Decoding the Surface Code with Attention-based Neural Networks

by

Kevin Y. Wu

Submitted to the Department of Computer Science
on April 19, 2024 in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Thesis supervisor: Qun Li
Title: Professor of Computer Science

Acknowledgments

Write your acknowledgments here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Quantum computing is an alternative computing paradigm that leverages quantum superposition and entanglement to accelerate computation. Quantum computers have the potential solve problems which are intractable for classical computers, providing quadratic to exponential speedups over classical computers for certain problems. Thus, quantum computers have the potential to revolutionize applications of computing in various fields, including cryptography, optimization, financial modeling, and machine learning. However, current quantum hardware is extremely limited, both in terms of noise and scalability. Quantum computers experience errors at a rate of about 10^{-3} per operation. Since many of these quantum algorithms include millions of operations, the error rates in these quantum machines are far too high to be useful in many of these applications. These high error rates remain a prominent reason why quantum systems are not widely used to accelerate computation.

To tackle this problem, we may apply *quantum error correction* techniques. Under the framework of quantum error correction, measurements of a *quantum error correction code* can used to deduce if errors occured in the system. This process of deduction is called *decoding*. In a physical quantum computer, a *decoder* usually takes the form of an algorithm running on a classical coprocessor, which reads in measurement results and outputs an error diagnosis. Afterwards, corrections may be applied to the quantum system or kept track of in classical control software. Such a decoder may take many different forms, including combinatorial algorithm or neural network, and exist in various computational mediums. Most importantly, the quality of a quantum computer's decoder will directly determine the system's capacity for fault tolerance.

Notably, an ideal decoder must have several properties:

1. **High Accuracy**: As the fault-tolerance of the quantum computer hinges on the accuracy of the decoder, an ideal decoder must be highly accurate.

2. **Low Latency**: Decoders operate in the liminal time between quantum operations. In current superconducting quantum architectures, this timeframe can be as short as 1μs. If the decoder does not finish decoding before the next round of quantum operations, errors will accumulate and the computation will be ruined. Reducing the decoder's latency will allow it to keep pace with quantum operations.

3. **Multimodality**: Current quantum computers provide many readouts, all of which can be leveraged to perform error correction. An ideal decoder should be able to utilize all of this information to provide better decoding.

4. **Scalability**: An ideal decoder's accuracy should not suffer as the size of the quantum system and the density of physical errors increase.

Creating a decoder that meets these criteria remains an active research area. This work will utilize the product of recent advancements in machine learning to construct a robust decoder for the surface code that satisfies many of these requirements.

Chapter 2

Quantum Mechanics and Quantum Computing

This chapter will provide a brief overview of the fundamentals of quantum information theory useful for quantum error correction, and how they correlate to quantum mechanics.

2.1 Qubits

In comparison to bits in classical computing, quantum computers use *qubits*, or quantum bits. Where the state of a bit represents high or low voltages in classical circuits, the state of qubits represent high and low energy states in systems with quantum properties. As we will discuss later, qubits have special properties, superposition and entanglement, that classical qubits do not possess. We may mathematically represent the state of a qubit as a complex unit vector $|\psi\rangle \in \mathbb{C}^n$, where $\mathbb{C}^n$ is a linear space of complex unit vectors, and $\mathbb{C}$ the complex numbers. Here, I use Dirac bracket notation to represent qubit states. I refer to $|\psi\rangle$ as a *ket vector*, and $\langle\psi|$ as a *bra vector*. They represent the following:

$$|\psi\rangle = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_n \end{pmatrix}, \langle\psi| = \begin{pmatrix} \psi_1 & \cdots & \psi_n \end{pmatrix}.$$

This ket vector representation corresponds to *pure states* in quantum mechanics.

2.2 Superposition

In 1804, Thomas Young conducted the double-slit experiment, where he showed that photons exhibit both wave-like and particle-like properties [1]. Through later experimentation, this phenomenon was also shown to exist for atoms and electrons as well. Returning to quantum mechanics, these experiments highlight an important concept — superposition. This quality of quantum particles fundamentally distinguishes quantum physics from classical physics. In fact, it gives rise to the property of quantum particles being able exist in multiple states simultaneously. Here, in our mathematical framework, the state of a particle in superposition

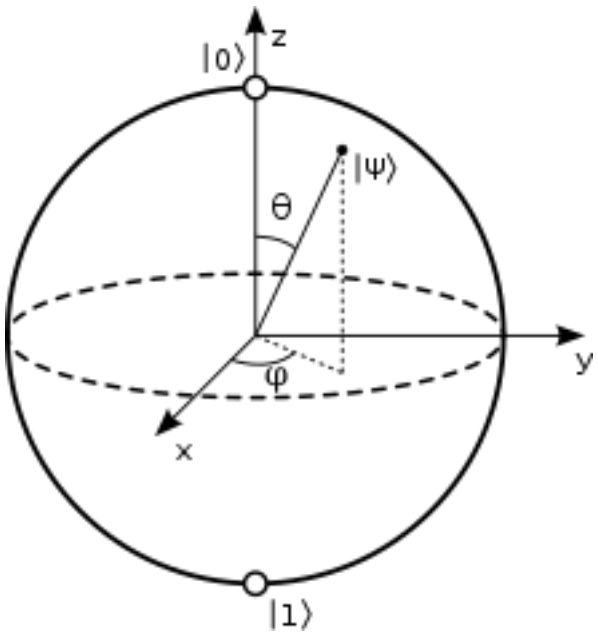


Figure 2.1: The Bloch Sphere

can be expressed as a convex combination of basis states, or vectors. We may define two basis states as follows:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Then, the state of a qubit in superposition, $|\psi\rangle$, can be expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, for $\alpha, \beta \in \mathbb{C}$, with $\alpha^2 + \beta^2 = 1$. These complex α and β values are called *probability amplitudes*. Such a state $|\psi\rangle$ has a α^2 chance of being in state $|0\rangle$, and a β^2 chance of being in state $|1\rangle$.

Thenm the state of a qubit can be expressed as a point on the *Bloch Sphere*, displayed in Figure 2.1. The Bloch Sphere is a unit sphere, with each point on its surface representing the possible state of a qubit. The north and south poles of the sphere correspond to the basis states $|0\rangle$ and $|1\rangle$, respectively. Any other point on the sphere represents a superposition of these basis states. The position of a point on the sphere is defined by two angles: θ and ϕ. These angles give us the state of the qubit on the sphere as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle,$$

where $\theta \in [0, \pi]$ is the polar angle measured from the z-axis, and $\phi \in [0, 2\pi]$ is the azimuthal angle in the x-y plane from the x-axis. As quantum states are indistiguishable under rotations corresponding to the roots of unity, we may assume that the probabililty amplitude for $|0\rangle$, α, is real and nonnegative. Afterwards, there remain three degrees of freedom, which correspond to the three dimensions in the Bloch sphere. A simple parameterization yields the two angles.

The Bloch Sphere representation is useful because it provides an intuitive way to visualize the state of a qubit, and as we will see later, the operations (quantum gates) that change these states.

2.3 Entanglement and Multi-Qubit Systems

One of the main draws of quantum computing is the quantum parallelism that results from way quantum information is stored and processed across multiple qubits. The *tensor product*, denoted $\otimes$, is central to mathematically representing such quantum states. For two qubits, each represented by a Hilbert space $\mathcal{H}$, the combined system is described by the tensor product space $\mathcal{H} \otimes \mathcal{H}$. If a qubit is in state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and a second qubit in state $|\phi\rangle = \gamma|0\rangle + \delta|1\rangle$, the joint system state $|\psi\rangle \otimes |\phi\rangle$ is given by:

$$|\psi\rangle \otimes |\phi\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle.$$

Such a composite system is *separable*, since it may be expressed as the tensor product of two single-qubit states. However, not all quantum states on multiple qubits have this property. Consider the Bell state, given below:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

From the perspective of quantum information science, we may describe this state as *entangled*. Multiple qubits may be entangled together, producing a state which cannot be expressed as separate states. Instead, such a state must be described for the system as a whole. Entanglement is often produced by quantum operations that act on multiple qubits at once. Such a quantum operation effectively acts on a vector space of exponential size, $\mathbb{C}^{2^n}$. This greatly increases the computational power of a quantum computer when compared to a classical computer.

2.4 Quantum Operators

Quantum states may be manipulated and controlled via various quantum operations. Mathematically, these quantum operators may be expressed as unitary operators on a Hilbert space. Such a unitary operator, or matrix U preserves the inner product, i.e. $U^\dagger U = UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of U, and I is the identity matrix. An application of a quantum operator to a qubit is then matrix multiplication. Several key properties follow from this definition:

1. Reversibility: Since $U^\dagger U = I$, unitary matrices have an inverse, their conjugate transpose. This allows quantum operators to be undone by an application of their conjugate transpose.

2. Conservation of Quantum Information: Unitary matrices are Euclidean isometries, meaning that these quantum operators are transforms on qubits that preserve the Frobenius norm. This means that the total probability in the quantum state remains 1 before and after application of such a unitary operator, reflecting conservation of quantum information.

These quantum operators serve as the mathematical foundation for *quantum gates*. By applying sequences of these unitary operations, we may maniplate qubits to perform more complex tasks.

2.4.1 The Pauli Group

The Pauli matrices are quantum operators that are used extensively in quantum computing. Each of these matrices represent a fundamental quantum gate that operates on a single qubit. They represent rotations of the qubit's state on the aforementioned Bloch sphere.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The X gate is analogous to the classical NOT operation. It flips the state of a qubit, performing a π-rotation about the x-axis of the Bloch sphere.The Y gate represents a phase shift of the qubit. Although it also flips the state of the qubit, it instead performs a π-rotation about the y-axis of the Bloch sphere. The Z gate affects a qubit's phase without changing its amplitude. It corresponds to a π-rotation about the z-axis of the Bloch sphere.

The Pauli group, denoted as P_n for a system of n qubits, is a group consisting of all possible tensor products of Pauli matrices, along with the identity matrix I. The elements of the Pauli group anticommute. For $\sigma_a, \sigma_b \in P_n$, these elements have the anticommutation relation $\{\sigma_a, \sigma_b\} = \sigma_a\sigma_b + \sigma_b\sigma_a$. The Pauli group is involutory, each element is its own inverse.

2.4.2 Other Gates

Aside from the Pauli rotation gates, there exist some other widely used gates.

1. **Hadamard Gate**: The Hadamard gate is a single qubit gate denoted by H, creating superposition states from basis states. Its matrix representation is given below:

   $$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

   When applied to the state $|0\rangle$ or $|1\rangle$, it produces the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $(|0\rangle - |1\rangle)/\sqrt{2}$, respectively. Notably, conjugation of the X gate via the Hadamard gate produces the Z gate; $Z = HXH$.

2. **Phase-Shift Gate**: The phase-shift gate applies a phase shift ϕ to a qubit. It applies this phase ϕ to the $|1\rangle$ component of a quantum state, leaving the $|0\rangle$ component unchanged. Its matrix representation can be given below:

   $$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}.$$

   Some commonly used phase-shift gates include the $Z = R_\pi$ gate, the $S = R_{\pi/2}$ gate, and the $T = R_{\pi/4}$ gate.

3. **CNOT Gate**: The CNOT, or controlled-not gate applies the XOR operation conditionally to a qubit based on the state of another qubit. It is a two qubit gate with the

following matrix representation:

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

The CNOT is very important, as it can create entanglement between two qubits.

Notably, the Pauli gates, along with the phase-shift gate and the CNOT gate form a viable set of universal quantum gates. This means that any quantum process can be approximated via a sequence of operations from this universal set. This gate set allows for the means to freely design quantum circuits.

2.4.3 The No Cloning Theorem

Theorem 1 (The No Cloning Theorem [2])**.** *Given two quantum states $|\psi\rangle, |e\rangle \in H$, there does not exist a quantum operator U on $H \otimes H$ such that*

$$U|\psi\rangle|e\rangle \rightarrow |\psi\rangle|\psi\rangle.$$

This theorem implies that there is no general way to copy a quantum state. This theorem will have great consequences for the design of quantum error correction schemes.

2.5 Measurement

Quantum measurements follow the *Born rule*, which states that given a measurement operator M_m, the probability of obtaining some outcome m in a quantum state $|\psi\rangle$ can be computed by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$$

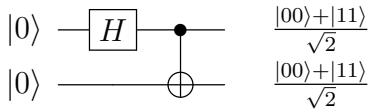
Following the measurement, the state $|\psi\rangle$ projected to a new state $|m\rangle$, corresponding to the outcome m.

$$|m\rangle = \frac{M_m|\psi\rangle}{\sqrt{p(m)}}.$$

For example, take the state $|\gamma\rangle = \alpha|0\rangle + \beta|1\rangle$. When an identity measurement is performed on $|\gamma\rangle$, e.g. $|\gamma\rangle$ is measured under the measurement operator I, the result could either be 0 with probability α^2, resulting in a projection of $|\gamma\rangle$ to the $|0\rangle$ state, or 1 with probability β^2, resulting in a projection of $|\gamma\rangle$ to the $|1\rangle$ state.

2.6 Quantum Circuits

Under the gate-based quantum computing framework, the building blocks laid out in the previous sections can be used together to create quantum circuits. The following is an example of a quantum circuit producing the previously mentioned Bell state. A register of qubits with states initialized to $|0\rangle$ are indicated on the left hand side of the circuit. Wires extend through the circuit. Gates are applied to these qubits, from left to right, with a gate drawn over a wire indicating an application of that gate to the corresponding qubit. Single qubit gates may be drawn as the single letter abbreviation for the gate, enclosed in a block. CNOT gates are drawn with an $\oplus$ indicating the target bit, and a solid dot indicating the control bit. A measurement operation takes the form of a gauge drawn in a box.



Chapter 3

Noise and Quantum Error Correction

In comparison to bits in classical computers, qubits experience noise at a much higher rate than their classical counterparts. Therefore, to achieve useful, scalable quantum computation, these qubits have to be error corrected. This chapter lays out the fundamentals of quantum error correction, along with

3.1 Mixed States

In many cases, mathematically expressing quantum states is not as simple as just assigning a single ket vector. Here, we may consider the use of a *density matrix*. For a pure state $|\psi\rangle$, its corresponding density matrix ρ is given by

$$\rho = |\psi\rangle\langle\psi|.$$

Such a density matrix is positive semi-definite, with its trace $\mathrm{Tr}(\rho)$ equal to 1. A *mixed state* then corresponds to a probabilistic mixture of these density matrices. Consider the following example:

3.2 Quantum Channels

Just as the pure state was inadequate for fully representing a more general quantum state, necessitating the introduction of the mixed state, the previously introduced unitary operator representation is also inadequate for representing quantum operations, especially in the presence of noise. Noise in a quantum operation may be throught of as a probabilistic deviation from the expected action. The resulting state could vary between applications of the same noisy operation. Motivated by this phenomenon, we introduce the *quantum channel*, which acts on mixed states, not just pure states.

Note that in quantum mechanics, the time evolution of a pure state in a closed, isolated quantum system under a time-independent Hamiltonian H is governed by Schrödinger's equation:

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle \Rightarrow |\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle.$$

Here, the Hamiltonian is an operator corresponding to the total energy in the system. The time evolution of density matrices are similarly governed by the Liouville-von Neumann equation:

$$i\hbar\frac{\partial\rho}{\partial t} = [H, \rho].$$

Quantum channels, formally known as Completely Positive Trace-Preserving (CPTP) maps, are transformations that describe the evolution of the state of a quantum system, incorporating the effects of environmental interactions. These maps are defined to be completely positive, ensuring that they do not introduce any physical impossibilities into the state dynamics, and trace-preserving, which guarantees that the total probability distribution of the state remains constant. Importantly for quantum error correction, these CPTP maps capture the effects of various noise types and interactions that are not accounted for by simple unitary evolution.

3.2.1 The Bit-flip Channel

The quantum bit-flip channel is analogous to the classical bit flip in digital communications where a bit 0 might randomly flip to 1, and vice versa. In quantum computing, the bit-flip channel affects qubits by flipping their state from $|0\rangle$ to $|1\rangle$ and vice versa with a certain probability p.

The mathematical description of the bit-flip channel $\mathcal{E}$ applied to a quantum state represented by a density matrix ρ can be given by:

$$\mathcal{E}(\rho) = (1-p)\rho + pX\rho X$$

where X is the Pauli-X matrix. As mentioned previously, the X matrix swaps the amplitudes of the states $|0\rangle$ and $|1\rangle$, effectively flipping the state of the qubit. The probability p represents the likelihood of the bit-flip error occurring, while $1-p$ represents the probability that the state remains unchanged. Note that we may also define the phase-flip and bit-phase-flip channels accordingly, for the Z and Y gates replacing X, respectively.

3.2.2 The Depolarizing Channel

The depolarizing channel represents a type of noise that uniformly randomizes the state of a qubit, driving it towards a completely mixed state. This channel is characterized by its simplicity and symmetry, making it widely used in theoretical studies of quantum communication and computation.

For a single qubit, the depolarizing channel can be described by the equation:

$$\mathcal{E}(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z)$$

where p is the probability of depolarization occurring within a given time interval, and the X, Y, and Z Pauli matrices. The effect of this channel is to replace the state ρ with the maximally mixed state $\frac{I}{2}$ with probability p, while the system remains in its original state with probability $1-p$. The addition of $\frac{p}{3}$ to each Pauli operation ensures that the trace of ρ

is preserved, adhering to the properties of a completely positive and trace-preserving map. This channel thereby introduces isotropic noise to the system, reducing the qubit's purity regardless of its initial state.

3.3 Stabilizer Codes

Detecting quantum errors is difficult. There are several properties of quantum states which complicate the process, preventing a direct application of classical error correction techniques.

1. The No Cloning Theorem — As mentioned in section 2.3, general quantum states cannot be copied through a single general unitary process. This precludes the classical technique of redundancy, as you cannot copy the quantum states holding data. Thus, any effective quantum error correction process must be able to detect and correct errors in a quantum system without copying any of the constituent quantum states.

2. Projective quantum measurements — As mentioned in section 2.4.3, quantum measurements "destroy" the quantum state, producing a basis state as the measurement result, and projecting the measured quantum state into that basis state. This rules out directly measuring qubits, as it would destroy the data they hold. Therefore, any effective quantum error correction process cannot directly measure qubits holding data as you would do to their classical counterparts in classical error correction.

A stabilizer code [3] is a type of quantum error correction code that successfully circumvents these obstacles. Stabilizer codes consist of an arrangement of physical qubits holding data, along with extra physical *ancilla qubits*. These physical qubits, data and ancillae, are entangled together according to the design of the code. The purpose of these ancilla qubits are to give a feasible way to detect errors without measuring any qubits holding data. In a stabilizer code, measurement of these ancilla qubits produce a *syndrome measurement*, the parity of which can be used by a decoder to produce error corrections [4].

3.3.1 Stabilizers

Central to stabilizer codes is the *stabilizer*. I begin our discussion of stabilizers with an illustrative example, first presented in [4]. Consider a two qubit system with qubits a and b. For convenience, I define the following operations:

$$Z_a = Z \otimes I, Z_b = I \otimes Z,$$
$$X_a = X \otimes I, X_b = I \otimes X.$$

First we note that operations on different qubits commute. For X_aX_b and Z_aZ_b, we have:

$$\begin{aligned}[X_aX_b, Z_aZ_b] &= (X_aX_b)(Z_aZ_b) - (Z_aZ_b)(X_aX_b) = X_aZ_aX_bZ_b - Z_aX_aZ_bX_b \\ &= (-Z_aX_a)(-Z_bX_b) - Z_aX_aZ_bZ_a = 0.\end{aligned}$$

Since commuting matrices share an eigenvector, there must exist a simultaneous eigenstate $|\psi_{ab}\rangle$ for both X_aX_b and Z_aZ_b. Under these two operator products, this simultaneous eigenstate is unique. This implies that repeatedly measuring $|\psi_{ab}\rangle$ under both the X_aX_b and Z_aZ_b

operators will not change the state. Without loss of generality, and for simplicity, let $|\psi_{ab}\rangle$ be the $(+1,+1)$ eigenstate for X_aX_b and Z_aZ_b. That is, $|\psi_{ab}\rangle$ has eigenvalues of $+1$ for both X_aX_b and Z_aZ_b. Now, the action of a depolarizing channel applying an extraneous X_a gate to $|\psi_{ab}\rangle$. Then, we have the following:

$$X_aX_bX_a|\psi_{ab}\rangle = X_aX_aX_b|\psi_{ab}\rangle = X_a|\psi_{ab}\rangle,$$
$$Z_aZ_bX_a|\psi_{ab}\rangle = -X_aZ_aZ_b|\psi_{ab}\rangle = -X_a|\psi_{ab}\rangle.$$

Notice that this external X error changes $|\psi_{ab}\rangle$ from a $(+1,+1)$ eigenstate to a $(+1,-1)$ eigenstate of X_aX_b and Z_aZ_b, flipping the eigenvalue for Z_aZ_b. We may also consider the same example for a Z error applying an extraneous Z_b gate to $|\psi_{ab}\rangle$. As before, we have the following:

$$X_aX_bZ_b|\psi_{ab}\rangle = -Z_bX_aX_b|\psi_{ab}\rangle = -Z_b|\psi_{ab}\rangle,$$
$$Z_aZ_bZ_b|\psi_{ab}\rangle = Z_bZ_aZ_b|\psi_{ab}\rangle = Z_b|\psi_{ab}\rangle.$$

In both of these cases, this induced change resulting from the external bit-flip flips the sign of one of the two eigenvalues of $|\psi_{ab}\rangle$, modifying the final measurement result. Repeatedly measuring a complete set of these operator products will allow us to determine whether or not an error has occured on these two qubits. This is quite useful. However, if the set of operator products is not complete, the simultaneous eigenstate of these operator products is not guaranteed to be unique.

In the context of quantum error correction, these X_aX_b and Z_aZ_b operator products are known as *stabilizers* for qubits a and b. As we can see, applications of these stabilizers have the potential to facilitate error correction, but there are some caveats. Note the effect of a depolarizing X error applying a X_b:

$$X_aX_bX_b|\psi_{ab}\rangle = X_bX_aX_b|\psi_{ab}\rangle = X_b|\psi_{ab}\rangle,$$
$$Z_aZ_bX_b|\psi_{ab}\rangle = -X_bZ_aZ_b|\psi_{ab}\rangle = -X_b|\psi_{ab}\rangle.$$

This maps our eigenvector from a $(+1,+1)$ eigenstate to a $(+1,-1)$, which is exactly the same outcome as the previous example applying an erroneous X_a! This implies that our setup of stabilizers is not enough to completely determine the errors that aflict this system.

In this example, even though the stabilizers, specific operator products, allow us to detect some errors, the simple setup involved does not allow us to correct them. True quantum error correction will require a more complex setup.

3.3.2 Stabilizer Formalism

We may now formally define stabilizers and stabilizer codes. I will use the definition given by Gottesman [5]. A stabilizer code $[n,k]$ encodes k logical qubits into n physical qubits. The stabilizers of such a code form an Abelian subgroup S of the Pauli group $P_n =< I,X,Z >$ on n qubits. Recall from the example in the previous section that the commutation of the stabilizers allows for simulatenous eigenstates, which is useful for error detection. Therefore, S should be Abelian. The codeword $|\psi\rangle$ for such a stabilizer code, residing in a Hilbert space $\mathbb{H}_n$ of size n, is chosen to be a $+1$ simultaneous eigenstate of the stabillizers of the code.

Such a stabilizer code with set of stabilizers S will detect errors that are either in S or anticommute with an element of S. Under the influence of depolarizing noise, when an error $E \in P_n$ occurs on a data qubit, it introduces an extraneous Pauli operator that anticommutes with an element of the stabilizer group. This action flips the eigenvalue of the codeword $|\psi\rangle$ with respect to the anticommuting stabilizer. The measurement of the stabilizers on the ancilla qubits allows for the detection of changes in eigenvalue, which are indicative of errors on the physical qubits. The result of the measurements of the stabilizers at any one time is called the *syndrome* of the code. We can detect and potentially correct errors through a careful analysis of the syndrome via a decoding algorithm.

3.4 The Surface Code

Surface codes [4] are a family of stabilizer codes with a high error threshold, making them a suitable candidate for fault-tolerant quantum computation. Recent research demonstrates the experimental viability of this family of codes. Surface codes encode logical qubits into a two-dimensional lattice of physical data qubits arranged in a grid. Ancilla qubits are evenly interspersed throughout the lattice to aid in error correction. The stabilizers of the surface code are generated by the tensor product of the X or Z operators corresponding to *vertex* and *plaquettes* on the lattice of physical qubits. Then, for X_e denoting an application of X to qubit e, we have $\hat{X}_s = \bigotimes_{e \in v} X_e$ denoting the vertex stabilizer generator acting on the qubits at the vertex site v. The plaquette stabilizer generator is similarly defined, for $\hat{Z}_p = \bigotimes_{e \in p} Z_e$ acting on the qubits at plaquette site p. The stabilizer group is as follows:

$$S =< \hat{X}_v, \hat{Z}_p > \subseteq P.$$

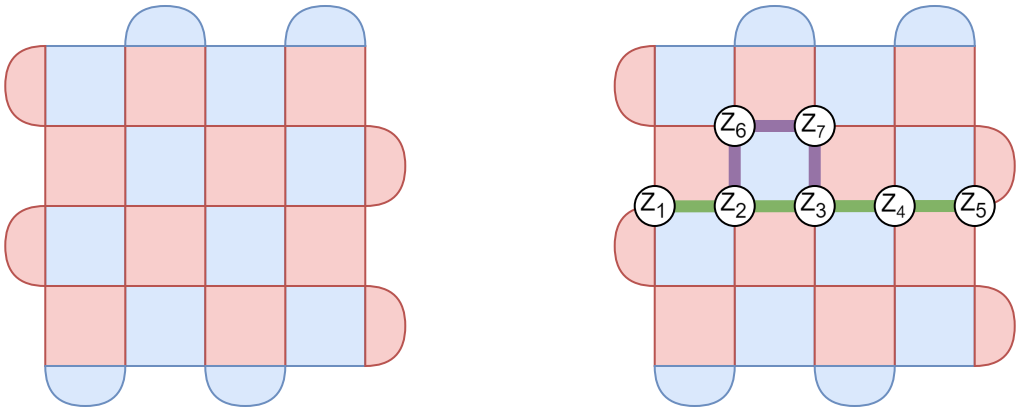
A codeword $|\psi\rangle$ encoded into the surface code and stabilized by S is a simultaneous eigenstate of these stabilizers. The codespace is then defined as follows:

$$T = \{|\psi\rangle \in (\mathbb{C}^2)^{\otimes n} : \hat{X}_v|\psi\rangle = \hat{Z}_p|\psi\rangle = |\psi\rangle, \forall v, p\}.$$

Notable examples of codes in this family are the toric code, rotated planar code, and XZZX code.

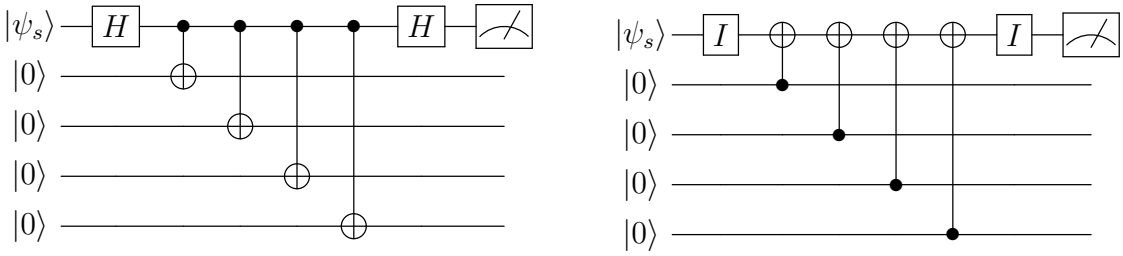
3.4.1 The Rotated Planar Code

This work will focus on the rotated planar code. The rotated planar code is constructed by rotating the standard square lattice of a planar code by 45 degrees. This rotation changes the lattice geometry to a diamond shape, effectively reducing the boundary conditions and the number of qubits needed to construct the code, when compared to a planar code. See Figure 3.1a for an example of a rotated planar code. Each colored square in the rotated planar code represents a single stabilizer qubit, with red and blue denoting X and Z-type stabilizers, respectively. The data qubits lay at the intersection of the grid lines. The rotated planar code is operated by measuring all of the stabilizer qubits in lockstep. See Figure 3.2 for the different types of stabilizer measurement circuits.



(a) A $d = 5$ Code (b) Equivalent Chains

Figure 3.1: The Rotated Surface Code



(a) X-stabilizer Measurement Circuit (b) Z-stabilizer Measurement Circuit

Figure 3.2: X and Z Measurement Circuits for the Rotated Planar Code

The code's stabilizers are products of Pauli operators that, involving either four or three qubits in the interior or at the boundaries, respectively. Typically, one logical qubit is encoded per lattice, with logical operators spanning from one edge of the lattice to the opposite edge along the shortest path. This is a product of Pauli operators that commutes with the stabilizers, and manipulates the two degrees of freedom in the code. Under the presence of error, a logical error consists of a chain of extraneous noise operators that take the form of a logical operation; the boundary-to-boundary chain of operators. Note that homologically equivalent chains, ones that have undergone a trivial deformation by adding stabilizer operators to the chain of products, have the same effect on this code. Therefore, any deformation of the original logical operator by adding stabilizer operators will have the same effect on the code. See Figure 3.1b for an example. In the diagram, $Z_1Z_2Z_3Z_4Z_5$ is logically equivalent to $Z_1Z_2Z_6Z_7Z_3Z_4Z_5$.

3.5 Decoding the Surface Code

Overview of Decoding in Quantum Error Correction Explanation of what decoding entails in the context of quantum error correction. The significance of decoding for the performance

and feasibility of quantum computers.

3.5.1 Maximum Probability Decoding: The MWPM Decoder

3.6 Maximum Probability Decoders

Maximum probability decoders find a recovery operator E which is maximally probable based on the observed syndrome. One popular example of such a decoder is the minimum weight perfect matching (MWPM) algorithm. This decoder solves the MPWM problem on a *decoder graph* G consisting of ancilla qubits of one type as vertices, and edges weighted by a metric, such as the Manhattan distance between such ancilla qubits in the surface code lattice. For example, the X-type stabilizers for the surface code may be chosen as the vertices in G. The decoder will also solve the same problem on the graph consisting of Z-type stabilizers. The edge weights of such a G may be appropriately modified to reflect the probability distribution of errors afflicting the code. A challenge that MWPM decoders have to overcome is the computational cost associated with solving the combinatorial optimization problem, especially on dense syndromes. Even so, there exist efficient implementations of these decoders, leveraging various characteristics of the decoding problem to greatly accelerate decoding [6]–[8]. See Section 5.1 for a closer look more of this type of decoder in the literature.

3.7 Quantum Computation Using the Surface Code

How are surface codes used for quantum computation? The answer to this question will provide useful details and requirements for constructing an effective decoder for the surface code. As mentioned before, a surface code can be treated as a single logical qubit. The X and Z operations are boundary-to-boundary chains of X and Z operations on individual qubits.

As mentioned in [9], a functional quantum computer at the logical level would consist of data blocks, which store logical quantum information, and distillation blocks, which are used for creating arbitrary logical quantum states.

In specific, the need for low latency in a decoder arises implementations of the logical phase-shift gate.

3.8 Modeling Superconducting Quantum Systems

Physically, what do quantum errors look like? The physical principles underlying qubits will inform the error models used later in this thesis. There are many types of qubits, including transmon, photonic, and trapped ion. Among the various types of qubits, superconducting qubits stand out due to their scalability and the advanced level of control achieved in recent experiments. As such, this work will just focus on the superconducting variety.

Superconducting quantum computers use Josephson junctions, thin insulating barriers that, with connected inductance and capacitance and a nearby microwave resonator, begins

to function as a quasi-particle at superconducting temperatures [10]. These quasiparticles exhibit quantized energy levels, allowing for the discrete quantum states necessary for quantum computing. Therefore, such a Josephson junction can then be treated as a single qubit. Recently, IBM has released a superconducting quantum processor with 1,124 qubits find citation!, and Google Quantum AI has published promising results utilizing the superconducting modality [11].

Chapter 4

Deep Learning

This chapter will serve as a primer for deep learning.

4.1 The Neural Network

At a very high level, neural networks are large parameterized black-box functions, whose parameters may be updated through gradient descent. The architecture of these networks typically includes input, hidden, and output layers that facilitate data processing and learning. There are various types of network architectures, notably *feedforward*, where data moves in one direction, and *feedback*, where outputs are looped back into the system as inputs. Training neural networks involves learning from data, which is achieved through defining loss functions and applying optimization algorithms like backpropagation and gradient descent. These methods adjust the weights and biases to minimize errors in predictions. Neural networks have found widespread applications across diverse fields such as image recognition and natural language processing.

4.1.1 Basic Components

Linear Layer

A linear layer, also known as a fully connected, dense layer, or affine layer, is a fundamental component of many neural networks. It is a linear transformation, where each input is multiplied by a weight and then a bias is added. Mathematically, this is represented as $y = Wx + b$, where x is the input vector, W is the weight matrix, b is the bias vector, and y is the output vector. The purpose of a linear layer is to map the input data into a space where it can be more easily classified or analyzed by subsequent layers, especially when combined with nonlinear activation functions that follow the linear layer.

4.1.2 Backpropagation

4.1.3 Gradient Descent

4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks that are particularly powerful for processing data with a grid-like topology, such as images. CNNs are distinguished by their use of convolutional layers, which adaptively learn spatial hierarchies of features. This makes them efficient for tasks like image recognition, video processing, and medical image analysis.

The primary feature of CNNs is the convolutional layer, which applies a convolution operation to the input, passing the result to the next layer. This operation captures the spatial dependencies in the input data by learning from small regions of the input, known as receptive fields. Unlike fully connected networks that learn patterns at a global scale, CNNs focus on local patterns, making them more efficient and requiring fewer parameters.

A typical CNN architecture consists of several layers that transform the input volume into an output volume (e.g., class scores) through differentiable functions. These include the aforementioned convolutional layers, which learn local feature representations, pooling layers, which help to reduce the dimensionality of each feature map, and fully connected layers, which compute class scores and output the prediction of the network. Each layer of neurons applies different filters, typically learned during the training process, and downsampling techniques like max pooling to reduce the spatial size of the representation.

Training CNNs involves backpropagation and optimization algorithms such as stochastic gradient descent (SGD). However, training deep CNNs comes with challenges. Due to the high capacity of some of these models, they are prone to overfitting, particularly when trained on small datasets.

- **Computational Resource Requirements:** CNNs require significant computational resources, especially for training on large datasets with high-resolution images.

4.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks designed specifically for processing sequential data. Unlike feedforward neural networks, RNNs have the distinctive feature of maintaining internal memory or state that captures information about the sequence they have processed so far. This makes RNNs ideal for applications where context and time are crucial, such as language modeling and speech recognition.

Whereas traditional neural networks take the form of directed acyclic graphs, RNNs are cyclic by design giving them the ability to process data through loops. This allows information to persist in the state of the RNN, enabling it to make decisions based on a history of inputs up to the current step, unlike traditional neural networks which process each input in isolation.

An RNN typically consists of a layer of neurons with a self-loop that represents the time dimension, enabling a form of memory. The architecture can be described as follows: -

Input Layer: Receives sequences of data points. - **Hidden Layer:** Processes inputs using weights that are shared across time steps, maintaining a state or memory of past inputs. - **Output Layer:** Produces the output for each time step, which can depend on the current input and the current state of the hidden layer.

The key equation that defines the basic operation of an RNN is:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

where h_t is the hidden state at time t, x_t is the input at time t, W_{xh} are the weights from the input layer to the hidden layer, W_{hh} are the weights from the hidden layer to itself, b_h is the bias, and σ is the activation function, typically a sigmoid or tanh function.

Training RNNs involves backpropagation through time (BPTT), a variant of the standard backpropagation algorithm adapted for sequential data. This method unfolds the RNN through time and then applies backpropagation. However, BPTT faces several challenges: - **Vanishing Gradient Problem:** When gradients of the loss function become very small, making the weights update negligible and thus stopping the network from learning long-range dependencies. - **Exploding Gradient Problem:** Conversely, gradients can become very large, leading to unstable training processes.

To address the aforementioned limitations of basic RNNs, several variants have been developed. This includes the Long Short-Term Memory (LSTM) netw0rk, which incorporates gates that control the flow of information. This effectively allows the network to learn when to "remember" and when to "forget" states, thereby mitigating the vanishing gradient problem. Another popular type of RNN is the Gated Recurrent Unit, which is a simpler alternative to LSTMs that use a similar gating mechanism, but with fewer parameters.

RNNs are utilized in a range of applications that require an understanding of sequential patterns: - **Natural Language Processing (NLP):** Tasks such as machine translation, text generation, and sentiment analysis. - **Speech Recognition:** Transcribing spoken language into text. - **Time Series Prediction:** Forecasting future values in finance, meteorology, and more based on past data.

4.4 Transformer Networks

Transformer networks have revolutionized the field of deep learning, particularly in tasks involving natural language processing (NLP) and image processing. The advent of this architecture has sparked exponential growth in the number of deep learning publications in recent years. Developed as an alternative to the recurrent neural network (RNN) architectures, transformers address several shortcomings of RNNs, including difficulty in parallelizing operations and challenges in learning long-range dependencies in sequences. Key to the transformer's success is its use of the self-attention mechanism, which allows it to weigh the importance of different words within a sequence, irrespective of their positional distances.

Transformers utilize an encoder-decoder structure. The encoder multiple identical layers, each with two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. Layer normalization and residual connections are employed around each of the sub-layers. Similarly, the decoder also consists of a series of identical layers, but with an additional third sub-layer that performs

multi-head attention over the output of the encoder stack. Similar to the encoder, the decoder employs residual connections and layer normalization.

The Multi-Head Attention module, included in both encoder and decoder layers, allows the model to jointly attend to information from different representation subspaces at different positions. By doing this, the transformer can integrate information more effectively across the sequence. By focusing this attention on the same sequence as the input, we get self-attention. The self-attention mechanism allows the model to focus on different parts of the input sequence as it processes each word. For each word, the transformer computes a score that signifies its importance relative to every other word. This approach enables the model to capture context more effectively and manage dependencies between words far apart in the input sequence. Since transformers lack recurrence, they use positional encodings to incorporate information about the position of words in the sequence. These encodings are added to the input embeddings at the base of the model and help preserve the order of the sequence.

4.4.1 Training and Optimization

Training transformers involves several techniques: - **Attention Masks:** Used in training to prevent positions from attending to subsequent positions. This masking ensures that the predictions for a position can depend only on the known outputs at previous positions. - **Layer Normalization:** Applied inside the encoder and decoder to help stabilize the network's training. - **Use of Large Datasets:** Transformers require substantial amounts of data to learn effectively, leveraging the availability of large-scale datasets. - **Computational Resources:** Due to their complex architecture, training transformers is computationally intensive, often requiring powerful hardware or specialized accelerators like GPUs or TPUs.

4.4.2 Impact and Applications

Transformers have led to significant breakthroughs in various fields: - **Language Understanding and Generation:** Transformer models like BERT and GPT have set new standards for performance in tasks such as machine translation, text summarization, and question-answering. - **Image Processing:** Extensions of transformer models to vision tasks have shown promising results, demonstrating their adaptability to domains beyond text, such as in image classification and object detection.

4.4.3 Structured Selective State Space (S4) Models

SSMs can be thought of as a hybrid between a recurrent and convolutional neural network. Typically, an SSM operates by mapping an input sequence through a series of state transformations. These transformations are defined mathematically by a set of parameters that determine how the input data is processed to produce the output. Typically, these parameters are matrices referred to as A, B, and C, where

- A represents the state transition matrix that describes how the hidden state evolves from one step to the next.

- B is the control matrix that maps the influence of the input sequence on the state.
- C is the output matrix that maps the hidden state to the output.

4.4.4 Mamba

Mamba is a recent iteration on the S4 model.

In comparison with previous SSMs, Mamba allows its parameters to vary based on the input. This innovation allows the model to selectively remember or ignore information along a sequence, thus optimizing the handling of data through the sequence. Another key element of the Mamba model is its hardware-aware design, which incorporates a parallel algorithm that operates in recurrent mode. This approach not only facilitates linear-time scaling in sequence length but also maximizes computational efficiency by avoiding the full materialization of expanded states across different GPU memory levels. Empirical evaluations demonstrate Mamba's effectiveness across different tasks, showcasing its ability to handle sequences up to a million elements long with significant improvements in throughput and performance compared to existing Transformer models.

Chapter 5

Related Works

This chapter will review some relevant decoders in literature. I broadly classify decoders into three categories, differentiated by which formulation of the decoding problem they solve: maximum probability decoders, maximum likelihood decoders, and machine learning decoders. The following sections will mention other important decoders in the previously mentioned class, and two new classes. Note that Section 3.5 already covers the maximum probability decoder as an example.

5.1 Maximum Probability Decoders

There also exist other matching decoders, which run belief propagation as a subroutine to reconcile the divide between the two types of stabilizers [12].

5.2 Maximum Likelihood Decoders

In comparison, maximum likelihood decoders search for the maximally likely **equivalence class** of errors based on the observed syndrome. An example of a maximum likelihood decoder is the tensor network decoder, which solves an approximation to this problem [13], [14].

5.3 Machine Learning Decoders

Introduction to Machine Learning in Decoding Overview of the use of machine learning (ML) techniques in quantum error correction. Types of ML models used for decoding (e.g., neural networks, reinforcement learning). Application to the Surface Code How ML decoders are specifically applied to the surface code. Comparison with traditional decoding methods. Challenges and Future Directions Discussion of the challenges in developing and training ML models for decoding. Potential for future research and advancements in ML decoders.

5.4 Other Decoders

Chapter 6

Methods

This chapter describes the methodologies employed to address the significant challenges of training neural network decoders for quantum error correction, particularly focusing on scaling issues related to the size of the code.

One of the primary challenges in training neural network decoders is the exponential increase in the amount of training data required to achieve high accuracy as the size of the quantum code increases. We propose two ways to approach this problem, both involving state compression. Here, we will refer to these two approaches as the convolutional encoding model and the transformer encoding model.

6.1 The Memory Experiment

Recall that under the paradigm of quantum computation on logical qubits, logical operations are implented via a combination of boundary measurements and maintaining the logical state prior to a logical measurement. The decoder mainly contributes to this second part. The memory experiment serves as a good analogue for the decoder's application to quantum computation at the logical scale. In a surface code memory experiment, the surface code is initialized to the $|0\rangle_L$ logical state. Here, we choose the logical Z operator as the extraneous operation to guard against. The experiment proceeds by running the surface code for r error correcting rounds. During each round, the system may be subjected to various types of errors. At the conclusion of these r rounds, a logical measurement is performed, checking whether or not the logical observable is still $|0\rangle_L$. The purpose of the decoder in this context is deciding whether the logical state has flipped or not, given the history of syndrome readouts.

6.2 Convolutional Encoding Model

The convolutional model treats the surface code as an "image", which it downscales via repeated applications of a convolutional autoencoder before further processing. This way, the neural network can theoretically be fine tuned to decode at arbitrary code-distances, without needing to pretrain a model from scratch. The model can simply take a large code, and compress the state repeatedly until it reaches a size that it was trained to decode at. In principle, this convolutional autoencoder functions in a manner similar to a renormalization

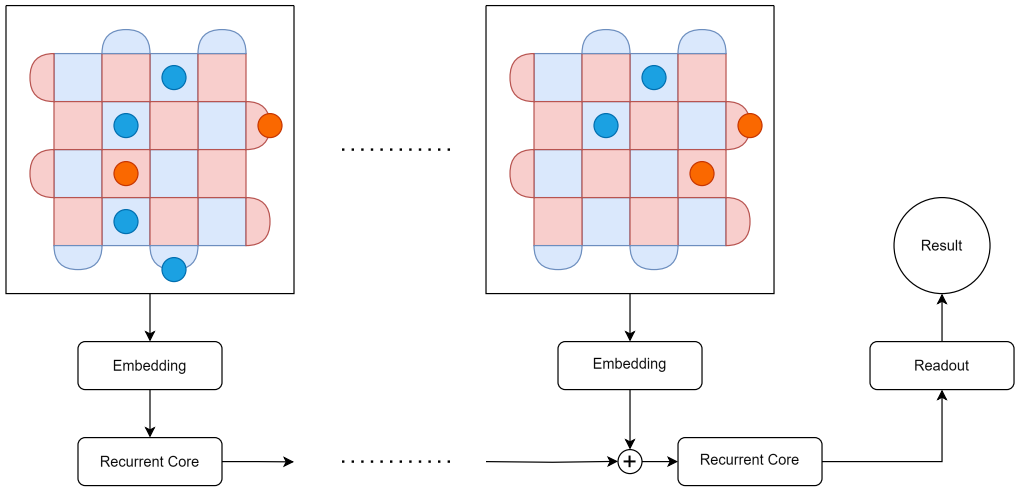


Figure 6.1: Mamba Decoder Model

group decoder [15], [16], where the decoding problem for the surface code can be scaled down using a combination of maximum likelihood decoding and belief propagation. Here, this is instead done on the model's hidden latent state, after embedding the input data.

6.2.1 Architecture

We train a recurrent neural network to decode surface code memory experiments, with architecture inspired by [17]. It consists of several smaller networks, including an embedding model, a recurrent core, and a readout network, which all work together in tandem to decode the surface code. Given the history of syndrome measurement data, it will produce a probability $p \in [0, 1]$ that the logical observable has flipped in the duration of the experiment.

Embedding

An embedding model can be thought of as a translator, translating the input data into a format that is comprehensible to the processing modules in the neural network. In the memory experiment, each round of measurements, or time step t, produces syndrome measurement outcomes with a binary 1 or 0 for each stabilizer in the surface code. For each of these stabilizer readouts, our RNN embeds the syndrome measurement outcomes through a linear projection. The two dimensional coordinates of each stabilizer on the surface code, are passed through a 2D sinusoidal positional embedding and added to this linear projection. This produces a vector in a high-dimensional latent space for each stabilizer.

These stabilizer embeddings are then scattered to a 2-dimensional grid of size $(d_e + 1) \times (d_e + 1)$, where d_e is the size of the experiment surface code. These stabilizers are placed in the same positions they would occupy on the surfac code. Then, this grid is passed into a fully convolutional autoencoder, consisting of a convolution that reduces the effective "distance" of 2-dimensional grid by two, and dilated convolutions. The number of times this autoencoder is run is based on the size of the surface code that is being decoded, and the "native distance" of the surface code that is being decoded. Since the autoencoder reduces

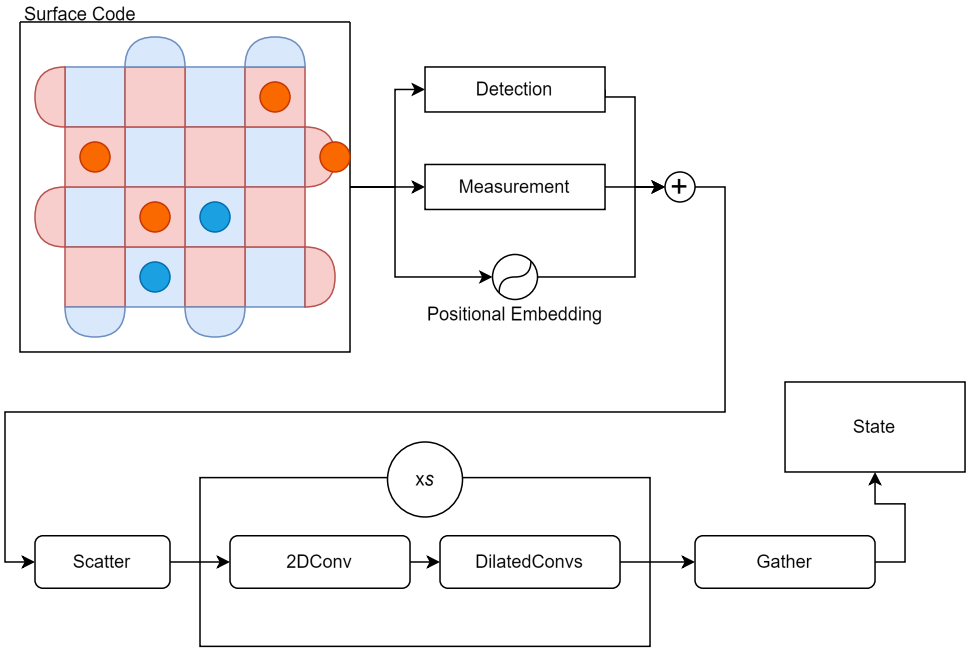


Figure 6.2: Embedding Model

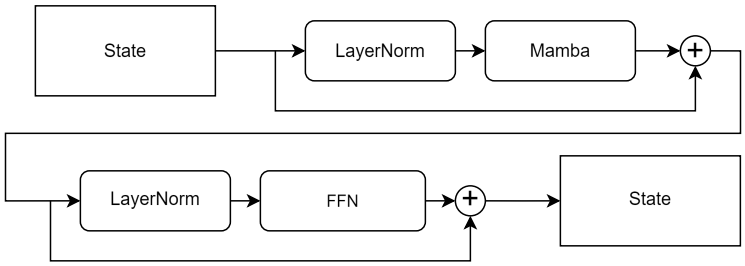


Figure 6.3: Recurrent Block

the size of the grid by two each time, it is run $(d_e - d_n)/2$ times, for d_n denoting the model's native distance. Through applications of this autoencoder, this grid-format latent space representation is scaled down to the RNN's "native" code distance where it is then gathered back into a sequence of updated stabilizer embeddings. This stabilizer embedding is added to the model's tracked hidden state.

Recurrent Core

The model tracks a hidden state concurrently with the progress of the memory experiment. This hidden state can be thought of as the model's "understanding" of the current state of the surface code at that timestep. Once the stabilizer embeddings are created for any one timestep, it is multiplied by a scalar quantity less than 1, and added to the model's tracked hidden state. The *recurrent block* resembles the encoder stack of the Transformer model. It conists of a gated Mamba model with layer norm, followed by a gated feedforward network with layer norm. Each time new stabilizers are encoded through the embedding model, the sum of the previous hidden state and the new embedding is passed through three recurrent

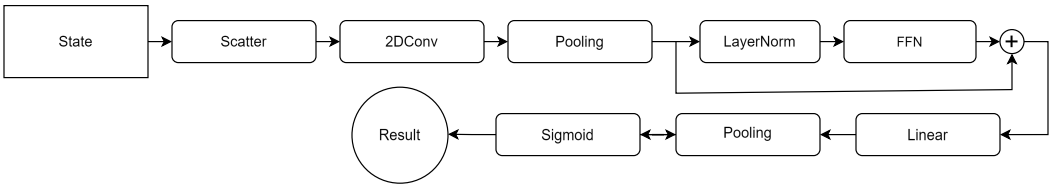


Figure 6.4: Readout Network

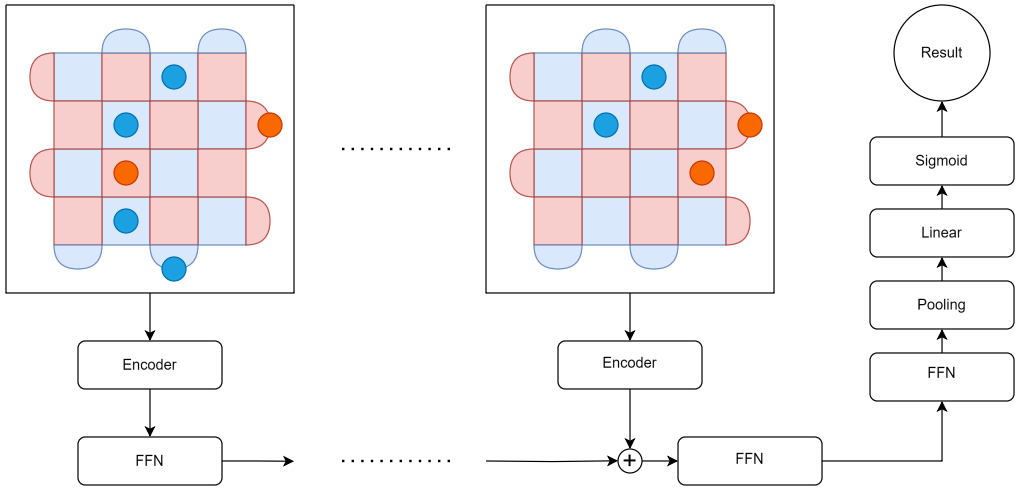


Figure 6.5: Transformer Decoder Model

blocks.

Readout Network

The readout network processes the model's hidden state to produce a prediction of whether the logical observable has flipped or not. First, the hidden state is scattered to a 2-dimensional grid of size $(d_n + 1) \times (d_n + 1)$, where d_n is the native distance of the decoder. Again, as in the embedding model prior to autoencoding, the stabilizers in the model's hidden state are scattered to the positions that they would normally occupy on the surface code. A 2D convolution layer convolves the stabilizer representation to instead represent data qubits, on a $d_n \times d_n$ grid. Then, this representation is passed through an average adaptive pooling layer, pooling in the direction of logical operations. This state is now in the form of d_n vectors, which is passed through a gated feedforward network, and pooled again to form a single vector. Finally, a linear layer with an attached sigmoid function outputs the probability of a flip in the logical observable at that timestep.

6.3 Transformer Encoding Model

The transformer encoding model instead approaches the surface code decoding problem from the perspective of natural language processing. The same recurrent framework remains, but the rationalization differs. Each stabilizer is treated as a "word", and the syndrome readout from each round is treated as a "sentence". We may apply an encoder network to produce a

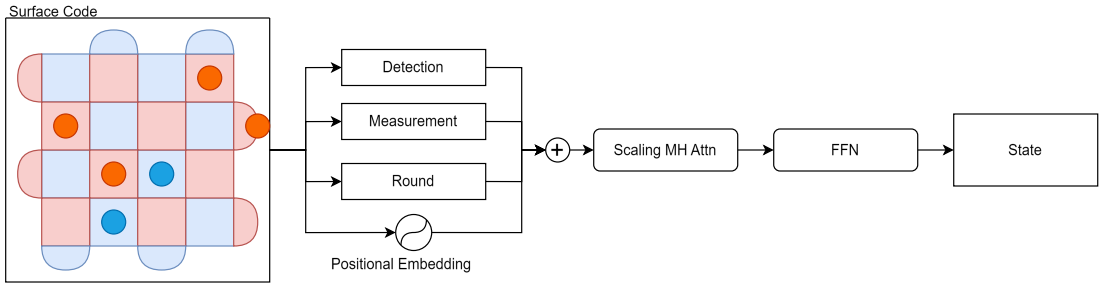


Figure 6.6: Encoding with Attention

"sentence embedding" for each round of measurement, which is added to the model's hidden state.

6.3.1 Architecture

The architecture of the transformer encoding model is very similar to the previous architecture. See Figure 6.5 for an overview of the architecture for this model.

Embedding and Encoding

As before, embeddings are generated for the stabilizers, consisting of the syndrome measurement and detection data for that round, along with a sinusoidal embedding representing encoding the stabilizers' positions in the code. In addition, we also embed a binary flag indicating whether or not it is the final round of decoding. Instead of compressing the state via convolutions, a multi-head attention mechanism instead maps these stabilizer embedding vectors to just a few vectors, which are then scaled up to a high dimension via a feed forward network. These high-dimensional vectors will then represent the state of the surface code at that round of syndrome measurement. See Figure 6.6 for a diagram of this encoder mechanism. Then, this high-dimensional representation can be added to the hidden state of the RNN. Here, the recurrent core that acts on the hidden state after modification is replaced by a feed-forward network. The readout network is also simplified, being a feed forward network followed by pooling to learn to reduce the set of vectors to a single high-dimensional vector, and a linear layer and sigmoid to scaling this single vector down to a probability.

6.4 Implementation and Training

Both networks were implemented using Pytorch, a Python library commonly used for training neural networks. See Appendix A for source code. We use the Mamba module provided on Github by the authors. The model was trained to decode the distance 11 surface code, afflicted with superconducting noise. The base error probability was set to 0.1%. The model was trained for 100 epochs, with 51200 Z-type memory experiments lasting 25 rounds each for all epochs. Since the training data was fully synthetic and could be generated at nominal cost, the training dataset size is effectively infinite. Therefore, overfitting is not a concern in this case; the model only sees each example once. However, to compare between various

iterations of the model, a validation dataset was generated using a fixed seed. The random number generator used to seed the training datasets was modified such that it will not generate the seed for the validation dataset. For training details, including hyperparmeters and loss curves, see Appendix B.

6.4.1 Stim

For training data, surface code memory experiments are generated using Stim [18] at nominal computational cost. Stim is a Python library that simulates quantum stabilizer circuits at high efficiency. Stim reads in and executes circuit objects, which contain definitions of a stabilizer circuit with Pauli noise annotations. We modify an existing script to generate data according to a superconducting (SI1000) noise model, defined in this paper [19]. Aside from plain depolarizing noise applied to qubits as a result of Clifford operations, it also applies depolarizing noise due to idling and resonator idling, mimicking the behavior of superconducting qubits subject to noise.

6.4.2 Efficient Training

As noted in [17], due to the similarities in architecture, the readout network can actually be called at any timestep during the memory experiment. If we generate data using the same psuedorandom seed once for each round of the memory experiment, we can collect final round measurement outputs and the logical observable for each round of the experiment. For a memory experiment of r rounds, training in a naive manner will require $r + 1$ calls to the stabilizer embedding model and recurrent core for a single output. Using this method, the same memory experiment will require $2r$ calls to the embedding model, recurrent core, and readout network to produce r outputs. Since the majority of computation is concentrated in the embedding model and recurrent core, this greatly reduces the amount of computation needed to train the model.

Chapter 7

Results

7.1 Evaluation

The model's performance was assessed using a surface code of distance 11. This code was subject to superconducting noise at varying base probabilities: 0.05%, 0.1%, 0.15%, and 0.2%, for 25 rounds. Validation datasets, each comprising 1024 trials, were generated for each base probability using the Stim simulation framework. The model was configured perform readouts after every round, meaning that the effective dataset size for each probability setting increased to 23,552 trials. The results of these evaluations are presented in Figure 7.1.

7.2 Discussion

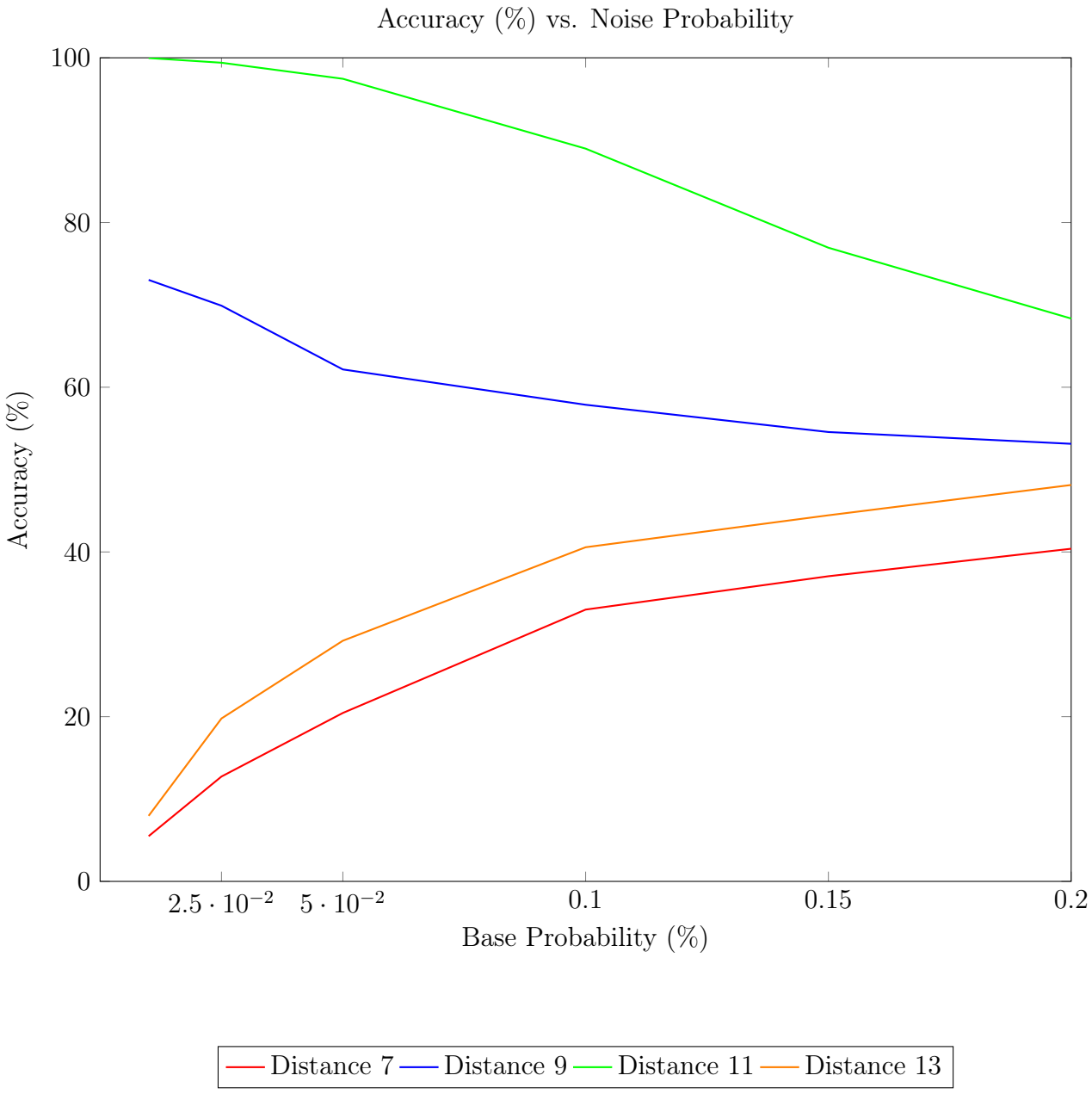
Testing the model at smaller code distances revealed a tendency for the convolutional autoencoder to overfit to the distance 11 codes it was trained on. This failure to generalize suggests that the autoencoder has not adequately learned to compress the syndrome information in general. One potential approach to reduce this overfitting could involve freezing the decoding layers of the network while further fine-tuning the convolutional layers. Alternatively, expanding the training dataset to include a variety of distance codes might help the network learn more generalizable features. Due to time constraints, it was not feasible to optimize the model for various distances during this study; thus, this task is recommended for future research.

We note that the model only saw about 110 million examples in total during training before convergence, 10x less examples than other comparable models in the literature [17], [20].However, we also note that this model is likely underfit, and may need more training to fully realize its potential. As the noise probability increased, there was pronounced performance degradation, meaning that the architecture is unable to handle high rates of noise, or more likely, that it is underfit for this level of noise. One possible avenue for improvement could be increasing the base error probability of the noise model as the neural network learns. This was tried briefly during our training, but was applied too aggressively, leading the models to diverge.

The threshold is a conventional metric for assessing decoder performance, where a decoder is benchmarked across various code distances and error rates. It represents the physical

error rate at which scaling the code distance will not lead to better error correction. Unfortunately, the testing necessary to establish a reliable threshold was not achievable within the limited timeframe of this project, as it requires the decoder function well at various distances and noise probabilities. This aspect of decoder evaluation also remains an area for future investigation. This was usually done in the literature by training a separate model for each distance.

Ideally, an ablation study would also be performed, to study the contributions of various submodules to the overall effectiveness of the model. However, similar time constraints prevented such a study from ocurring.


Accuracy (%) vs. Noise Probability
100
80
60
40
20
0
Accuracy (%)
$2.5 \cdot 10^{-2}$
$5 \cdot 10^{-2}$
0.1
0.15
0.2
Base Probability (%)
Distance 7
Distance 9
Distance 11
Distance 13

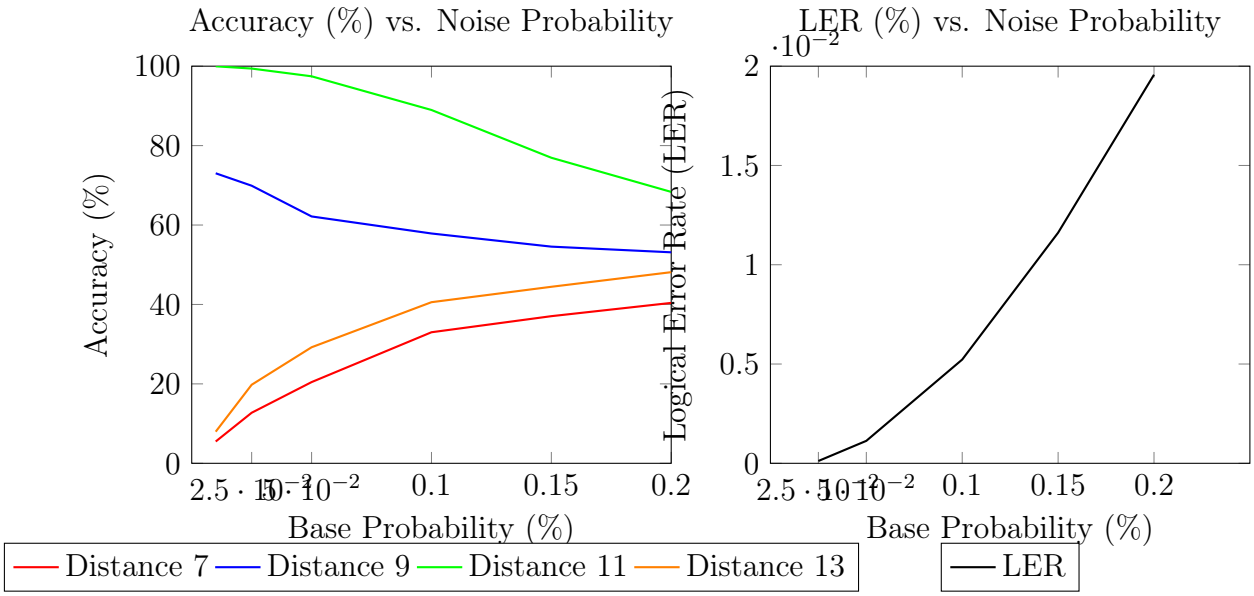


Figure 7.1: Comparison of model performance and error rates at varying base probabilities of noise.

Appendix A

Code listing

A.1 Convolutional Encoder Model

A.2 Transformer Encoder Model

Appendix B

Training Details

B.1 Hyperparameters

B.2 Loss Curves

References

[1] T. Young, "I. The Bakerian Lecture. Experiments and calculations relative to physical optics," vol. 94, Dec. 1804, ISSN: 0261-0523. DOI: 10.1098/rstl.1804.0001. (visited on 03/30/2024).

[2] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, Oct. 1982, ISSN: 1476-4687. DOI: 10.1038/299802a0. (visited on 05/02/2024).

[3] A. M. Steane, "Error Correcting Codes in Quantum Theory," *Physical Review Letters*, vol. 77, no. 5, pp. 793–797, Jul. 1996. DOI: 10.1103/PhysRevLett.77.793. (visited on 05/05/2023).

[4] 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, p. 032 324, Sep. 2012, ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.86.032324. (visited on 05/05/2023).

[5] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, May 1997. DOI: 10.48550/arXiv.quant-ph/9705052. arXiv: quant-ph/9705052. (visited on 03/26/2024).

[6] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, "Towards Practical Classical Processing for the Surface Code," *Physical Review Letters*, vol. 108, no. 18, p. 180 501, May 2012, ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.108.180501. (visited on 05/05/2023).

[7] O. Higgott and C. Gidney, *Sparse Blossom: Correcting a million errors per core second with minimum-weight matching*, Mar. 2023. DOI: 10.48550/arXiv.2303.15933. arXiv: 2303.15933 [quant-ph]. (visited on 07/14/2023).

[8] Y. Wu and L. Zhong, *Fusion Blossom: Fast MWPM Decoders for QEC*, May 2023. arXiv: 2305.08307 [quant-ph]. (visited on 07/11/2023).

[9] D. Litinski, "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery," *Quantum*, vol. 3, p. 128, Mar. 2019, ISSN: 2521-327X. DOI: 10.22331/q-2019-03-05-128. arXiv: 1808.02892 [cond-mat, physics:quant-ph]. (visited on 10/17/2023).

[10] O. Ezratty, "Perspective on superconducting qubit quantum computing," *The European Physical Journal A*, vol. 59, no. 5, p. 94, May 2023, ISSN: 1434-601X. DOI: 10.1140/epja/s10050-023-01006-7. (visited on 03/30/2024).

[11] R. Acharya, I. Aleiner, R. Allen, *et al.*, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, Feb. 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-022-05434-1. (visited on 05/05/2023).

[12] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, *Improved decoding of circuit noise and fragile boundaries of tailored surface codes*, Jul. 2023. arXiv: 2203.04948 [quant-ph]. (visited on 07/22/2023).

[13] C. T. Chubb, *General tensor network decoding of 2D Pauli codes*, Oct. 2021. DOI: 10.48550/arXiv.2101.04125. arXiv: 2101.04125 [quant-ph]. (visited on 08/29/2023).

[14] C. Piveteau, C. T. Chubb, and J. M. Renes, *Tensor Network Decoding Beyond 2D*, Oct. 2023. arXiv: 2310.10722 [quant-ph]. (visited on 12/23/2023).

[15] G. Duclos-Cianci and D. Poulin, *A renormalization group decoding algorithm for topological quantum codes*, Jun. 2010. arXiv: 1006.1362 [quant-ph]. (visited on 08/28/2023).

[16] K. Duivenvoorden, N. P. Breuckmann, and B. M. Terhal, "Renormalization group decoder for a four-dimensional toric code," *IEEE Transactions on Information Theory*, vol. 65, no. 4, pp. 2545–2562, Apr. 2019, ISSN: 0018-9448, 1557-9654. DOI: 10.1109/TIT.2018.2879937. arXiv: 1708.09286 [quant-ph]. (visited on 04/23/2024).

[17] J. Bausch, A. W. Senior, F. J. H. Heras, *et al.*, *Learning to Decode the Surface Code with a Recurrent, Transformer-Based Neural Network*, Oct. 2023. arXiv: 2310.05900 [quant-ph]. (visited on 12/19/2023).

[18] C. Gidney, "Stim: A fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, Jul. 2021. DOI: 10.22331/q-2021-07-06-497. (visited on 06/24/2023).

[19] C. Gidney, M. Newman, A. Fowler, and M. Broughton, "A Fault-Tolerant Honeycomb Memory," *Quantum*, vol. 5, p. 605, Dec. 2021. DOI: 10.22331/q-2021-12-20-605. (visited on 01/24/2024).

[20] H. Wang, P. Liu, K. Shao, D. Li, J. Gu, D. Z. Pan, Y. Ding, and S. Han, *Transformer-QEC: Quantum Error Correction Code Decoding with Transferable Transformers*, Nov. 2023. DOI: 10.48550/arXiv.2311.16082. arXiv: 2311.16082 [quant-ph]. (visited on 02/22/2024).