Use of Tensor Networks in Quantum Error Correction

A report presented in fulfilment of the requirements for the Supervised Learning Project

Lakshya Gadhwal

Project Advisor: Prof. Himadri Shekhar Dhar
Project Co-Advisor: Dr. Pavithran Iyer

7th November 2024

Contents

Abstract

Tensor Networks offer a promising approach for modeling and managing complex quantum systems. By representing the quantum state as an interconnected network of tensors and their contractions, this framework has the potential to enable more efficient computation and practical quantum error correction. In this study we begin with foundational concepts in quantum error correction, including stabiliser codes and their role in mitigating noise. We then use tensor networks to simulate a 1-d spin chain model and the integration of stabiliser codes. Additionally, the simulation of a 7-qubit Steane code highlights the efficiency of sampling error syndromes using tensor networks. The results demonstrate that tensor networks not only improve computational efficiency but also provide a structured and scalable approach to error correction, advancing the development of fault-tolerant quantum systems.

Chapter 1

Introduction

1.1. Motivation and Context

Quantum Computing is often named as the next frontier, promising breakthroughs in materials science, cryptography and machine learning. However, these supposed benefits come with their own set of hurdles. The challenge of maintaining accurate quantum states in the presence of unavoidable noise and error has been the major roadblock preventing us from realising commercial quantum systems. Quantum bits, or qubits, are extremely sensitive. Even when isolated and kept in an extremely low temperatures, they tend to interact with the surroundings, causing decoherence.

There have been many studies and attempts to develop large scale quantum error correcting codes [1][2][3], but none of them are anywhere near feasible enough to implement Shor's algorithm to break RSA, which requires atleast a couple hundred logical qubits to break [4]. The best known Quantum Error Correcting code today has only come close to implementing a handful of logical qubits. Also note that the number of physical qubits required grows exponentially with the number of logical qubits [5].

Tensor Networks provide a new and promising approach to modeling and managing large many-body quantum systems. Unlike traditional methods that represent quantum states as a single exponentially large vector, tensor networks factorize the state into a network of smaller tensors. This allows for more efficient and scalable computation, including the potential for practical quantum error correction[6][7][8]. The tensor network framework is well-suited for describing the structure and dynamics of strongly correlated quantum systems, which is essential for developing fault-tolerant quantum computers. By representing the quantum state as an interconnected network of smaller tensors, tensor network methods can capture the necessary complexity while avoiding the exponential growth that exists in other techniques. This makes tensor networks a compelling candidate for tackling the challenges of quantum error correction and bringing us closer to realizing the potential of quantum computing.

In this report we start by discussing the basics of quantum states, density matrices and quantum circuits and then eventually discuss Quantum Error Correction and Stabilizer formalism. We then move onto the topic of Tensor Networks and discuss the framework in the context of Quantum Error Correction (QEC) with examples and code to demonstrate the working and mechanism of the model and its equivalence to the encoding and decoding phases in QEC.

Chapter 2

Quantum Error Correction

2.1. Quantum States, Density Matrices, and Circuits

In quantum mechanics, the state of a quantum system is described by a *wavefunction* or *state vector*, which encapsulates all the information about the system [9]. The state vector is represented mathematically as a complex-valued vector in a Hilbert space. For a quantum system with n possible states, the state vector is a vector of 2^n complex numbers. The state of a single qubit, the fundamental unit of quantum information, is represented by a two-dimensional state vector. The general form of a single-qubit state vector is:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are complex numbers that satisfy $|\alpha|^2+|\beta|^2 = 1$. The states $|0\rangle$ and $|1\rangle$ are the *computational basis states* and represent the two possible classical states of the qubit. Instead of using state vectors, the state of a quantum system can also be described by a *density matrix*, which is a $2^n \times 2^n$ Hermitian matrix that encodes the same information as the state vector. The density matrix for a single qubit is:

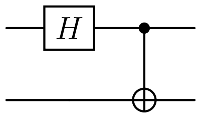
$$\rho = \begin{bmatrix} |\alpha|^2 & \alpha\beta^* \\ \alpha^*\beta & |\beta|^2 \end{bmatrix}$$

The density matrix formalism is particularly useful for describing mixed states, which are statistical mixtures of pure states. Mixed states arise when a quantum system is entangled with its environment, causing the system to lose its coherence.

Quantum computations are performed by applying *quantum gates* to the state of the system. Quantum gates are represented by unitary matrices that transform the state vector or density matrix of the system. For example, the Hadamard gate, denoted by H, is a single-qubit gate that transforms the state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and the state $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. The matrix representation of the Hadamard gate is:

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

Quantum circuits are diagrams that represent the sequence of quantum gates applied to a quantum system. Each qubit is represented by a wire, and the quantum gates are represented by boxes or symbols on the wires. The order of the gates in the circuit determines the overall transformation of the quantum state. For example, the following quantum circuit applies a Hadamard gate to a single qubit, followed by a controlled-NOT (CNOT) gate, which flips the state of a target qubit if the control qubit is in the $|1\rangle$ state:



The state of the quantum system evolves through the application of these quantum gates, allowing for the implementation of quantum algorithms and the exploration of quantum phenomena.

2.2. Quantum Error Correction Basics

Quantum error correction builds on the ideas of classical error correction, but with some important differences due to the unique properties of quantum mechanics. In a quantum system, the state of a qubit cannot be perfectly copied due to the no-cloning theorem. This means that traditional error correction techniques cannot be directly applied to quantum information.

One of the first quantum error correction codes is the 3-qubit bit-flip code. In this code, a single logical qubit is encoded into three physical qubits. The encoding is as follows:

$$|0\rangle_L \to |000\rangle$$
$$|1\rangle_L \to |111\rangle$$

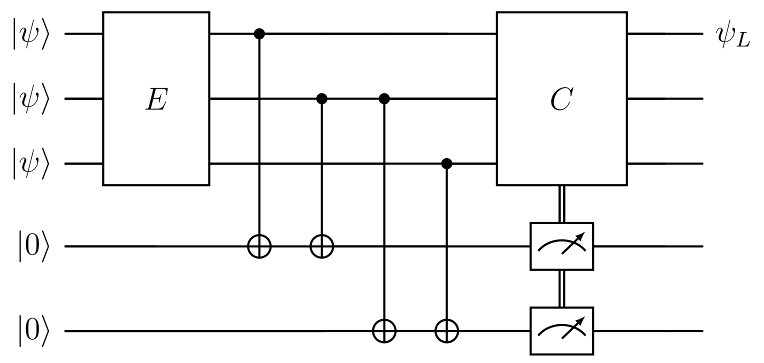
Suppose the initial state of the system is $|\psi\rangle_L = \alpha|0\rangle_L + \beta|1\rangle_L$. After encoding, the state becomes:

$$|\psi\rangle = \alpha|000\rangle + \beta|111\rangle$$

Now, let's consider what happens if a bit-flip error occurs on one of the qubits. For example, if the second qubit flips from 0 to 1, the state becomes:

$$|\psi'\rangle = \alpha|010\rangle + \beta|101\rangle$$

To detect and correct this error, we can measure the parity of the three qubits. If the parity is even (000 or 111), we know there is no error. If the parity is odd (010, 101, or 011), we know that a single-bit error has occurred, and we can correct it by flipping the appropriate qubit.



The above circuit shows one possible implementation of the error detection mechanism of the three qubit repetition code. E is a "Error" quantum circuit that randomly applies an X gate on one of the three qubits and C is a "Correction" quantum circuit that applies an X or I gate on one of the three qubits based on the measurement outcome of the two ancilla qubits, denoted by the meter symbols and the vertical wires coming out from them. If the measurement outcome of the 4th and 5th qubit is 0 and 0 respectively then we apply I to all qubits (i.e. do nothing). If the outcome is 10 then apply X on the first qubit and I on the other two. Similarly, for 01 apply X on third qubit and for 11 apply X on second qubit. The reasoning behind this is that if the second qubit of $|\psi\rangle$ got an error then that would result in the 4th qubit tranforming to the $|1\rangle$ state because $|\psi\rangle \oplus X|\psi\rangle \oplus |0\rangle = |1\rangle$. Similarly the 5th qubit would also tranform to $|1\rangle$ state singnalling to us that the error is in the second qubit (because if it were in the first qubit then the fifth qubit would have remained as $|0\rangle$).

2.3. Stabiliser Formalism

The stabilizer formalism provides a powerful mathematical framework for describing and constructing quantum error correcting codes. It builds upon the intuition gained from the classical repetition code example discussed earlier, but introduces a more general and rigorous approach.

2.3.1. Pauli Group and Stabilizers

At the core of the stabilizer formalism is the Pauli group, which is the group generated by the single-qubit Pauli operators I, X, Y, Z. The n-qubit Pauli group $\mathcal{P}^n$ is the n-fold tensor product of the single-qubit Pauli group, and its elements are of the form:

$$\mathcal{P}^n = \pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ^{\otimes n}$$

The Pauli group is an Abelian group under the operation of tensor product, meaning that any two Pauli operators commute or anti-commute. This property is crucial for the stabilizer formalism. A stabilizer $\mathcal{S}$ of a quantum code is defined as an Abelian subgroup of the Pauli group $\mathcal{P}^n$ which fixes some subspace of the set of n qubits, V_S. Every element of the space V_S is invariant under the action of the elements of $\mathcal{S}$. The stabilizer $\mathcal{S}$ is generated by a set of $n-k$ independent Pauli operators, where k is the number of logical qubits encoded in the n physical qubits.

For example, consider the 3-qubit bit-flip code discussed earlier, where we are trying to encode 1 logical qubit into 3 physical qubits. The stabilizer for this code for the vector space $V_S = \text{span}(|000\rangle, |111\rangle)$ is given by $\mathcal{S} = \{I_1I_2I_3, Z_1Z_2, Z_2Z_3, Z_1Z_3\}$. Since any element of V_S on applying an element of $\mathcal{S}$ remains unchanged. Incidentally, the set $\mathcal{S}$ is generated by the two operators:

$$S_1 = Z_1Z_2, \quad S_2 = Z_2Z_3$$

where the subscripts indicate the qubit on which each Pauli operator acts. The encoded logical states $|0\rangle_L$ and $|1\rangle_L$ are the $+1$ eigenstates of all the stabilizer operators in $\mathcal{S}$. Any error that anti-commutes with one of the stabilizers will change the eigenvalue of that stabilizer, allowing us to detect the error.

It is advantageous to use the generators of a stabiliser group to describe the group because of the smaller size of the generators group. Furthermore, generators of any group satisfy the property that any and all elements of the group can be expressed as a product of the generators of the group. This allows us to only check the stabilization of an arbitrary vector for the generators, since if it is stabilised by the generators, it will be stabilised by every other element of the group also. More generally, for an $[[n, k]]$ quantum error correcting code that encodes k logical qubits into n, the stabilizer $\mathcal{S}$ is an Abelian subgroup of the Pauli group $\mathcal{P}^n$ with $n - k$ independent generators.

2.3.2. Shor Code

The bit flip code is interesting but it leaves many other questions unanswered. What if the error is of some other nature like a phase flip? The phase flip error can be modelled as applying a Z gate to a qubit with some probability p, i.e. the state $|\psi\rangle = a|0\rangle + b|1\rangle$ is taken to the state $a|0\rangle - b|1\rangle$ with probability p. Our bit flip code no longer works here because it doesn't detect any changes in the phase. However, a simple change can make it work for a phase flip channel: using the states $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ as the basis in the circuit. This is due to the fact that applying the Z error on any of these states transforms it to the other, effectively acting as the same way that X was in the bit flip code! The Stabilisers must also be changed to the new basis, $S_{Phase} = HSH = \{I_1I_2I_3, X_1X_2, X_2X_3, X_1X_3\}$.

After seeing the phase flip code, the natural question that arises is "Does there exist a code that can correct both bit flip and phase flip errors at the same time?" The answer to this question is the Shor code which incidentally is sufficient to correct the error of any arbitrary single qubit error!

The code itself is quite simple, just encode the state into the phase flip code and then encode each of those qubits into the bit flip code. $|0\rangle_L = | +_l +_l +_l\rangle$
$= \frac{|000\rangle+|111\rangle}{\sqrt{2}} \frac{|000\rangle+|111\rangle}{\sqrt{2}} \frac{|000\rangle+|111\rangle}{\sqrt{2}}$ and similarly for $|1\rangle$. The error detection part is also quite simple: detect and correct bit flip errors in the encoded qubits in groups of 3. Finally perform phase flip detection and correction on the resultant qubits. For example, suppose an error of Z_4 was applied on the encoded qubit. Then the first step gives us no error detected, since bit flip code can't detect phase flip errors. Then on the second step, we will apply the stabilisers $X_1X_2X_3X_4X_5X_6$ and $X_4X_5X_6X_7X_8X_9$ on the qubits and get the measurement output 11, which denotes an error in one of the 3 middle qubits. This phase error can be fixed by simply applying the operator $Z_4Z_5Z_6$.

The generator of stabilisers for Shor's code is

$$G = \{Z_1Z_2, Z_2Z_3, Z_4Z_5, Z_5Z_6, Z_7Z_8, Z_8Z_9, X_1X_2X_3X_4X_5X_6, X_4X_5X_6X_7X_8X_9\}$$

This is consistent with the above stated fact that a $[[9, 1]]$ code that encodes 1 qubit into 9 qubits will have a minimum of 9-1 = 8 stabilizer generators.

Logical Operators

With 9 physical qubits and 8 stabilizer generators, there is one remaining degree of freedom, which can take two possible eigenvalues: +1 and -1. These correspond to the logical states $|0\rangle$ and $|1\rangle$ of the encoded qubit. Consequently, we can define transformations between these logical states. Such transformations take encoded states to other encoded states, and they are called logical operators.

For the Shor code, we define a logical Z operator Z_L and a logical X operator X_L, which act as the Pauli Z and X operators on the encoded states. For example, $Z_L|0\rangle_L =$

$|0\rangle_L, Z_L|1\rangle_L = -|1\rangle_L$, and similarly for X_L, which transforms $|0\rangle_L$ to $|1\rangle_L$ and vice versa. Logical operators commute with the stabilizer generators, preserving the encoded space.

One possible expression for the expansion of the logical operators is $Z_L = X_1X_2X_3$ and $X_L = Z_1Z_4Z_7$.

The stabilizer formalism can be generalized to construct more advanced quantum error correcting codes, such as the Steane code, the surface code, and the color code. These codes have better error-correcting capabilities and are essential for fault-tolerant quantum computation.

2.4. Quantum error correction with the Stabilizer code

Now that we have seen how Stabilisers can be used to correct errors in a single qubit, we shall talk about how general QEC works with stabiliser codes.

Recall that in a QEC, we encode a state $|\psi\rangle$ as $|\psi_L\rangle$, which is an entangled state of many qubits. Errors affect the individual qubits in $|\psi_L\rangle$, so in general we will get some state $E|\psi_L\rangle$. In any QEC scheme, we start by detecting if an error has occured. Thanks to the stabilizer formalism, we can easily determine if a state is part of the code by measuring the stabilizer generators and checking in each case if the outcome is equal to +1. If the outcome of measuring any stabilizer generator is -1, this indicates that some error has occurred. In particular, if the error E anti commutes with a stabilizer generator S_i, it is easy to see that $S_iE|\psi_L\rangle = -ES_i|\psi_L\rangle$, indicating a -1 outcome on measuring the stabilizer generator S_i. Thus, measuring all the stabiliser generators yields a vector of length $n-k$ called the error syndrome, denoted by $s = s_1s_2s_3\dots s_{n-k}$, where,

$$s_i = \begin{cases} 0 & \text{if } [E, S_i] = 0 \\ 1 & \text{if } \{E, S_i\} = 0 \end{cases} \tag{2.1}$$

After measuring the error syndrome, we know all the eigenvalues of the state $E|\psi\rangle_L$ with respect to the Stabilizer generators. So, the state after measurement is simply

$$\left(\frac{\mathbb{I} + (-1)^{s_1}S_1}{2}\right)\left(\frac{\mathbb{I} + (-1)^{s_2}S_2}{2}\right)\dots\left(\frac{\mathbb{I} + (-1)^{s_{n-k}}S_{n-k}}{2}\right)E|\psi_L\rangle \tag{2.2}$$

The probability of measuring the error syndrome s, denoted by $P(s)$ is the trace of the above state.

After detecting the error, the next step in a QEC is to correct it. The point to note here is that many errors yield a given error syndrome, s. In fact, there is an exponential amount of them. The decoder thus has to guess the most likely error pattern E from the exponential possibilities by looking at the syndrome. Finally, we can say whether a QEC is successful whenever on applying the reverse error $\hat{E}$ on $E|\psi\rangle$ we get back the original state and it fails otherwise. In other words, we would like $\hat{E} \cdot E$ to be a Stabilizer of the underlying code.

Now, ideally we would want the success probability of any QEC to be maximum, so the decoding algorithm needs to choose the best guess for E that it possibly can. However, due to the space of possible errors being exponential, it is infeasible to directly compute all probabilities and taking the maximum. Therefore, as a workaround we instead estimate the failure probability of QEC for a given syndrome numerically, typically via Monte Carlo simulations. Again, computing, and hence sampling from the distribution of all probabilities involves computing the trace of an exponentially sized matrix which is numerically intensive. We want to leverage the tools of Tensor Networks to address this problem in the coming sections.

Chapter 3

Tensor Networks

Tensor Networks have emerged as a revolutionary framework in quantum many-body physics and quantum information theory, addressing one of the field's most fundamental challenges: exponential dimensionality. The exponential growth of the Hilbert space of quantum states poses a significant barrier to both theoretical analysis and numerical computations. Tensor networks provide an elegant solution by decomposing complex high-dimensional tensors into networks of smaller tensors, while preserving the essential physical properties and correlations of the system under study [10].

3.1. Fundamental Components of Tensor Networks

At their core, tensors are mathematical objects that generalize the concepts of vectors and matrices to higher dimensions. While vectors are rank-1 tensors existing in a d-dimensional space $\mathbb{C}^d$, and matrices are rank-2 tensors in $\mathbb{C}^{n\times m}$, a general rank-r tensor inhabits a much larger space defined by $\mathbb{C}^{d_1\times\cdots\times d_r}$. This hierarchical structure allows tensors to naturally represent increasingly complex quantum systems and their interactions.

The graphical notation used in tensor networks provides an intuitive and powerful visual language for representing these complex mathematical objects. In this notation, tensors are depicted as geometric shapes or nodes, with protruding lines or "legs" representing their indices. For example, a rank-4 tensor $R_{\rho\sigma\mu\nu}$ would be represented as a node with four external legs. This visual representation makes it remarkably easy to understand and manipulate complex tensor operations that would be cumbersome to express in traditional mathematical notation.

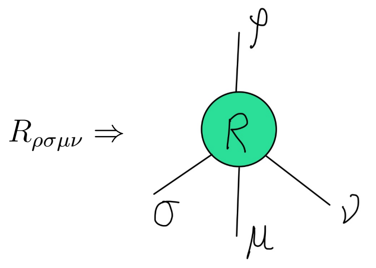


Figure 3.1: Representation of a rank-4 tensor with indices ρ, σ, μ, ν

3.2. Fundamental Operations in Tensor Networks

Indices In tensor network notation, indices represent the dimensions or degrees of freedom a tensor can possess. Each index corresponds to a specific basis in the vector space, and the number of indices (also called the rank) determines the complexity of the tensor. For instance, while a vector has a single index ranging over its components, a matrix has two indices corresponding to its rows and columns. For instance, in 3.1, the indices of R are denoted by ρ, σ, μ, ν and their dimensions, or the range of values each of the indices can take, represent the bond dimensions of the indices.

The nature and treatment of indices in tensor networks follows conventions that enhance both clarity and computational efficiency. Indices can be broadly categorized into two types: external (or free) indices that remain uncontracted in the final result, and internal (or dummy) indices that are summed over during contractions. The dimensionality of each index, often denoted as d_i for the i-th index, determines the size of the corresponding vector space. For example, in quantum computing applications, many indices are of dimension 2, corresponding to qubit states and only take the values 0 or 1.

Tensor Product The tensor product, denoted by $\otimes$, is a fundamental operation that generalizes the outer product of vectors. For two tensors A and B, their tensor product is defined mathematically as:

$$[A \otimes B]_{i_1,\ldots,i_r,j_1,\ldots,j_s} := A_{i_1,\ldots,i_r} \cdot B_{j_1,\ldots,j_s}$$

For example, the tensor product of the operators $X_1 \otimes Z_2$ is defined to be the following matrix:

$$X_1 \otimes Z_2 = X \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} X \times 1 & X \times 0 \\ X \times 0 & X \times -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

This operation creates a larger tensor whose components are all possible products of the components of the original tensors. In the graphical notation, tensor products are represented simply by placing the tensor nodes side by side. For example, consider two vectors $\vec{x}$ and $\vec{y}$. Their tensor product creates a matrix whose elements are given by $(x \otimes y)_{ij} = x_i y_j$. This operation is crucial in quantum mechanics for describing composite systems, where the state space of the combined system is the tensor product of the individual state spaces. For example, when discussing a 3 qubit system the qubits were implicitly under a tensor product with each other $|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle$

Trace Operation The trace operation in tensor networks generalizes the familiar matrix trace to higher-rank tensors. For a tensor A with compatible indices x and y (meaning they have the same dimension), the partial trace over these indices is defined as:

$$[Tr_{x,y}A]_{i_1,\ldots,i_{x-1},i_{x+1},\ldots,i_{y-1},i_{y+1},\ldots,i_r} = \sum_{\alpha=1}^{d_x} A_{i_1,\ldots,i_{x-1},\alpha,i_{x+1},\ldots,i_{y-1},\alpha,i_{y+1},\ldots,i_r}$$

In the graphical notation, this operation is represented by connecting the corresponding legs of the tensor, forming a loop. This visual representation makes it particularly easy to understand properties like the cyclic property of the trace. For example, the

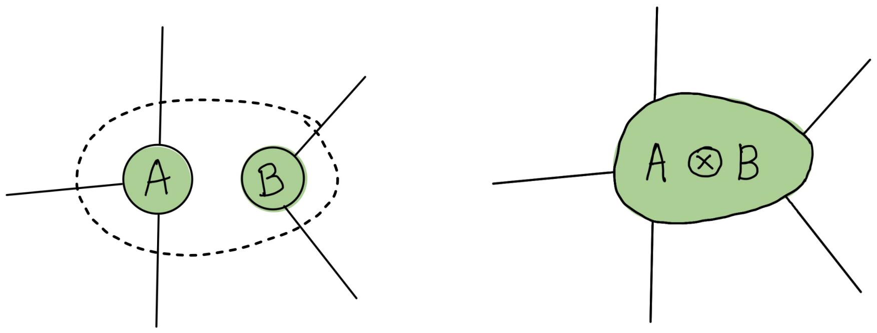
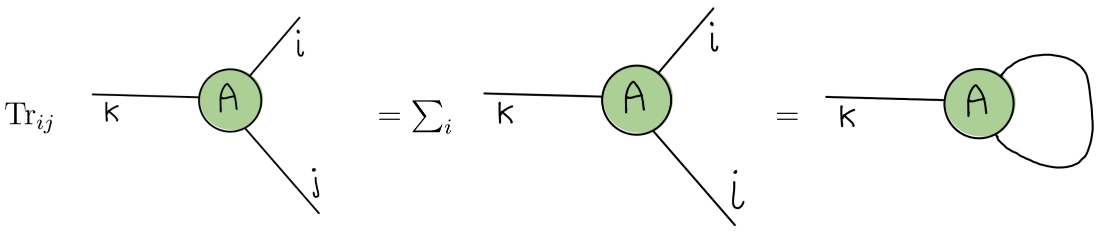


Figure 3.2: Multiplying two tensors

well-known property $Tr(AB) = Tr(BA)$ becomes immediately obvious when represented in the tensor network notation, as it simply involves sliding one tensor around a loop to a different position.



Contraction Tensor contraction is perhaps the most important operation in tensor networks, combining aspects of both tensor products and traces. It involves taking a tensor product followed by a trace over specified indices. In practice, this operation is equivalent to a summation over shared indices between tensors. For example, the contraction of two rank-3 tensors over two pairs of indices would be written as:

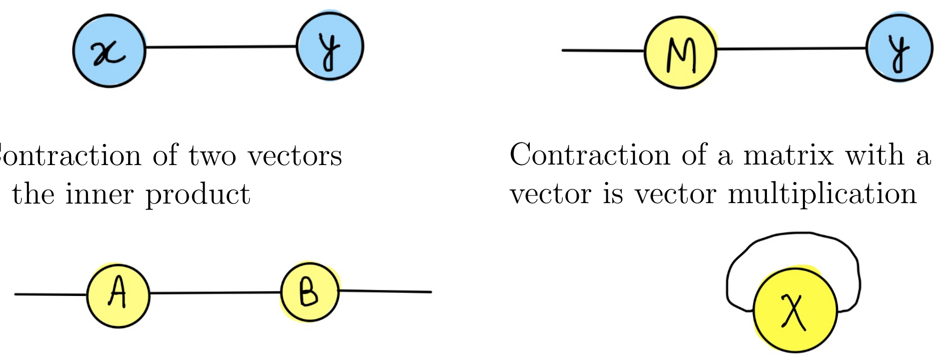
$$C_{ij} = \sum_{\alpha,\beta} A_{i\alpha\beta} B_{\alpha\beta j}$$

This operation appears frequently in physics and mathematics, with matrix multiplication being a simple example of tensor contraction. In the graphical notation, contraction is represented by connecting the corresponding legs between tensors, making the operation visually intuitive and easy to track.

3.3. Matrix Product States

Matrix Product States (MPS) represent a particularly important class of tensor networks, especially suited for describing one-dimensional quantum systems. An MPS decomposes a quantum state $|\psi\rangle$ of N qudits into a product of matrices, with each matrix corresponding to a local site. For a completely general state of N qudits: $|\psi\rangle = \sum_{j_1 j_2 \ldots j_N = 0}^{d-1} C_{j_1 j_2 \ldots j_N} |j_1\rangle \otimes |j_2\rangle \otimes \cdots \otimes |j_N\rangle$ This can be expressed in MPS form as:

$$|\psi[A]\rangle = \sum_{i_1 i_2 \ldots i_N} \mathrm{Tr}[A^{(1)} i_1 A^{(2)} i_2 \ldots A^{(N)}_{i_N}] |i_1 i_2 \ldots i_N\rangle$$



Contraction of two vectors is the inner product

Contraction of a matrix with a vector is vector multiplication

Contraction of two matrices is matrix multiplication

Contraction of a matrix with itself is the trace

Figure 3.7: Various types of contractions of tensors

where each $A_{i_n}^{(n)}$ is a matrix associated with site n and physical index i_n. The power of MPS lies in their ability to efficiently represent states with limited entanglement, particularly those satisfying an area law for entanglement entropy. For states with bounded entanglement rank D across any bipartition, MPS provide an exact representation using only $\mathcal{O}(dND^2)$ parameters, compared to the exponential d^N parameters needed for the full state vector.

In the graphical notation, an MPS is represented as a chain of tensors, where each tensor has three indices: one physical index (pointing vertically) and two virtual indices (pointing horizontally) that connect to neighboring sites. The trace operation in the MPS expression corresponds to connecting the virtual indices in a chain-like structure.

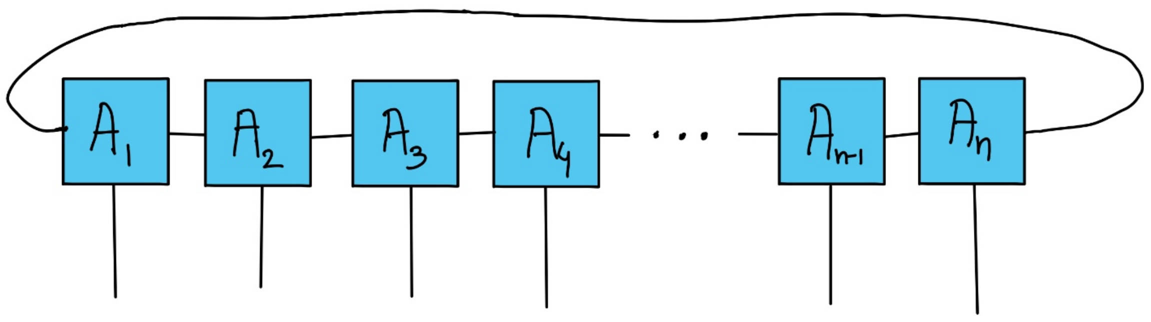


Figure 3.8: A typical Matrix Product State with n tensors

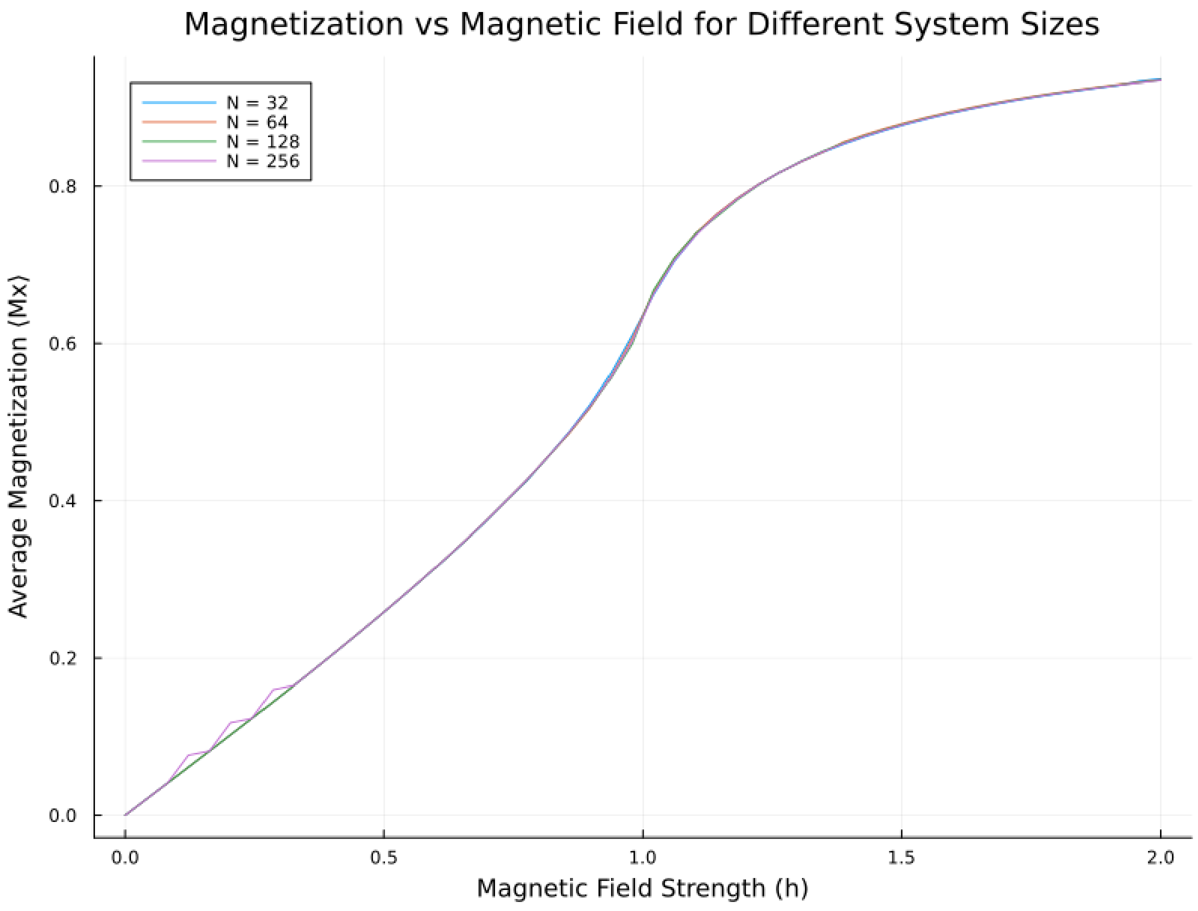
3.4. 1D Transverse-field Ising Model

With the foundational understanding of tensor networks and matrix product states (MPS), we can now explore a practical application: the simulation of the 1D transverse-field Ising model. This model is a quintessential example of a quantum spin chain, where a series of particles (spins) interact with their nearest neighbors while also experiencing a transverse magnetic field. The Hamiltonian of the transverse-field Ising model is given by: $H = -J \left(\sum_{\langle i,j \rangle} Z_i Z_j + g \sum_j X_j \right)$ Here, the $Z_j Z_i$ term represents the nearest-neighbor interactions along the chain, while the X_j term captures the transverse field's influence on individual spins.

This system exhibits a quantum phase transition at a critical value of the external field h_c, where the magnetization behavior changes dramatically. Specifically, the average magnetization along the xx-axis $\langle X \rangle$ remains low for small values of the transverse field h, but increases sharply beyond the critical threshold. This behavior reflects the transition from a magnetically ordered phase to a paramagnetic phase dominated by the external field.

By leveraging the MPS formalism, the 1D transverse-field Ising model can be simulated efficiently. MPS is particularly well-suited for such 1D systems due to the limited entanglement between spins, enabling the representation of the wavefunction with a polynomial number of parameters $\mathcal{O}(dND2)$. This efficiency is a stark contrast to the exponential scaling d^N required for a full state-vector representation.

In our simulation, we computed the system's magnetization as a function of the external magnetic field. The plot in the below figure depicts the magnetization $\langle X \rangle$ for spin chains of varying lengths, showcasing a clear critical point at $g = 1.0$. Above this value, the magnetization increases rapidly, indicating a transition to the paramagnetic phase. Notably, the results match the exact analytical solution of the transverse-field Ising model, validating the accuracy and utility of tensor network methods [11] [12]. For the implementation code, refer to the Appendix.



3.5. Quantum Error Correction (QEC) with Tensor Networks

Tensor networks provide a natural and powerful framework for representing and analyzing quantum error correction codes. Their ability to efficiently capture quantum correlations and entanglement structures makes them particularly well-suited for this application. The framework allows for explicit representation of quantum states, error syndromes, and correction operations, while maintaining computational tractability for many practical cases.

3.5.1. State Representation and Error Correction

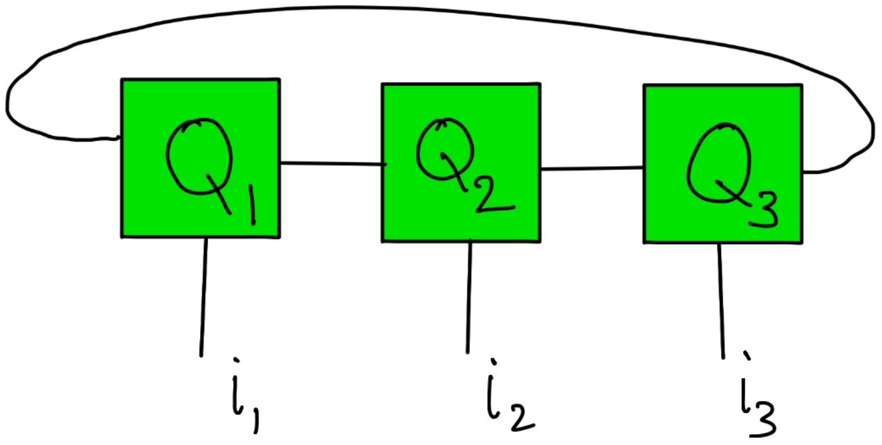
In the context of quantum error correction, tensor networks can represent both the protected quantum states and the error correction procedures. The network structure naturally reflects the entanglement patterns present in error correction codes, with stabilizer codes being particularly amenable to efficient representation. The process of error detection through syndrome measurements can be implemented as specific tensor contractions, while error propagation can be tracked through the evolution of the network structure.

The advantage of using tensor networks in QEC lies in their ability to handle the complex quantum correlations present in error correction codes while maintaining computational efficiency. The graphical nature of tensor networks also provides valuable intuition about the structure of quantum codes and their behavior under various types of errors.

3.5.2. Stabiliser code as a Tensor Network

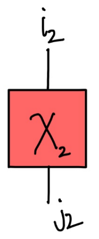
In this section, we will apply the tensor network formalism to describe error correction codes, specifically focusing on the bit flip code. Recall that implementing any error correcting code requires defining four key operations: encoding the state, detecting errors, correcting those errors, and decoding the state. For the bit flip code, encoding and decoding involve replicating the qubit three times and then collapsing three qubits back into one, respectively. Here, we will focus on the error detection and correction procedures using tensor network terminology.

Setup We shall model the encoded state $|\psi\rangle_L$ as an MPS of 3 qubits in their density matrix forms.

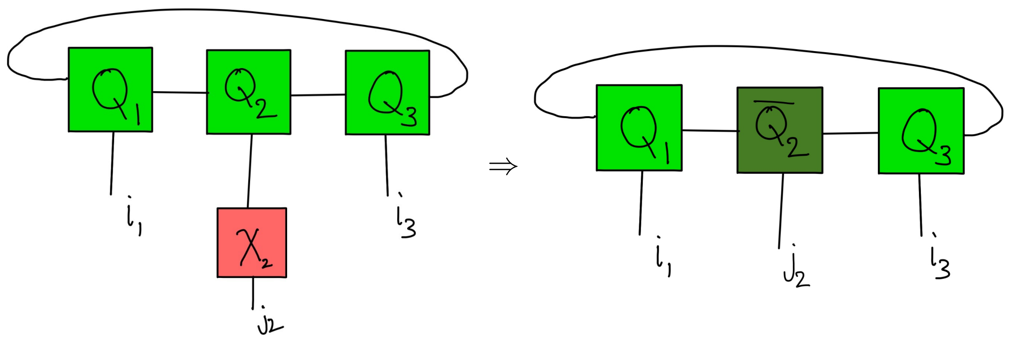


For example, on encoding the state $|0\rangle_L$ into $|000\rangle$ we have $Q_1Q_2Q_3 = |0\rangle \otimes |0\rangle \otimes |0\rangle$. Each tensor has dimensions $(1, 2, 1)$ where the left bond dimension is 1, physical dimension is 2 (representing the states $|0\rangle$ and $|1\rangle$) and the right bond dimension is 1.

Now, the bit flip error in the channel can again be modeled as a tensor with two indices which applies the X gate.



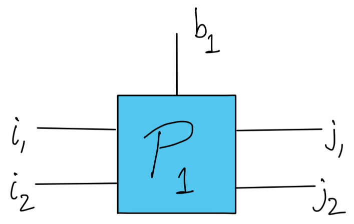
Contracting this with the second qubit of our MPS, we get the following state:



Error Detection To detect the error in the given state, we can make use of the stabilisers for the bit flip channel, Z_1Z_2 and Z_2Z_3. To get the syndromes, we shall contract the current state with the tensors

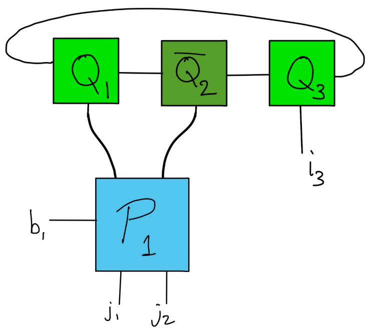
$$\mathcal{P}_1^{b_1} = \frac{\mathbb{I} + (-1)^{b_1} Z_1 Z_2}{2} \text{ and } \mathcal{P}_2^{b_2} = \frac{\mathbb{I} + (-1)^{b_2} Z_2 Z_3}{2}$$

The inclusion of the identity and the term $(-1)^{b_i}$ allows us to transform the eigenvalues from 1 and -1 to 1 and 0 respectively. Now, each of the above tensors can be expressed with five indices i_1, i_2, j_1, j_2, b with respective dimensions (2,2,2,2,2). Here, i_1, i_2 denote the row indices and j_1, j_2 denote the column indices of the tensor. The index b denotes the value of the parameter b_i.



To contract this with our state, we need to rename the indices of the state back to i_1, i_2, i_3. Then contracting $\mathcal{P}_1^{b_1}$ along the common indices i_1 and i_2 we get,

From this tensor, the syndrome can be obtained by taking the trace of its density matrix. Note that while taking the trace, we have to take b_1 as a fixed value that is either



0 or 1. If the trace comes out to be 0 then the syndrome is $1-b_1$, otherwise the syndrome is b_1.

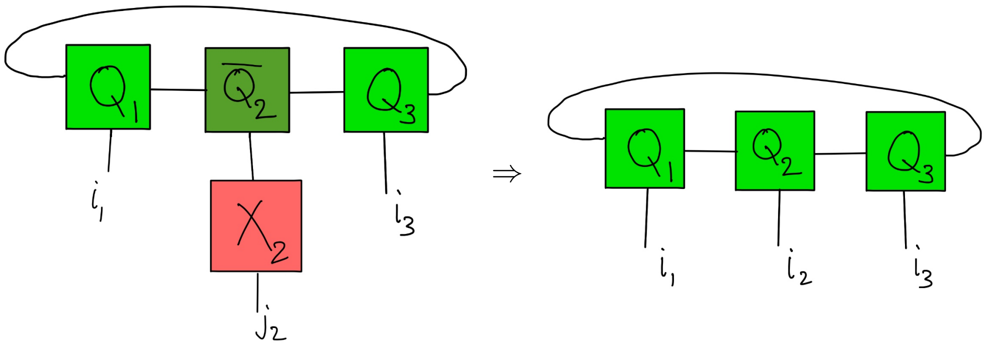
We can perform a similar contraction of $\mathcal{P}_2^{b_2}$ on the second and third qubits and get the second syndrome.

Correction After we know the values of the two syndromes, we can deduce the position of the error in the state.

Syndromes	**Correction**
00	$I_1I_2I_3$
01	X_3
10	X_1
11	X_2

Table 3.1: Syndromes and their corresponding corrections

In our example, the syndromes are 11 and thus we shall apply the X_2 gate and rename the indices to recover the original state.



3.6. Steane Code Simulation Using Tensor Networks

Having demonstrated the use of tensor networks for the bit flip code, it is worth noting that this framework extends naturally to more complex quantum error correction codes, such as the Steane code. The Steane code, a [[7,1,3]] stabilizer code, encodes a single logical qubit into seven physical qubits, protecting against both bit-flip and phase-flip errors. The simulation begins by initializing a quantum state, followed by encoding it using a sequence of Hadamard and controlled-X (CX) gates. To model realistic quantum noise, amplitude damping errors are applied to each qubit, simulating the effects of environmental interactions. The error detection process leverages the stabilizers of the Steane code, projecting the corrupted state onto subspaces defined by specific syndromes. The probabilities of different error syndromes are computed, reflecting the likelihood of various error patterns. The plot in the below figure illustrates the syndromes obtained during the simulation of the Steane code. This simulation highlights the efficiency of tensor network methods in representing complex quantum states and performing error correction tasks, providing a robust framework for studying the performance of quantum codes under noise. For the implementation code, refer to the Appendix.

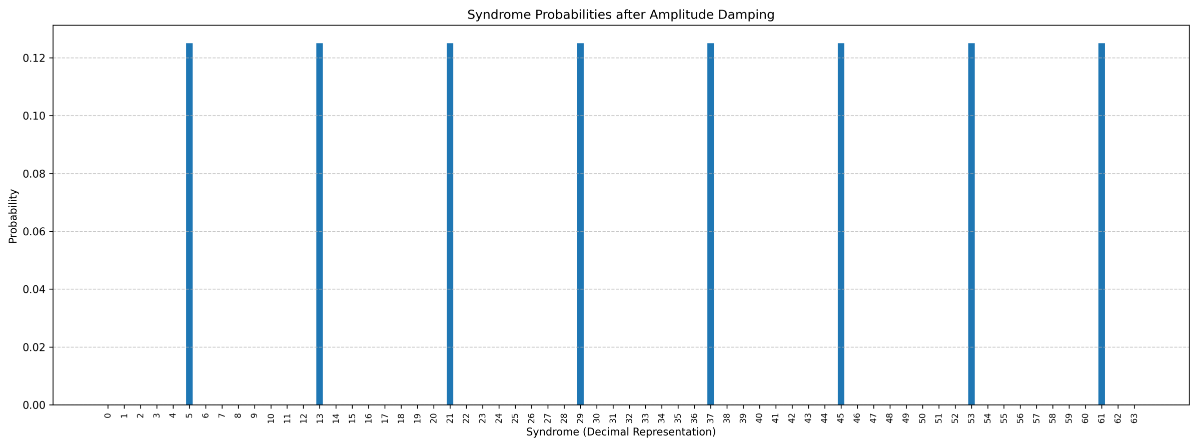


Figure 3.9: Steane code syndrome probabilities under realistic noise simulation

3.7. Conclusion

Tensor Networks offer a highly intuitive and efficient framework for representing quantum error correction codes. While the example of the bit flip code is small in scope and scale, tensor networks reveal their true power when applied to larger systems with many qubits, as seen by the simulation on the Steane code. The surface code is a generalisation of linear codes for higher dimensions. Implementation of surface codes in the language of Tensor Networks can provide more insights on the properties of TNs and how they cleverly use the localisation of error propagation to maintain computational efficiency, even for large many-body systems [13].

Future work in this area could involve exploring tensor network implementations for advanced quantum error correction codes beyond stabilizer codes, including color codes and topological codes, which offer additional robustness against certain types of quantum noise [14]. Additionally, leveraging tensor networks for hybrid models—where classical and quantum error correction techniques intersect—may further enhance error resilience in quantum systems. As quantum computing continues to evolve, tensor networks may play a pivotal role in achieving scalable and practical error correction solutions, paving the way for more robust quantum technologies.

Bibliography

[1] Austin G Fowler et al. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86.3 (2012), p. 032324.

[2] Daniel Gottesman. “Stabilizer codes and quantum error correction”. In: *arXiv preprint arXiv:quant-ph/9705052* (1997).

[3] Rami Barends et al. “Superconducting quantum circuits at the surface code threshold for fault tolerance”. In: *Nature* 508.7497 (2014), pp. 500–503.

[4] Bao Yan et al. *Factoring integers with sublinear resources on a superconducting quantum processor*. 2022. arXiv: 2212.12372 [quant-ph]. URL: https://arxiv.org/abs/2212.12372.

[5] Julian Kelly et al. “State preservation by repetitive error detection in a superconducting quantum circuit”. In: *Nature* 519.7541 (2015), pp. 66–69. DOI: 10.1038/nature14270.

[6] Andrew J Ferris and David Poulin. “Tensor networks and quantum error correction”. In: *Physical Review Letters* 113.3 (2014), p. 030501.

[7] Román Orús. “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of Physics* 349 (2014), pp. 117–158.

[8] Fernando Pastawski et al. “Holographic quantum error-correcting codes: Toy models for the bulk/boundary correspondence”. In: *Journal of High Energy Physics* 2015.6 (2015), p. 149.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[10] Jacob C Bridgeman and Christopher T Chubb. “Hand-waving and interpretive dance: an introductory course on tensor networks”. In: *Journal of Physics A: Mathematical and Theoretical* 50.22 (May 2017), p. 223001. ISSN: 1751-8121. DOI: 10.1088/1751-8121/aa6dc3. URL: http://dx.doi.org/10.1088/1751-8121/aa6dc3.

[11] Sei Suzuki, Jun-ichi Inoue, and Bikas K. Chakrabarti. “Quantum Ising Phases and Transitions in Transverse Ising Models”. In: 1996. URL: https://api.semanticscholar.org/CorpusID:117986766.

[12] Hendrik Weimer. “Lecture Notes on Quantum Simulation”. In: (Apr. 2017). DOI: 10.48550/arXiv.1704.07260.

[13] Andrew S. Darmawan and David Poulin. “Tensor-Network Simulations of the Surface Code under Realistic Noise”. In: *Phys. Rev. Lett.* 119 (4 July 2017), p. 040502. DOI: 10.1103/PhysRevLett.119.040502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.119.040502.

[14] Hidetaka Manabe, Yasunari Suzuki, and Andrew S. Darmawan. *Efficient Simulation of Leakage Errors in Quantum Error Correcting Codes Using Tensor Network Methods*. 2023. arXiv: 2308.08186 [quant-ph]. URL: https://arxiv.org/abs/2308.08186.

Appendix

Magnetization vs Magnetic field

Here is the Julia code for simulating the spin chains and numerically calculate the Magnetization vs Magnetic Field for different system sizes.

```
using ITensors
using Plots

function create_tfim_hamiltonian(N::Int, h::Float64)
    # Create array of sites
    sites = siteinds("S=1/2", N)

    # Initialize Hamiltonian MPO
    ampo = OpSum()

    # Add ZZ interaction terms with periodic boundary condition
    for j in 1:N
        ampo += -1.0, "Z", j, "Z", mod1(j+1, N)
    end

    # Add transverse field terms
    for j in 1:N
        ampo += -h, "X", j
    end

    # Convert to MPO
    H = MPO(ampo, sites)

    return H, sites
end

function get_ground_state(H, sites)
    # Random initial state
    psi0 = randomMPS(sites)

    # DMRG sweeps parameters
    nsweeps = 20  # Increased number of sweeps for larger system
    maxdim = [10,20,100,200,300,400,500]
    cutoff = 1E-12  # Tighter cutoff
```

```julia
    # Find ground state
    energy, psi = dmrg(H, psi0; nsweeps, maxdim, cutoff)

    return psi
end

function compute_magnetization(psi, sites)
    # Compute total magnetization in x direction
    mx = 0.0
    for i in 1:length(sites)
        # Measure local x magnetization for each site
        ops = OpSum()
        ops += 1.0, "X", i
        op_mpo = MPO(ops, sites)
        local_mx = real(inner(psi', op_mpo, psi))
        mx += local_mx
    end

    # Return average magnetization
    return mx / length(sites)
end

# Compute magnetization as a function of h
function magnetization_vs_h(N::Int, h_values::StepRangeLen{Float64})
    magnetizations = Float64[]

    for h in h_values
        # Create Hamiltonian and find ground state
        H, sites = create_tfim_hamiltonian(N, h)
        psi = get_ground_state(H, sites)

        # Compute magnetization
        mx = compute_magnetization(psi, sites)
        push!(magnetizations, abs(mx))
    end

    return magnetizations
end

# Parameters
h_values = range(0.0, 2.0, length=50)  # Range of magnetic field strengths
system_sizes = [32, 64, 128, 256]  # Different system sizes to explore

# Create plot
p = plot(xlabel="Magnetic Field Strength (h)",
         ylabel="Average Magnetization ⟨Mx⟩",
         title="Magnetization vs Magnetic Field for Different System Sizes",
         legend=:topleft,
         linewidth=2,
         size=(800, 600))

# Compute and plot magnetization for each system size
for N in system_sizes
    magnetizations = magnetization_vs_h(N, h_values)
    plot!(p, h_values, magnetizations, label="N = $N")
end

# Save the plot
savefig(p, "tfim_magnetization_multiple_sizes.png")

# Print the results for verification
println("System Sizes: ", system_sizes)
```

Simulation of Steane Code

Here is the Julia code for encoding a given quantum state into the steane code, then applying an amplitude damping channel on each of the qubits, followed by the sampling of the probabilities of each of the syndromes.

```
using ITensors
using Plots

# Define the site indices for 7 qubits
N = 7
s = siteinds("S=1/2", N)

# Existing operator and initialization functions from the previous implementation
# (Copying over the functions from the original code)
function ITensors.op(::OpName"Z", ::SiteType"S=1/2", s::Index)
    mat = [1 0
           0 -1]
    return ITensor(mat, s', s)
end

function ITensors.op(::OpName"X", ::SiteType"S=1/2", s::Index)
    mat = [0 1
           1 0]
    return ITensor(mat, s', s)
end

function ITensors.op(::OpName"H", ::SiteType"S=1/2", s::Index)
    mat = [1 1
           1 -1]/sqrt(2)
    return ITensor(mat, s', s)
```

```
end

function initialize_state(::Number, ::Number)
     = MPS(s)

    # Set first qubit to |0⟩ + |1⟩
    [1] = ITensor([, ], s[1])

    # Set other qubits to |0⟩
    for j in 2:N
        [j] = ITensor([1.0, 0.0], s[j])
    end

    return normalize!()
end

function encode_steane(::MPS)
    encoding_ops = [
        ("H", 5), ("H", 6), ("H", 7),  # Create |+⟩ states
        ("CX", 1, 2), ("CX", 1, 3), # first round
        ("CX", 7, 1), ("CX", 7, 2), ("CX", 7, 4), # second round
        ("CX", 6, 1), ("CX", 6, 3), ("CX", 6, 4), # third round
        ("CX", 5, 2), ("CX", 5, 3), ("CX", 5, 4) # fourth round
    ]
    _encoded = apply(ops(encoding_ops, s), )
    return normalize!(_encoded)
end

# Updated compute_syndrome_probabilities function
function compute_syndrome_probabilities(::MPS)
    probs = Dict{Tuple{Int,Int,Int,Int,Int,Int}, Float64}()

    # Define the stabilizer operators and their locations
    stabilizers = [
        ("X", [4,5,6,7]),  # X4567
        ("X", [2,3,6,7]),  # X2367
        ("X", [1,3,5,7]),  # X1357
        ("Z", [4,5,6,7]),  # Z4567
        ("Z", [2,3,6,7]),  # Z2367
        ("Z", [1,3,5,7])   # Z1357
    ]

    for j in 1:length()
        [j] = noprime([j])
    end

    # Iterate through all possible syndrome patterns (64 combinations)
    for s1 in 0:1, s2 in 0:1, s3 in 0:1, s4 in 0:1, s5 in 0:1, s6 in 0:1
```

```
        syndromes = [s1, s2, s3, s4, s5, s6]

        # Make a copy of the state to work with
        _projected = copy()

        # Apply each stabilizer projector
        for (idx, (op_type, qubits)) in enumerate(stabilizers)
            sign = (-1)^syndromes[idx]
            # Store original state before applying stabilizer
            _orig = copy(_projected)

            # Apply operator to each qubit in the stabilizer
            for q in qubits
                _projected = orthogonalize!(_projected, q)
                _projected[q] = op(op_type, s[q]) * _projected[q]
            end

            # Unprime all tensors before addition
            for j in 1:length(_projected)
                _projected[j] = noprime(_projected[j])
            end
            for j in 1:length(_orig)
                _orig[j] = noprime(_orig[j])
            end

            # Form the projector (I ± S)/2 where S is the stabilizer
            _projected = (_orig + sign * _projected)/2.0
        end

        # Compute probability
        prob = abs2(inner(', _projected))
        if prob > 1e-10  # Only store non-negligible probabilities
            probs[(s1,s2,s3,s4,s5,s6)] = prob
        end
    end

    # Normalize probabilities
    total = sum(values(probs))
    for k in keys(probs)
        probs[k] /= total
    end

    return probs
end

# Apply amplitude damping error
function apply_amplitude_damping(::MPS, site::Int, ::Float64)
     = orthogonalize!(, site)
    T = [site]
    si = siteind(, site)

    K0 = ITensor([1.0 0.0; 0.0 sqrt(1-)], si', si)
    K1 = ITensor([0.0 sqrt(); 0.0 0.0], si', si)

    0 = copy()
    0[site] = K0 * T
    normalize!(0)

    1 = copy()
    1[site] = K1 * T
    normalize!(1)

    p0 = real(inner(0', 0))
    p1 = real(inner(1', 1))

    total_prob = p0 + p1
    p0 /= total_prob
    p1 /= total_prob

    if rand() < p0
        return normalize!(0)
    else
        return normalize!(1)
    end
end

# Comprehensive test function
function test_steane_code_full_analysis(::Number, ::Number, ::Float64)
    # Initialize and encode
    _initial = initialize_state(, )
    println("Initial state prepared")

    _encoded = encode_steane(_initial)
    println("State encoded using Steane code")

    # Apply amplitude damping to all qubits
    _with_error = copy(_encoded)
    for error_site in 1:7
        _with_error = apply_amplitude_damping(_with_error, error_site, )
    end
    println("\nApplied amplitude damping error with  = $ on all qubits")

    # Compute syndrome probabilities
    probs = compute_syndrome_probabilities(_with_error)

    return _with_error, probs
```

```
end

# Run test with parameters , ,
_final, syndrome_probs = test_steane_code_full_analysis(0.7, 0.7, 0.6)

# Optional: Display non-negligible syndrome probabilities
for (syndrome, prob) in sort(collect(syndrome_probs), by=x->x[2], rev=true)
    if prob > 0.001
        println("P$(syndrome) = $(round(prob, digits=4))")
    end
end
```