels at all, and this results in a more constant accuracy performance for the PLUT. The accuracy of the neural network is better at noise level of 3% and 5% compared to both the MWPM decoder and PLUT. The MWPM decoder has lower accuracy than the PLUT at noise level of 5%, which indicates that while the MWPM decoder may have more true positives than the PLUT, it has more false negatives with increasing noise levels that result in a lower accuracy than the PLUT.

4.3.3 Increasing Measurement Error on Depth 7

The results of increasing measurement errors for depth 7 surface code, shown in Figure 23c and Figure 24c, are straightforward. All decoders decrease in F1 and accuracy for increasing measurement errors. The neural network has higher F1 and accuracy than the MWPM and PLUT decoders for all probabilities of measurement error. The MWPM algorithm decreases in accuracy and F1 at a faster rate than the neural network, which can be observed in the differences between the F1 and accuracy scores for each noise level. The PLUT has a decreasing F1 score with increased noise levels, but has a constant accuracy for increasing noise levels as well. The PLUT has a better accuracy than the MWPM algorithm for noise levels of 3% and 5% for the reason that is explained in Section 4.3.2 where there is a similar result.

4.3.4 Summary of Performance for Increasing Probability of Measurement Error

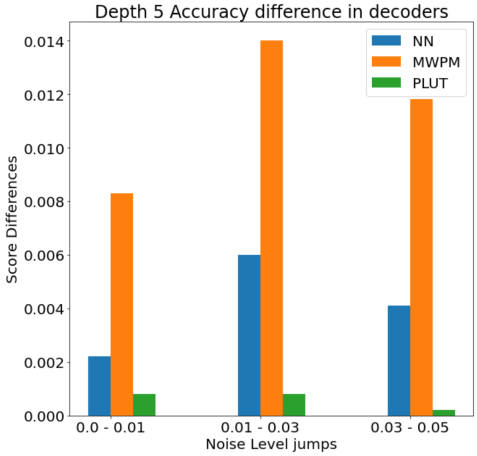
The hypotheses being tested that relate to the decoder performance with increasing measurement error are:

- Increasing noise levels results decreases performance metrics in all decoders
- Machine learning decoders have execution times that increase at a slower rate than algorithmic decoders with increasing code depths

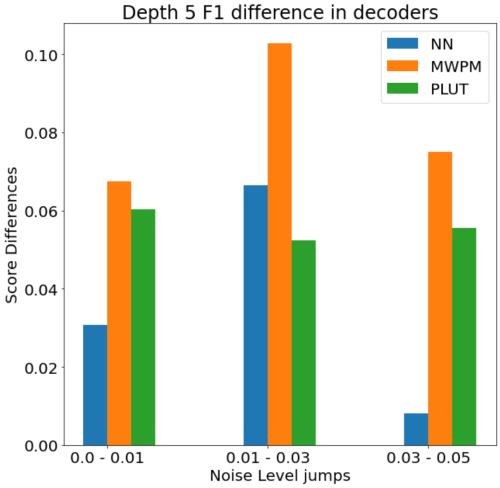
The results of depths 5 and 7 surface codes support the first hypothesis, except in the case where the PLUT's accuracy. Depth 3 code does not support this hypothesis since the neural network and PLUT decoders are poor and the sample sizes used in testing during k-fold cross validation are small. However, in the case of the MWPM and neural network decoder, the results show evidence that this hypothesis is true.

The second hypothesis is about the rate of performance decreases in response to increasing measurement error. PLUT decoding and neural network decoding for depth 3 are poor decoders, but depth 5 and depth 7 surface codes can support this hypothesis. Both the neural network and MWPM decoder decrease in response to increased noise levels on measurements, but as noise levels increase, the MWPM decoder decreases in F1 and accuracy at a faster rate. To demonstrate this, Figure 25, as well as Tables 16 and 17 (in Appendix A) summarize the differences in F1 and accuracy with increasing noise levels.

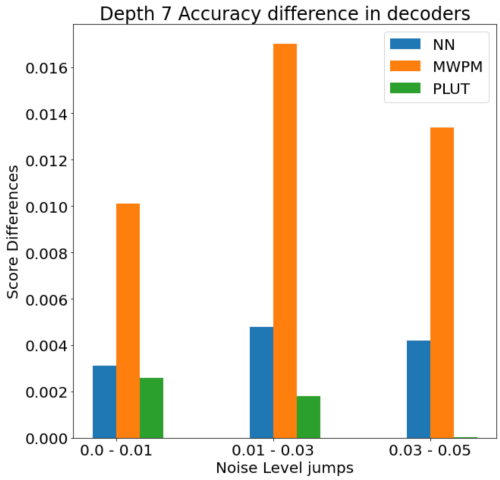
Figures 25b and 25a show the differences in decoders for depth 5 code, while Figures 25d and 25c show the differences in decoders for depth 7 code. In these figures, it can be seen that the MWPM algorithm has the highest difference in accuracy and F1 as the probability of measurement error increases. There is not a clear asymptotic or constant relationship with the increase of noise and decrease in the performance of



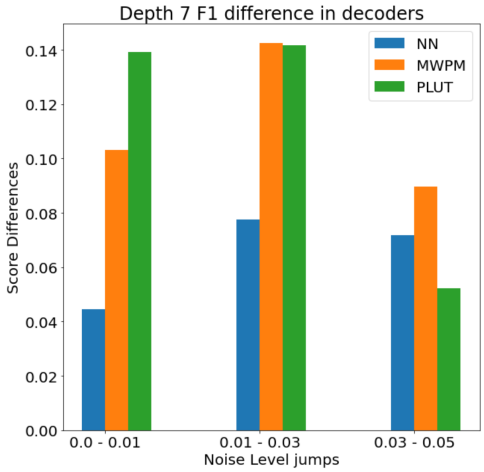
(a) Depth 5 accuracy difference



(b) Depth 5 F1 difference



(c) Depth 7 accuracy difference



(d) Depth 7 F1 difference

Figure 25: Summary of differences in F1 and accuracy with increasing probability of measurement error

MWPM, and likely needs more tests with noise to understand if a clear relationship exists.

The PLUT is a poor decoder overall, which is the reason why it has constant differences in F1 and accuracy with respect in increasing noise. Even though it has small differences in accuracy with respect to noise, this alone does not support the PLUT being a better decoder than the MWPM algorithm or the neural network decoder.

The neural network decoder has a significantly smaller difference in F1 and accuracy when compared to the MWPM algorithm for both depth 5 and depth 7. This supports the hypothesis that the effectiveness with predicting corrective operations on surface codes decreases at a higher rate for the MWPM algorithm with respect to increase measurement noise levels when compared to neural network decoding.

4.4 Execution Time Results and Analysis

The execution times reported here (Figure 26, as well as Table 18 in Appendix A) are the average times required to predict the corrective operations for one syndrome observation in the test set, across all depths and noise levels, since noisy measurement does not affect execution time. The time in seconds is recorded for each decoder at each noise level to predict corrections for the test set used for each fold in cross validation. The average is then computed for each fold by dividing by the number of observations. The mean of the times gathered from each fold is displayed. The PLUT and neural network decoders both have low execution times, with the MWPM algorithm clearly having the highest. While the execution times of the PLUT and neural network decoder do not seem to increase much as surface code depth increases, that of the MWPM algorithm is shown to increase more slowly with larger code depths. This is consistent with the time complexity of MWPM decoders in other

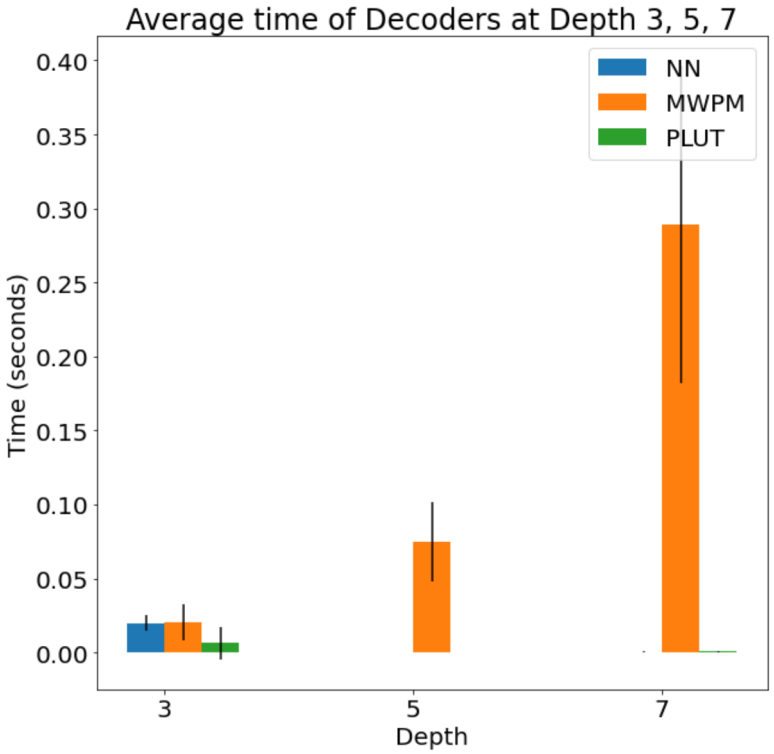


Figure 26: Execution Time of Partial Lookup Table, Minimum Weight Perfect Matching, and Neural Network decoder for Depths 3, 5, and 7

research, which is $O(n^2)$. It is clear that the MWPM algorithm as implemented for this research has a clear execution time disadvantage when compared to other decoders.

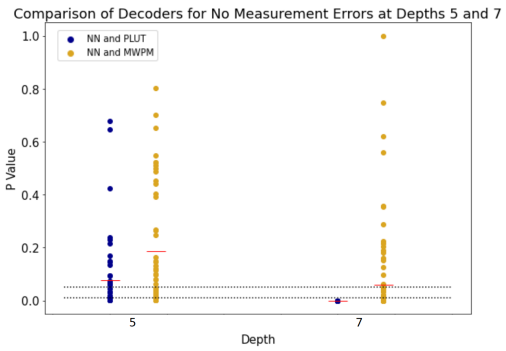
4.5 Analysis of P-Values

McNemar's test is performed for all experiments to compare the neural network decoders to the MWPM and PLUT decoders. This test is used to determine whether the decoders' predictions differ enough to be statistically significant. A p-value less than 0.05 is interpreted to indicate that the marginal probabilities of misclassification for a label are significantly different from each other, and validate that the classifiers are performing significantly differently. A p-value is made for each class label, and all of the p-values are plotted for the decoders being compared at various noise levels and depths. Due to the lack of data for the depth 3 decoder, McNemar's test is only performed for the depth 5 and depth 7 decoders. Since McNemar's test is only used to compare binary classifiers, the test is performed for each class label in the multilabel output vector of the classifiers.

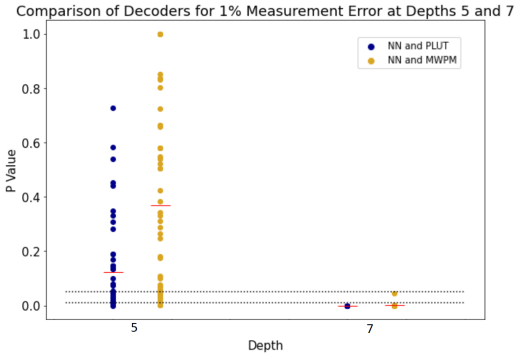
4.5.1 McNemar's Test with Depth 5 and 7

Figure 4.5.2 summarizes p-values of each class label for the test statistic at depths 5 and 7 with varying noise levels (see also Tables 19–22 in Appendix A). Each point on the scatterplot indicates a p-value for a class label. For a probability of measurement error of 0%, the neural network and MWPM algorithms do not have statistically significant p-values and do not support these two decoders having marginal probabilities of misclassification different from each other. The only instance where the p-value is below is with the McNemar's test at depth 7 between the neural network and the PLUT. There is sufficient evidence in this case to conclude that the two decoders

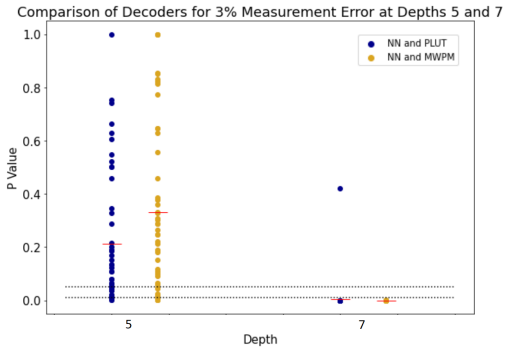
have different marginal probabilities of error.



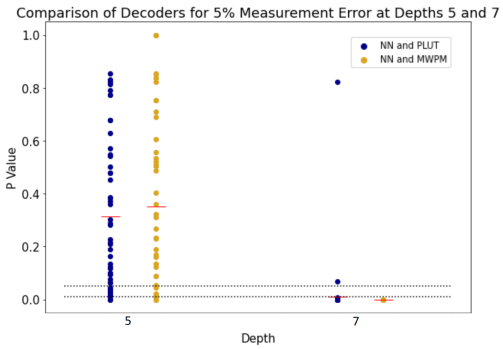
(a) 0% probability of measurement error



(b) 1% probability of measurement error



(c) 3% probability of measurement error



(d) 5% probability of measurement error

Figure 27: Results of McNemar's test for depth 5 and depth 7 with various probabilities of measurement error. The horizontal lines indicate p-values of 0.05 and 0.01. A p-value at or below 0.05 is considered statistically significant in this research. Each point represents a p-value for a class label in the multioutput vector

At a probability of measurement error of 1%, shown in Figure 27b, the depth 5 decoders have p-values above 0.05, it cannot be concluded that these decoders have marginal homogeneity. However, both the PLUT and MWPM decoder have p-values below 0.01 when compared to the outcome of the neural network, which provides strong evidence that the marginal probabilities for the outcomes of these decoders are different.

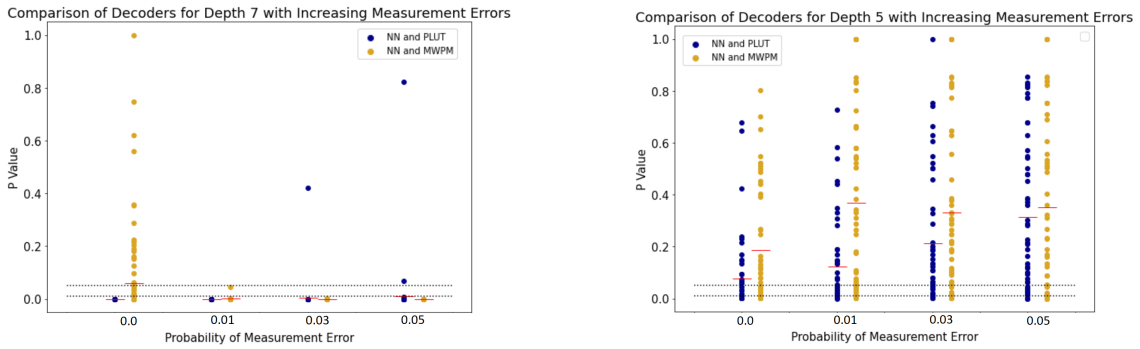
Figure 27c plots the p-values of each class label for the test statistic at depths 5 and 7 (see also Table 21 in Appendix A). These results are similar to those with probability

of measurement error of 1%, with the depth 7 decoder comparisons being statistically significant and the depth 5 decoder comparisons not statistically significant.

Figure 27d plots all of the p-values of each class label for the test statistic at depths 5 and 7 (see also Table 22 in Appendix A). These results are similar to those with probability of measurement error of 1% and 3%, with the depth 7 decoder comparisons being statistically significant and the depth 5 decoder comparisons not statistically significant.

4.5.2 McNemar's Test with Increasing Probability of Measurement Error

Figure 4.5.2 (and Tables 19–22 in Appendix A) summarize the results from McNemar's test in relation to increasing noise levels on measurements. Depth 5 p-values



(a) Depth 7 (b) Depth 5

Figure 28: McNemar's test results for decoders at depth 7 and depth 5 with increasing probability of measurement error

are not significantly affected by increase of noise, and all p-values are above the 0.05 threshold of significant difference between marginal probabilities. The depth 7 results show almost all of the p-values below 0.01, which indicates that the marginal probabilities of outcomes are different.

4.6 Neural Network Decoder Analysis

This section presents an overview of the neural network decoder, including its challenges in learning and training. An evaluation of AUROC is done as well at various depths and noise levels for the neural network decoders.

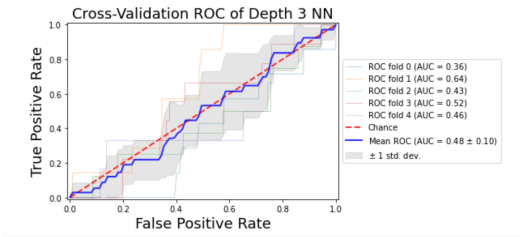
How well a neural network decoder can learn to perform corrective operations depends on the training and the depth of the code. One of the most glaring results is that the depth 3 surface code is not suitable for machine learning methods. This could be for a number of reasons, but the most likely reason is insufficient data. However, since this is only a concern for depth 3 surface code, alternative methods such as a complete lookup table can be used since there are significantly less syndrome possibilities and combinations of data qubit errors when compared with much larger codes.

4.6.1 AUROC

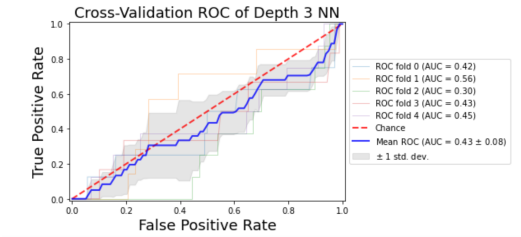
Another way to understand how each neural network model is performing is to look at the AUROC. For each model, there is an ROC for each label, which is micro-averaged for each fold. The mean of the mirco average of the ROCs at each depth is an indicator of the quality of the model during testing. The only mean mirco-average AUROC that is below 50%, indicating a poor model, is for depth 3. The other two have a AUROC of over 90%, which indicates good model performance.

Depth 3 ROCs all have an average area under the curve below 50%, which indicates poor performance. This occurs at all noise levels, and there is no evidence to show significantly better or worse performance for the various noise levels. The micro-average AUROC for varying noise levels is summarized for depth 3 in Figure 29.

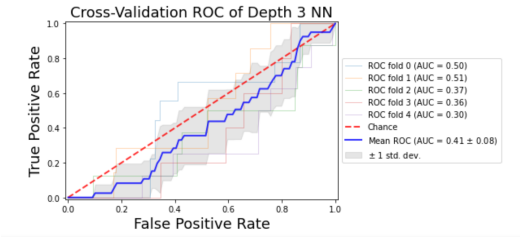
Depth 5 ROCs all have an average area under the curve above 90%, which indicates good performance. This occurs at all noise levels, and there is no evidence to show



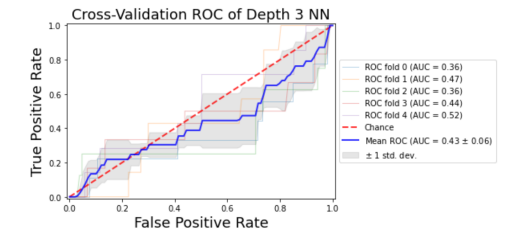
(a) 0% probability of measurement error



(b) 1% probability of measurement error



(c) 3% probability of measurement error



(d) 5% probability of measurement error

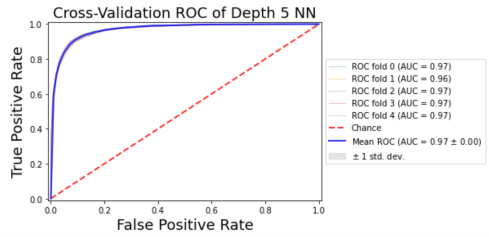
Figure 29: Micro-average ROC for depth 3

significantly better or worse performance for the various noise levels. The micro-average AUROC for varying noise levels is summarized for depth 5 in Figure 30.

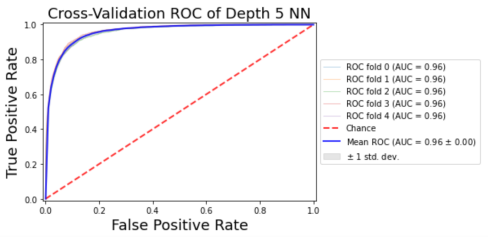
Depth 7 ROCs indicate good classification performance with a high micro-average ROC for all folds. Although this micro-average decreases with increasing probability of measurement error, it is still relatively high for all levels of noise, and indicates good learning from the neural network. The micro-average AUROC for varying noise levels is summarized for depth 5 in Figure 31.

4.6.2 Neural Network Decoding Cycles

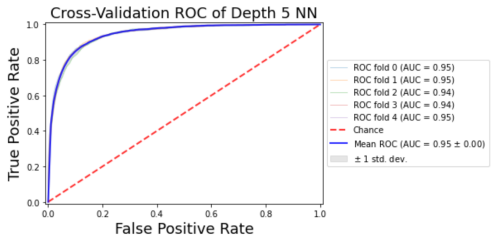
To see how neural network decoding does in the context of how long it can protect the logical qubit state in surface code, tests were run for surface code depths 3, 5, and 7. These tests loaded the trained neural networks and ran the predictions with the custom-made simulation used to generate the error syndrome and labels for the experiments. These tests counted the number of error correction cycles the code would run for with the corrective operations proposed by the trained model. For each code



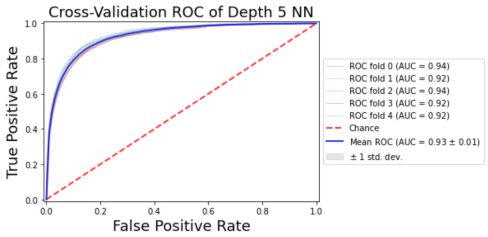
(a) 0% probability of measurement error



(b) 1% probability of measurement error

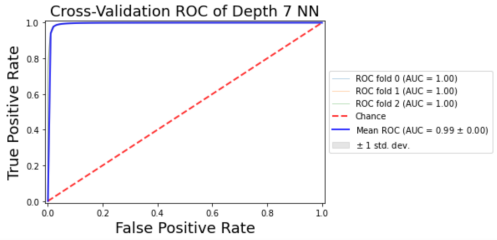


(c) 3% probability of measurement error

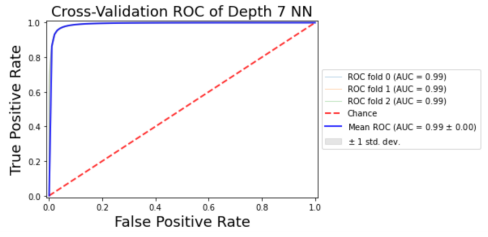


(d) 5% probability of measurement error

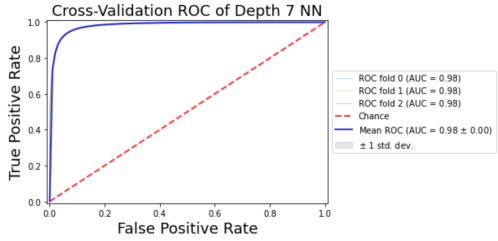
Figure 30: Micro-average ROC for depth 5



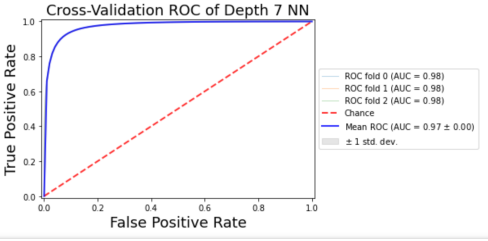
(a) 0% probability of measurement error



(b) 1% probability of measurement error



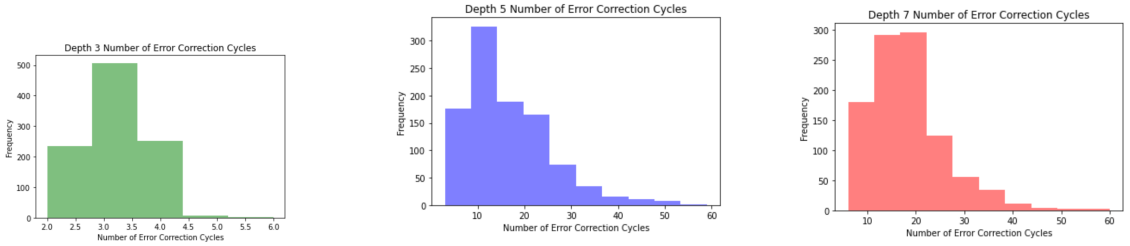
(c) 3% probability of measurement error



(d) 5% probability of measurement error

Figure 31: Micro-average ROC for depth 7

depth, the surface code was generated without any data qubit errors. Before each correction cycle, one data qubit error is randomly generated on the code. The model then performs corrective operations on the surface code. This process is finished when the surface code reaches the lowest number of data qubits that could produce a logical error. In the case of the surface code, the lower number of data qubits that could produce a logical error is equal to the code depth, since a logical error occurs with an error chain that touches either the top and the bottom or the right and the left side. The number of cycles run before the data qubit errors reached this threshold are recorded. For each depth of surface code, this test was run 1000 times to create a meaningful distribution, mean, and standard deviation for the error correction cycles the model predicted before the maximum number of tolerable data qubit errors was reached. These results are summarized in Table 24 in Appendix A and Figure 32. The separate distributions are presented in Figures 32a, 32b, and 32c.



(a) Depth 3 (b) Depth 5 (c) Depth 7

Figure 32: Distribution of number of error correction cycles for depths 3, 5, and 7

These tests also help to test the hypothesis that all machine learning based decoders can predict corrective operations with better than chance accuracy. The number of error correction cycles a decoder that can only predict corrective operations no better than chance would fail to correct the surface code for more cycles than the maximum allowable number of number of data qubit errors. For these experiments, this maximum number is equal to the code depth. Depth 3 has a mean of less than 3, which indicates that this type of decoder cannot correct data qubits with better than

chance accuracy. However, the neural network decoder for depth 5 has an average error correction cycle of 10 rounds, while the neural network decoder for depth 7 has an average error correction cycle of 21 rounds. Although the standard deviation is high for these two decoders, it is still evidence of the neural network decoder of depths 5 and 7 can predict corrective operations better than chance.

4.7 Summary

In this chapter, the effects of depth and increasing measurement error are shown for MWPM, PLUT, and NN decoders. These decoders have F1 and accuracy analyzed in Sections 4.2 and 4.3, and in Section 4.5 the significance of decoder performance through p-values generated from McNemar's test. In general, more conclusive results are generated from more data, such as the case of depth 7 surface code. Due to lack of data because of the constraints of the experiments, the depth 3 surface code's neural network decoder is poor. However, both depth 5 and depth 7 decoders are demonstrated to increase in F1 and accuracy with increasing probability of measurement error, although only depth 7 has results that are confirmed to be significant from McNemar's test. However, all of these results serve to answer the hypotheses posed in this research.

V. Conclusions

5.1 Overview

This chapter presents the conclusions and contributions of this research and some of the future of directions by which it could be extended. Section 5.2 summarizes the conclusions drawn from the experimental results presented in Chapter IV. Section 5.3 talks about the contributions that this work makes to the area of quantum error correction. Section 5.4 expands on the future work that follows from the research done in this thesis. Lastly, Section 5.5 finishes with some concluding remarks about low-level decoding and neural networks.

5.2 Conclusions

Chapter I introduced research questions regarding low-level decoding and testable hypotheses that would increase understanding with respect to those research questions. The experimental results supported some hypotheses and contradicted others. A summary of those hypotheses and the evidence to support or contradict them from the results follows:

1. **Hypothesis 1:** The effectiveness of predicting corrective operations of machine learning-based decoders decreases more slowly than that of algorithmic decoders with increasing noise levels on measurements.

    - This is supported by the results in Section 4.3, in which the differences F1 and accuracy scores for increasing probability of measurement error are shown to be higher for MWPM algorithm than the neural network decoder.

2. **Hypothesis 2:** Neural network-based decoders have execution times that increase more slowly by a constant factor than algorithmic decoders with increasing code depths.

    - This is supported by the results in Section 4.4, in which the MWPM algorithm is shown to have far greater average execution time than the neural network decoder. The average execution times of neither the neural network nor the partial lookup table decoders increase significantly with increasing code depths.

3. **Hypothesis 3:** A neural network-based decoder can predict corrective operations on surface codes of depth 3, 5, and 7 with better than chance accuracy for noise levels of 0%, 1%, 3%, and 5%.

    - This is supported by the results in Sections 4.2 and 4.6, but only for depths 5 and 7. Neural network decoding using depth 3 performed poorly overall, but experimental evidence indicates that for both depths 5 and 7, neural network decoding can predict corrective operations better than chance.

4. **Hypothesis 4:** For graph algorithm decoders and neural network-based decoders, increasing code depth does not change effectiveness of predicting corrective operations.

    - This hypothesis is not supported by the experimental results. Code depth does affect the change in performance among decoders, with most decoders improving in F1 and accuracy with increasing code depth. This observation is based on the results presented in Section 4.5.1, which show that at depth 5, the neural network decoder is similar in the marginal probabilities of its outcomes to those of the partial lookup table and the MWPM decoder. At depth 7, the marginal probabilities of the outcomes are statistically

significant, which indicates that code depth does change the performance of decoders when compared to one another.

5. **Hypothesis 5:** Increasing measurement noise levels decreases effectiveness of predicting corrective operations in all decoders.

    - This is supported by the results presented in Section 4.3, in which nearly all of the decoders decrease in both F1 and accuracy with increasing levels of noise. The only instance in which this is not the case is the partial lookup table decoder, which is likely due to it having predicted no corrective operations at all for many unseen observations.

The p-values discussed in Section 4.5 also provides evidence for statistically significant differences among the decoders in how effectively they predict corrective operations. This applies primarily to the depth 7 case, in which there are much more statistically significant differences in the marginal probabilities of the outcomes of MWPM and partial lookup table decoders when compared to the neural network decoder. Since the F1 and accuracy score is higher for the neural network decoder when compared to the MWPM and partial lookup table decoders, this difference in outcomes supports the conclusion that the neural network decoder has significantly better decoding than the other decoders at depth 7 for all noise levels.

5.3 Contributions

This research contributes to the body of knowledge of low-level decoders for surface codes of small depths. The results in this research point to advantages in neural network decoding for its fast execution times when compared to an unoptimized MWPM algorithm as well as greater effectiveness in increasing surface code depths in the presence of noise on the ancilla qubits. Neural network decoding is said to be

more flexible and faster than algorithmic decoders [48], and this research supports this claim with experimental results that show greater resistance to noise for a neural network decoder than for an MWPM decoder. Along with this, the execution time of the neural network decoder is lower than that of the MWPM decoder and the PLUT decoder. Together, these conclusions imply that neural network decoders are a viable alternative to algorithmic decoding for fast decoding of surface code syndromes.

These experiments are the first to look at the impact of increasing only the probability of error on the ancilla while the error rate of data qubits remains constant. An understanding of how well decoders can withstand measurement errors in ancilla qubits is intrinsic to how well they can perform under more realistic noise. Since the marginal probabilities of different outcomes are statistically significant for a depth 7 code with measurement errors, neural network decoding should be seriously considered as a decoder than can contribute to fault-tolerant decoding with large code depths.

The caveat regarding low neural network execution time and high effectiveness is that more care needs to be taken in training the network. Although execution times of neural network decoders are small, training time increases with the code depth, along with the time required to simulate sufficient data for training. However, a similar process to the one used in this thesis can be used with more accurate (and complex) simulations of errors due to noise on quantum computers. This option is comparable to changing algorithmic decoders, such as MWPM, by making them more accurate (as well as more complex and difficult to understand) while also increasing their execution times. Despite difficulties in training neural network decoders, enough high quality data can be used to train these decoders to predict corrective operations for unseen error syndromes.

5.4 Future Work

Several areas of future work in the research in low-level decoding of the surface code are suggested. One important step would be to optimize the efficiency of data simulation so that data with greater code depths can be generated and used in similar experiments. This would expand on the evidence of decoder performance relative to code depth if patterns in evaluation metrics in larger code depths are consistent with those in smaller code depths. Another way to improve the simulation tool that would be useful in future work would be to implement more accurate noise models that can better represent real-life quantum computers. This would include not only increasing the noise data qubits and the measurement qubits, but also introducing noise in the corrective operations and noise probability distributions more complex than the one used in this research (an X, Y, or Z error happening with equal probability on a arbitrary qubit). With the extension of low-level decoding in these areas, a further understanding of decoders and the challenges of surface code decoding can be achieved.

Another area to be considered in the future is the use of error correction cycles consisting of more than one measurement of the ancilla qubits before decoding and corrective operations are applied to the code. Error correction cycles are commonly used to make fault tolerant decoders [43]. Cycles can detect measurement errors and better account for those errors in the decoding process. However, they also make decoding itself substantially more complicated, since the two spatial dimensions of the grid of measurements are extended by a third temporal dimension. Since decoding cycles are helpful in determining where measurement errors occur, error correction cycles are used in simulations, and decoders are made to account for more than one measurement. However, since making those additional measurements requires more time, and the decoding process is harder for algorithmic decoders, which leads to

larger execution times, the threshold for error for the code is lowered. Future work that extends this research could use neural networks, which are more resistant to the effects of measurement error on performance, and compare them to the MWPM algorithm that uses error correction cycles. An analysis could be done on the tradeoffs of using error correction cycles with MWPM and a neural network decoder without error correction cycles to understand if they are needed in for error correction with a decoder that can withstand the affects of measurement error.

Dimensionality reduction of the input is not a significant consideration for the code depths used in this research. However, the effectiveness of neural network decoding for larger code depths would potentially benefit from the use of Convolutional Neural Networks (CNNs). . This would contribute to the body of knowledge regarding low-level decoders and the neural network architecture and training that would be most suited for large code depths. In the future, this research could even be integrated with Qiskit Ignis, which has a small topological code module. The decoder implemented in this module is the MWPM algorithm, and there is a desire to implement a machine learning decoder in the future, which this research and further research in this area supports.

5.5 Concluding Remarks

Quantum computation has untapped potential that can only be realized through improvements in hardware and systems that can mitigate the error present on quantum devices. Quantum error correction is one such way to address that challenge, but much work remains in this field. Finding an effective and efficient error correction architecture, such as the surface code and other stabilizer codes, is one key component. Another is the decoder to be used alongside this architecture. Depending on which of the various metrics are considered, some decoders are better than others, and the full

advantages and disadvantages of decoders are still being explored. This research has produced evidence to support the conclusions that neural network decoding has advantages over other decoders and that it is a viable decoding mechanism to integrate into quantum simulations such as those performed by Qiskit. Given enough evidence and development, neural network decoders could even be integrated with quantum hardware with future advances in quantum computing. This would help to realize the potential of practical quantum computation through quantum error correction.

Appendix A. Tables of Experimental Results

	NN	MWPM	PLUT
Depth 3 Mean	0.864	0.950	0.912
Depth 3 StDev	0.115	0.017	0.010
Depth 5 Mean	0.968	0.977	0.949
Depth 5 StDev	0.002	0.001	0.000
Depth 7 Mean	0.990	0.984	0.965
Depth 7 StDev	8.023×10^{-5}	0.000	2.911×10^{-5}

Table 8: Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with perfect measurements

	NN	MWPM	PLUT
Depth 3 Mean	0.896	0.955	0.911
Depth 3 StDev	0.072	0.016	0.010
Depth 5 Mean	0.966	0.969	0.948
Depth 5 StDev	0.001	0.001	0.001
Depth 7 Mean	0.987	0.974	0.962
Depth 7 StDev	5.801×10^{-5}	0.000	8.169×10^{-6}

Table 9: Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 1% probability of measurement error

	NN	MWPM	PLUT
Depth 3 Mean	0.090	0.615	0.036
Depth 3 StDev	0.052	0.096	0.073
Depth 5 Mean	0.520	0.528	0.111
Depth 5 StDev	0.020	0.005	0.005
Depth 7 Mean	0.674	0.467	0.022
Depth 7 StDev	0.001	0.001	0.001

Table 15: F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 5% probability of measurement error

	NN	MWPM	PLUT
Depth 3 Mean	0.884	0.923	0.914
Depth 3 StDev	0.100		0.009
Depth 5 Mean	0.960	0.955	0.948
Depth 5 StDev	0.002	0.001	0.001
Depth 7 Mean	0.982	0.957	0.960
Depth 7 StDev	5.560×10^{-6}	5.129×10^{-5}	1.190×10^{-5}

Table 10: Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 3% probability of measurement error

	NN	MWPM	PLUT
Depth 3 Mean	0.895	0.944	0.905
Depth 3 StDev	0.069	0.012	0.015
Depth 5 Mean	0.956	0.943	0.948
Depth 5 StDev	0.003	0.001	0.001
Depth 7 Mean	0.978	0.944	0.960
Depth 7 StDev	7.827×10^{-5}	0.000	3.445×10^{-5}

Table 11: Accuracy of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 5% probability of measurement error

	NN	MWPM	PLUT
Depth 3 Mean	0.057	0.648	0.044
Depth 3 StDev	0.059	0.139	0.089
Depth 5 Mean	0.669	0.774	0.279
Depth 5 StDev	0.016	0.011	0.010
Depth 7 Mean	0.868	0.803	0.355
Depth 7 StDev	0.002	0.001	0.001

Table 12: F1 of Partial Lookup Table, Minimum Weight Perfect Matching, and Neural Network decoder with perfect measurements

	NN	MWPM	PLUT
Depth 3 Mean	0.103	0.690	0.036
Depth 3 StDev	0.054	0.069	0.073
Depth 5 Mean	0.638	0.706	0.219
Depth 5 StDev	0.016	0.007	0.017
Depth 7 Mean	0.823	0.700	0.215
Depth 7 StDev	0.001	0.002	0.001

Table 13: F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 1% probability of measurement error

	NN	MWPM	PLUT
Depth 3 Mean	0.084	0.542	0.078
Depth 3 StDev	0.040	0.138	0.097
Depth 5 Mean	0.571	0.603	0.166
Depth 5 StDev	0.015	0.010	0.015
Depth 7 Mean	0.745	0.557	0.074
Depth 7 StDev	0.001	0.001	0.000

Table 14: F1 of partial lookup table, Minimum Weight Perfect Matching, and neural network decoder with 3% probability of measurement error

	0.0-0.01	0.01-0.03	0.03-0.05
F1 Difference MWPM	0.068	0.103	0.075
Accuracy Difference MWPM	0.008	0.014	0.012
F1 Difference NN	0.031	0.067	0.008
Accuracy Difference NN	0.002	0.006	0.004
F1 Difference PLUT	0.060	0.052	0.056
Accuracy Difference PLUT	0.001	0.001	0.000

Table 16: F1 and accuracy differences with measurement errors on depth 5 code

	0.0-0.01	0.01-0.03	0.03-0.05
F1 Difference MWPM	0.103	0.143	0.090
Accuracy Difference MWPM	0.010	0.017	0.013
F1 Difference NN	0.045	0.078	0.072
Accuracy Difference NN	0.003	0.005	0.004
F1 Difference PLUT	0.139	0.142	0.052
Accuracy Difference PLUT	0.003	0.002	0.000

Table 17: F1 and accuracy differences with measurement errors on depth 7 code

	Depth 3	Depth 5	Depth 7
NN Mean	0.020	0.000	0.000
NN StDev	0.005	9.182×10^{-5}	0.000
MWPM Mean	0.020	0.075	0.289
MWPM StDev	0.012	0.027	0.107
PLUT Mean	0.006	3.393×10^{-5}	0.001
PLUT StDev	0.011	1.329×10^{-5}	0.000

Table 18: Execution Time of Partial Lookup Table, Minimum Weight Perfect Matching, and Neural Network decoders for Depths 3, 5, and 7

	Depth 5	Depth 7
NN/MWPM Mean	0.187	0.060
NN/MWPM StDev	0.223	0.161
NN/PLUT Mean	0.075	2.724×10^{-13}
NN/PLUT StDev	0.146	2.683×10^{-12}

Table 19: Table of results of McNemar's test for depth 5 and depth 7 with 0% probability of measurement error

	Depth 5	Depth 7
NN/MWPM Mean	0.368	0.000
NN/MWPM StDev	0.323	0.004
NN/PLUT Mean	0.123	2.146
NN/PLUT StDev	0.172	2.112

Table 20: Table of results of McNemar's test for depth 5 and depth 7 with 1% probability of measurement error

	Depth 5	Depth 7
NN/MWPM Mean	0.330	9.217
NN/MWPM StDev	0.311	7.875
NN/PLUT Mean	0.211	2.146
NN/PLUT StDev	0.247	2.112

Table 21: Table of results of McNemar's test for depth 5 and depth 7 with 3% probability of measurement error

	Depth 5	**Depth 7**
NN/MWPM Mean	0.352	3.654
NN/MWPM StDev	0.333	3.570
NN/PLUT Mean	0.314	0.009
NN/PLUT StDev	0.280	0.083

Table 22

Table 23: Results of McNemar's test for depth 5 and depth 7 with 5% probability of measurement error

	Mean	**Standard Deviation**
Depth 3	2.534	1.069
Depth 5	9.518	4.963
Depth 7	21.381	10.082

Table 24: Summary of mean and standard deviation of error correction cycles for neural network decoding

Bibliography

1. "Classification: Roc curve and auc nbsp;—nbsp; machine learning crash course." [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

2. "The no-cloning theorem," 2015. [Online]. Available: https://www.quantiki.org/wiki/no-cloning-theorem

3. S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," *Physical Review A*, vol. 70, 11 2004.

4. H. Abraham, AduOffei, R. Agarwal, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, E. Arbel, Arijit02, A. Asfaw, A. Avkhadiev, C. Azaustre, AzizNgoueya, A. Banerjee, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, A. Bhobe, L. S. Bishop, C. Blank, S. Bolos, S. Bosch, Brandon, S. Bravyi, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, J. M. Chow, S. Churchill, C. Claus, C. Clauss, R. Cocking, F. Correa, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Daniels, M. Dartiailh, DavideFrr, A. R. Davila, A. Dekusar, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, R. Fouilland, FranckChevallier, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, A. Gilliam, A. Giridharan, J. Gomez-Mosquera, Gonzalo, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, M. Haglund, I. Haide, I. Hamamura,

O. C. Hamido, F. Harkins, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, JamesSeaward, A. Javadi, A. Javadi-Abhari, W. Javed, Jessica, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, V. K, T. Kachmann, A. Kale, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, J. Kelso, S. King, Knabberjoe, Y. Kobayashi, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, D. Liu, P. Liu, Y. Maeng, K. Majmudar, A. Malyshev, J. Manela, J. Marecek, M. Marques, D. Maslov, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, J. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, M. Motta, MrF, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, J. Nicander, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, V. R. Pascuzzi, S. Perriello, A. Phan, F. Piro, M. Pistoia, C. Piveteau, P. Pocreau, A. Pozas-iKerstjens, M. Prokop, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. I. Rahman, A. Raja, N. Ramagiri, A. Rao, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, M. L. Rocca, D. M. Rodríguez, RohithKarur, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, H. Sandesara, R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, SooluThomas, Starfish, D. Steenken, M. Stypulkoski,

S. Sun, K. J. Sung, H. Takahashi, T. Takawale, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, M. Tomasik, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, V. Villar, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, S. Wood, J. Wootton, D. Yeralin, D. Yonge-Mallo, R. Young, J. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, Zoufalc, a kapila, a matsuo, bcamorrison, brandhsn, nick bronn, brosand, chlorophyll zz, csseifms, dekel.meirom, dekelmeirom, dekool, dime10, drholmie, dtrenev, ehchen, elfrocampeador, faisaldebouni, fanizzamarco, gabrieleagl, gadial, galeinston, georgios ts, gruu, hhorii, hykavitha, jagunther, jliu45, jscott2, kanejess, klinvill, krutik2966, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, sagar pahwa, rmoyard, saswati qiskit, scottkelso, sethmerkel, shaashwat, sternparky, strickroman, sumitpuri, tigerjack, toural, tsura crisaldo, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and M. Čepulkovskis, "Qiskit: An open-source framework for quantum computing," 2019.

5. P. Baireuther, T. E. O'brien, B. Tarasinski, and C. W. J. Beenakker, "Machine-learning-assisted correction of correlated qubit errors in a topological code," 2017.

6. H. Bombín, "Structure of 2d topological stabilizer codes," *Communications in Mathematical Physics*, vol. 327, no. 2, p. 387–432, Mar 2014. [Online]. Available: http://dx.doi.org/10.1007/s00220-014-1893-4

7. S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," 1998.

8. A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Physical Review A*, vol. 54, no. 2, p. 1098–1105, Aug 1996. [Online]. Available:

http://dx.doi.org/10.1103/PhysRevA.54.1098

9. D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, Jan. 2020. [Online]. Available: https://doi.org/10.1186/s12864-019-6413-7

10. F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

11. B. Criger and I. Ashraf, "Multi-path Summation for Decoding 2D Topological Codes," *Quantum*, vol. 2, p. 102, Oct. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-10-19-102

12. J. Dacombe, "An introduction to artificial neural networks (with example)," Oct 2017. [Online]. Available: https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b

13. A. S. Darmawan and D. Poulin, "Linear-time general decoding algorithm for the surface code," *Physical Review E*, vol. 97, 5 2018.

14. E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, p. 4452–4505, Sep 2002. [Online]. Available: http://dx.doi.org/10.1063/1.1499754

15. T. G. Dietterich, "Approximate statistical test for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, no. 7, pp. 1895–1923, 1998. [Online]. Available: http://neco.mitpress.org/cgi/content/abstract/10/7/1895

16. A. Ding, S. Jha, H. Liu, S. Singh, and W. Sun, "Qiskit surface code encoder/decoder," https://github.com/Phionx/qiskit_surface_codes, 2020.

17. A. G. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time," 2014.

18. A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, Sep 2012. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.86.032324

19. A. G. Fowler, D. S. Wang, and L. C. L. Hollenberg, "Surface code quantum error correction incorporating accurate error propagation," 2010.

20. S. Gamble, "Quantum computing: What it is, why we want it, and how we're trying to get it," Jan 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK538701/

21. I. Georgescu, S. Ashhab, and F. Nori, "Quantum simulation," *Reviews of Modern Physics*, vol. 86, no. 1, p. 153–185, Mar 2014. [Online]. Available: http://dx.doi.org/10.1103/RevModPhys.86.153

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

23. D. Gottesman, "Stabilizer codes and quantum error correction," 1997.

24. ——, "An introduction to quantum error correction and fault-tolerant quantum computation," *Quantum Information Science and Its Contributions to Mathematics*, p. 13–58, 2010. [Online]. Available: http://dx.doi.org/10.1090/psapm/068/2762145

25. L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical Review Letters*, vol. 79, no. 2, p. 325–328, Jul 1997. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.79.325

26. B. Gu and I. Franco, "When can quantum decoherence be mimicked by classical noise?" *The Journal of Chemical Physics*, vol. 151, no. 1, p. 014109, Jul 2019. [Online]. Available: http://dx.doi.org/10.1063/1.5099499

27. C. Horsman, A. G. Fowler, S. Devitt, and R. V. Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, Dec. 2012. [Online]. Available: https://doi.org/10.1088/1367-2630/14/12/123011

28. M. H. Hsieh and F. L. Gall, "Np-hardness of decoding quantum error-correction codes," *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 83, p. 052331, 5 2011. [Online]. Available: https://journals.aps.org/pra/abstract/10.1103/PhysRevA.83.052331

29. E. Huang, A. C. Doherty, and S. Flammia, "Performance of quantum error correction with coherent errors," *Physical Review A*, vol. 99, no. 2, Feb 2019. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.99.022313

30. V. Jagannath, "Random forest template," Aug 2020. [Online]. Available: https://community.tibco.com/wiki/random-forest-template-tibco-spotfire

31. G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/

32. A. Y. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, pp. 2–30, 7 2003. [Online]. Available: https://www.arxiv-vanity.com/papers/quant-ph/9707021/

33. D. Kobran and D. Banys, "Auc (area under the roc curve)." [Online]. Available: https://docs.paperspace.com/machine-learning/wiki/auc-area-under-the-roc-curve

34. S. Krastanov and L. Jiang, "Deep neural network probabilistic decoder for stabilizer codes," *Scientific Reports*, vol. 7, 12 2017.

35. N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

36. R. Mahmoudvand, H. Hassani, and R. Wilson, "Is the sample coefficient of variation a good estimator for the population coefficient of variation?" *University Library of Munich, Germany, MPRA Paper*, vol. 2, 01 2007.

37. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

38. N. Maskara, A. Kubica, and T. Jochym-O'Connor, "Advantages of versatile neural-network decoding for topological codes," *Physical Review A*, vol. 99, 2019.

39. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

40. N. A. of Engineering, *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2018 Symposium*. Washington, DC: The National Academies Press, 2019. [Online]. Available: https://www.nap.edu/catalog/25333/frontiers-of-engineering-reports-on-leading-edge-engineering-from-the

41. G. Ogundur, "Model k fold cross validation," Jan 2020. [Online]. Available: https://medium.com/@gulcanogundur/model-seC3A7imi-k-fold-cross-validation-4635b61f143c

42. J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, Aug 2018. [Online]. Available: http://dx.doi.org/10.22331/q-2018-08-06-79

43. J. Roffe, "Quantum error correction: an introductory guide," *Contemporary Physics*, vol. 60, no. 3, p. 226–245, Jul 2019. [Online]. Available: http://dx.doi.org/10.1080/00107514.2019.1667078

44. M. Schlosshauer, "Quantum decoherence," *Physics Reports*, vol. 831, p. 1–57, Oct 2019. [Online]. Available: http://dx.doi.org/10.1016/j.physrep.2019.10.001

45. A. M. Stephens, "Fault-tolerant thresholds for quantum error correction with the surface code," *Physical Review A*, vol. 89, no. 2, Feb 2014. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.89.022321

46. S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 987–999. [Online]. Available: https://doi.org/10.1145/3297858.3304007

47. G. Torlai and R. G. Melko, "Neural decoder for topological codes," *Physical Review Letters*, vol. 119, 7 2017.

48. S. Varsamopoulos, K. Bertels, and C. G. Almudever, "Decoding surface code with a distributed neural network–based decoder," *Quantum Machine Intelligence*, vol. 2, pp. 1–12, 6 2020. [Online]. Available: https://doi.org/10.1007/s42484-020-00015-9

49. ——, "Comparing neural network based decoders for the surface code," *IEEE Transactions on Computers*, vol. 69, no. 2, p. 300–311, Feb 2020. [Online]. Available: http://dx.doi.org/10.1109/TC.2019.2948612

50. S. Varsamopoulos, B. Criger, and K. Bertels, "Decoding small surface codes with feedforward neural networks," *Quantum Science and Technology*, vol. 3, 2018.

51. S. Verma, "Multi-label image classification with neural network: Keras," May 2020. [Online]. Available: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede

52. D. Voorhoede, "Bloch sphere." [Online]. Available: https://www.quantum-inspire.com/kbase/bloch-sphere/

53. D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. L. Hollenberg, "Threshold error rates for the toric and surface codes," 2009.

54. J. R. Wootton and D. Loss, "High threshold error correction for the surface code," *Physical Review Letters*, vol. 109, 10 2012.

55. N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*, 1st ed. USA: Cambridge University Press, 2008.

56. M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

57. W. Zhang, "Machine learning approaches to predicting company bankruptcy," *Journal of Financial Risk Management*, vol. 06, pp. 364–374, 01 2017.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
05–03–2020	Master's Thesis	Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

Performance of Various Low-Level Decoders for Surface Codes in the Presence of Measurement Error

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Claire E. Badger

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-008

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL Quantum Science and Technology
525 Brooks Rd.
Building 3, Suite H6-2
Rome NY 13441
DSN: 587-2504, COMM 315-330-2937
Email: afrl.ritc@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RITQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Quantum error correction is a research speciality within the area of quantum computing that constructs quantum circuits that correct for errors. Decoding is the process of using measurements from an error correcting code, known as error syndrome, to decide corrective operations to perform on the circuit. High-level decoding is the process of using the error syndrome to perform corrective logical operations, while low-level decoding uses the error syndrome to correct individual data qubits. Research on machine learning-based decoders is increasingly popular, but has not been thoroughly researched for low-level decoders. The type of error correcting code used is called surface code. A neural network-based decoder is developed and compared to a partial lookup table decoder and a graph algorithm-based decoder. The effects of increasing error correcting code size and increasing measurement errors on the error syndrome are analyzed for the decoders. The results demonstrate that there are advantages in terms of average execution time and resistance to increasing measurement error with the neural network-based decoder when compared to the two other decoders.

15. SUBJECT TERMS

quantum computing, quantum error correction, surface code, machine learning, neural network

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lieutenant Claire E. Badger, AFIT/ENG
U	U	U	UU	130	19b. TELEPHONE NUMBER *(include area code)* (312) 785-3636 x4526; laurence.merkle@afit.edu