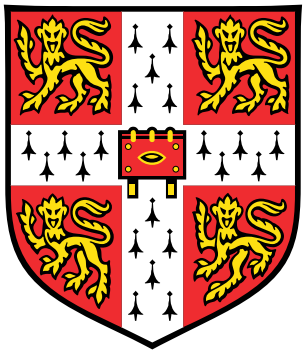
Fault-Tolerant Quantum Computation

Essay for Part III of the Mathematical Tripos

Miles D. Gorman



Department of Applied Mathematics and Theoretical Physics
University of Cambridge

May 2023

Contents

1

Introduction

Certain algorithms can leverage quantum phenomena to achieve a speedup over their best-known classical counterparts. The upshot of this is that these *quantum algorithms* offer a means of computing solutions to problems that might otherwise be intractable [1]. Because quantum algorithms are underpinned by quantum physics, their implementation requires a physical platform that is also 'quantum': namely, they need a *quantum computer*. Quantum computers are fundamentally different from the already well-developed *(classical) computers* we are used to, and are challenging to realize in part due to how delicate they are to *noise*.

Noise inevitably arises from various physical sources and is often modeled by *errors*, which are unintended *operations* that can significantly reduce the reliability of a computer's output [1]. Active techniques are generally required to overcome quantum errors and achieve accurate and meaningful quantum computations. In fact, at the time of writing (2023), noise is perhaps the greatest obstacle to realizing useful quantum computers.

This essay focuses on the active methods that allow for noise-robust quantum computations, and is organized as follows: in Chapter 2 I introduce *quantum error correction* as a naive way of combating errors. I then show where quantum error correction falls short for practical purposes, which motivates the definition of its more physically-relevant cousin: *fault tolerance*. In Chapter 3, I provide a method for constructing these so-called fault-tolerant quantum computations and show how this yields the core concept of this essay, the *threshold theorem*. The threshold theorem is a remarkable result that demonstrates the feasibility of quantum computing in a noisy environment: it proves that noisy quantum computations can be made arbitrarily reliable at an asymptotically-low cost in overhead, provided the hardware achieves error rates below some *threshold*. Naturally, these thresholds, which reach as high as $\sim 1\%$ [2], set a target error rate for quantum computing experimentalists to strive for. While present quantum devices can achieve error rates near this threshold [3, 4], there are further factors to consider when scaling up quantum computers, which I address later in the essay. Additionally, I analyze the threshold theorem's critical assumptions and briefly discuss other fault-tolerant constructions that lead to different versions of the threshold theorem before finally concluding in Chapter 4 with a summary and suggestions for future fault-tolerance research.

2

Quantum Error Correction and Fault Tolerance

In this chapter, I introduce the essay's necessary background. I will begin with a speed-run of *quantum error correction*: explaining each step at a high level before providing pedagogical examples using the *3-qubit repetition code*. I will then introduce *fault tolerance*, which - in short - makes quantum error correction practical under more physical error models and leads to the threshold theorem. More experienced - or more confident - readers may wish to skip the quantum error correction section as I included it largely to make many of the (perhaps otherwise abstract) fault-tolerant processes that I will use later in the essay more concrete and because it offers a natural way of defining requisite terminology and concepts.

Other preliminary comments: in this essay, I will consider *stabilizer codes*; I will not introduce the *stabilizer formalism* [5] but the essay is nonetheless self-contained. I will also only focus on *qubits*, which are two-level *quantum systems* living in a two-dimensional *Hilbert space* and are commonly represented in the *computational basis* (i.e. the *Z-eigenbasis*)

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

2.1 Quantum Error Correction

There is no ideal realization of quantum computers. Because physical quantum computers are unavoidably coupled to a noisy environment, their constituting qubits' states are subject to unintended evolutions called *errors*. Without special techniques, said errors are unknown to the computer's user, which presents a problem: errors can damage a computation's output to such an extent that no meaningful results are retrievable. What's worse, practical *quantum algorithms* generally require many quantum operations - often called *gates* - and considerable time (relative to error rates) which introduces a significant amount of *noise* and makes errors frequent over the algorithm's runtime. Indeed, errors undermine potential quantum advantage and pose a considerable challenge to realizing useful quantum computers.

Quantum error correction aims to build tolerance against errors. Showing how quantum error correction works will require an explicit form for errors. Here we will assume that errors

Name of Operator	Representation	Action on Basis States
Identity	$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$I \lvert a \rangle = \lvert a \rangle$
Bit-Flip / Pauli-X	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$X \lvert a \rangle = \lvert a \oplus 1 \rangle$
Bit-and-Phase-Flip / Pauli-Y	$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$Y \lvert a \rangle = i(-1)^a \lvert a \oplus 1 \rangle$
Phase-Flip / Pauli-Z	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$Z \lvert a \rangle = (-1)^a \lvert a \rangle$

Table 2.1: Pauli-basis generators in the computational basis. i denotes the imaginary unit, $\oplus$ denotes addition modulo 2, and $a \in \mathbb{Z}_2$ per Equation 2.1.

are single-qubit unitary transformations. While this is not a completely faithful treatment, as we will examine in Subsection 3.1.1, it is an instructive starting point. Expanding some arbitrary unitary error E in the *Pauli basis* gives

$$E(\alpha_I, \alpha_X, \alpha_Y, \alpha_Z) = \alpha_I I + \alpha_X X + \alpha_Y Y + \alpha_Z Z \tag{2.2}$$

where the identity I and Pauli operators X, Y, and Z are defined in Table 2.1 and $\alpha_{P \in \{I,X,Y,Z\}}$ are amplitudes that satisfy the normalization of E. Correcting this continuum of possible errors may seem a daunting task. However, as I will show, specific measurements of the state can collapse the error E onto either the identity (no error) or a single Pauli operator (an error). In this way, error correction needs 'only' to *correct* Pauli-Xs and Pauli-Zs independently on each qubit to correct an erroneous state (up to an unobservable global phase and noting that $Y = iXZ$). Here, correct means taking in erroneous data and returning it without errors (with high probability).

With errors defined, I can explain the error correction process. Following the high-level depiction of error correction in Figure 2.1a and the example in Figure 2.1b, the steps of error correction are as follows:

2.1.1 Encoding

An error correction procedure begins with k qubits of data in some state $|\psi\rangle$ that we aim to protect from noise. *Encoding* adds redundancy: the input state is fed into an *encoder* $\mathcal{E}$, which is an operation that acts on the input data $|\psi\rangle$ to produce an $n \geq k$-qubit state $|\psi\rangle_L$, i.e.

$$\mathcal{E}(|\psi\rangle) = |\psi\rangle_L . \tag{2.3}$$

To be more explicit, the encoding operation $\mathcal{E}$ consists of initializing $n - k$ (redundant) *ancilla* qubits and applying a series of gates that are selected according to the desired quantum error-correcting *code*, which defines the better-protected *logical state* $|\psi\rangle_L$ we want to produce.

Briefly, a quantum error correcting code is commonly characterized by the parameters n, k, and d written $[[n, k, d]]$ where n is the total number of qubits, called the *physical qubits*; k is the number of qubits of information, called the *logical qubits*; and d is the code's *distance*, which quantifies the minimum number of non-trivial single-qubit Pauli errors the code can

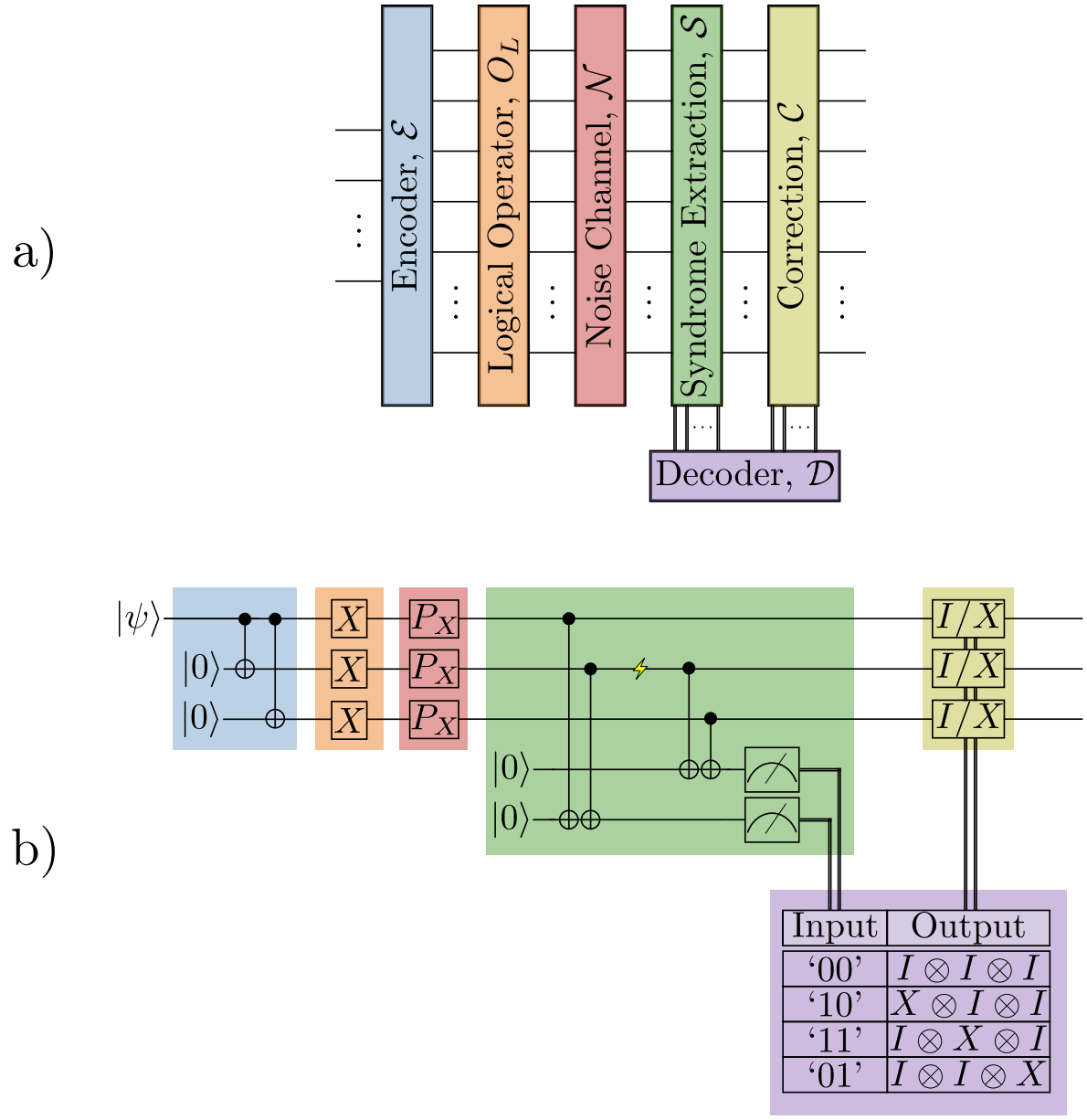


Figure 2.1: (a) a high-level view of quantum error correction *circuits* and (b) an example using the 3-qubit repletion code. In quantum circuits, time runs from left to right. Each double black line represents the path of a classical bit, and the horizontal single black lines each represent a qubit. Qubits are initialized at the left end of their horizontal line in a state specified by a *ket*. Here $|\psi\rangle$ is an arbitrary single-qubit state, defined in Equation 2.4. Qubits then encounter operations. The first two gates (black dot - *control* - say on qubit i and cross-hair - *target* - say on qubit j joined by a vertical line) are $CNOT_{i,j}$s defined in Equation 2.6. The X gates are Pauli-Xs, defined in Table 2.1; the P_X gates independently apply a Pauli-X to their qubit with some probability p; the dial in a box is a measurement in the computational basis which *projects* the state onto a state consistent with the measurement outcome (that is communicated as a classical bit through its connected classical *wire*); and the three I/X gates collectively apply either an identity on all qubits or a Pauli-X on a single qubit, conditioned on the output of the decoder. For clarity, the left (right) bit in the syndrome 'input' in the decoder is from the left (right) classical wire. Finally, ignore the yellow lightning bolt; it will be discussed in text when appropriate.

correct, or, more precisely, the minimum number of Pauli operators required to map one logical state of the code $|\psi\rangle_L$ to another logical state of the code $|\psi'\rangle_L$. As such, because error correction attempts to preserve the information inputted by keeping the computation in a logical state, we can only hope to correct $t = \lfloor \frac{d-1}{2} \rfloor$ errors, i.e. just under half the distance. Any more errors could push a logical state $|\psi\rangle_L$ *closer* to a different logical state $|\psi'\rangle_L$ and hence the procedure, to which the specific errors are unknown but are assumed to be less likely if there are more, would expect that the closer state $|\psi'\rangle_L$ is more likely to have been the information inputted.

To offer enhanced protection against noise, encoded states $|\psi\rangle_L$ will generally be highly entangled and have many more physical qubits than logical qubits, $n >> k$. In a hand-wavy sense, this results from encoding aiming to delocalize information so that localized errors are less likely to act directly on the information and corrupt it.

Example 1 (Encoding the 3-Qubit Repetition Code)**.** The 3-qubit repetition code is a $[[3, 1, 1]]$ code. It encodes a single qubit of information across three qubits. Additionally, it has distance 1, meaning it cannot correct even a single arbitrary error. However, if we restrict our errors to just Pauli-Xs, this code can correct precisely one error on any of its qubits. (The 3-qubit repetition code's inability to correct any arbitrary error renders it impractical for general purposes, but its simplicity helps introduce error correction.)

The procedure begins with some qubit of data,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.4}$$

where $\alpha, \beta \in \mathbb{C}$ satisfy $|\alpha|^2 + |\beta|^2 = 1$. The encoder then initializes two ancilla qubits, each in the state $|0\rangle$, to produce the 3-qubit state

$$|\psi\rangle \otimes |0\rangle \otimes |0\rangle \equiv |\psi 00\rangle \tag{2.5}$$

where $\otimes$ is the *tensor product*. Next, we apply gates of the form

$$CNOT_{i,j} \equiv |0\rangle_i \langle 0|_i \otimes I_j + |1\rangle_i \langle 1|_i \otimes X_j \tag{2.6}$$

using the notation that O_i is the operator O acting on i-th qubit and where implicit identities act on every other qubit. Encoding gives the state

$$CNOT_{1,3}CNOT_{1,2}\,|\psi 00\rangle = \alpha\,|000\rangle + \beta\,|111\rangle \equiv |\psi\rangle_L\,. \tag{2.7}$$

Observe that redundancy is added through repetition: encoding maps $|0\rangle \to |0\rangle_L \equiv |000\rangle$ and $|1\rangle \to |1\rangle_L \equiv |111\rangle$. For instance, if a measurement of the encoded state reveals $|001\rangle$, something must have gone wrong. Now consider the action of no error I, and each single-qubit bit-flip error on these $|000\rangle$ and $|111\rangle$ *codewords*:

$$\begin{aligned} &I\,|000\rangle = |000\rangle\,, \quad X_1\,|000\rangle = |100\rangle\,, \quad X_2\,|000\rangle = |010\rangle\,, \quad X_3\,|000\rangle = |001\rangle\,, \\ &I\,|111\rangle = |111\rangle\,, \quad X_1\,|111\rangle = |011\rangle\,, \quad X_2\,|111\rangle = |101\rangle\,, \quad X_3\,|111\rangle = |110\rangle\,. \end{aligned} \tag{2.8}$$

Because these erroneous states are unique, they are - in principle - distinguishable. However, learning the error is a non-trivial task, which we will consider shortly.

2.1.2 Logical Operations

After encoding, we feed the logical state $|\psi\rangle_L$ produced into a *logical operation*. A logical operation O_L, of some operator O acting on the data $|\psi\rangle$, satisfies

$$\mathcal{E}(O|\psi\rangle) = O_L|\psi\rangle_L = |\psi'_O\rangle_L \tag{2.9}$$

where $\mathcal{E}$ is encoding and $|\psi'_O\rangle$ is a (generally different) logical state of the code that depends on the operator O. That is, after encoding data, operations similarly need to be encoded: mapped to operators acting on the state $|\psi\rangle_L$ that gives the same output state as encoding the state $|\psi\rangle$ after having the desired operation applied. This results from encoded information $|\psi\rangle_L$ being delocalized to protect from local errors: local gates will necessarily no longer act directly on encoded data. (Although it is irrelevant until the next section, one does not want to de-encode, apply operations, and re-encode as this will expose the data directly to uncorrectable (logical) errors.)

Example 2 (A Choice for the 3-Qubit Repetition Code's Logical X Operator)**.** Before encoding into the 3-qubit repetition code, we had some logical qubit of information $|\psi\rangle$, Equation 2.4. Say the algorithm we wanted to implement was simply a Pauli-X, which maps our data to

$$X|\psi\rangle = \alpha|1\rangle + \beta|0\rangle\,. \tag{2.10}$$

Encoding this gives

$$X_L|\psi\rangle_L = \alpha|111\rangle + \beta|000\rangle\,. \tag{2.11}$$

Comparing this expression with $|\psi\rangle_L$, Equation 2.7, shows that X_L maps $|000\rangle \leftrightarrow |111\rangle$. So, $X_L = X \otimes X \otimes X$ is a (non-unique) choice for the logical Pauli-X operator on the 3-qubit repetition code. This gives the ability to perform operations directly on the protected information, and logical operators for a *universal set of gates* would allow arbitrary quantum computations on the encoded data.

2.1.3 Noise

Error correction then assumes that the state undergoes some noise process where it potentially picks up errors. Noise is modeled by a *noise channel* $\mathcal{N}$, which - to introduce error correction - we will say acts as

$$\mathcal{N}(|\psi\rangle_L) = E|\psi\rangle_L \tag{2.12}$$

where E is some error operator. We treat noise more rigorously in Subsection 3.1.1.

Example 3 (A Noise Model for the 3-Qubit Repetition Code)**.** Our example applies bit-flip errors on each qubit independently with probability p directly after logical operations. (For simplicity in these examples, we will only explicitly consider any one *pure state* in an erroneous *mixed state*. However, these ideas indeed generalize to mixed states.)

2.1.4 Syndrome Extraction

The redundancy afforded by encoding allows for *local consistency checks* that give a *syndrome*: classical information on the errors afflicting our encoded state. The procedure of retrieving the syndrome is called *syndrome extraction*. Syndrome extraction has the aforementioned subtle consequence that, due to its constituting measurements, it projects the potentially erroneous state onto a state consistent with those measurement outcomes. Accordingly, because codes are designed to give syndromes with sufficient information to correct Pauli errors (up to their distance), syndrome extraction effectively projects errors onto Pauli operators.

Syndrome extraction $\mathcal{S}$ acts on an encoded state $|\psi\rangle_L$ with error E according to

$$\mathcal{S}(E\,|\psi\rangle_L) = E'\,|\psi\rangle_L \otimes |s\rangle \tag{2.13}$$

where E' is the error that syndrome extraction has projected the original error E onto by retrieving the syndrome s, a *bit string* of information on the error E'. Error correction then aims to correct the error E' and return the encoded data $|\psi\rangle_L$ using the syndrome s.

Error correction is limited in the number of errors it can correct. This results from the concern that to avoid losing the potential for quantum computational advantage, we must perform syndrome extraction in such a way as to not collapse the encoded state to a classical one. We avoid this collapse by only measuring particular operators of a code, called *stabilizer generators* [5, 1], each of which separately arises from one of the $n-k$ redundant ancilla qubits introduced in encoding [6]. Their corresponding $n-k$ bits of information on the error reduces the errors' *state space* - which is of size $O(4^n)$ - by a factor of at most $2^{-(n-k)}$ due to the *Hamming bound* and *Shannon information*, so there will necessarily be syndrome degeneracy. (Errors include all possible combinations of identities and the three Paulis over the code's n physical qubits, and I use big-O notation here because - not to digress - not all errors need to be corrected [1].)

Example 4 (Method for Syndrome Extraction on the 3-Qubit Repetition Code)**.** For the 3-qubit repetition code, we will use the operators $S_1 = Z_1Z_2$ and $S_2 = Z_2Z_3$ (a choice of stabilizer generators for the code) as our local consistency checks in syndrome extraction. Each of the measurement outcomes of these operators will return a *syndrome bit*, one bit in the syndrome, and so this procedure gives a 2-bit syndrome. Note also that the operators we are checking, S_1 and S_2, are independent, so they each give unique information on the state; they are $+1$ eigenoperators on the logical states $|\psi\rangle_L$ of the code, Equation 2.7, so they leave the logical states unchanged when acting on them; and S_1 (S_2) checks that the left (right) two qubits are in the same Z-eigenstate: if these qubits are (not) in the same Z-eigenstate, S_i is a $+1$ (-1) eigenoperator of that state, giving a (non-)trivial syndrome bit denoted by '0' ('1'). Therefore, X errors potentially give non-trivial syndromes. In particular, they give the syndromes summarised in Table 2.2.

2.1.5 Decoding

A *decoder* $\mathcal{D}$ is a classical algorithm that takes in the syndrome s and returns an estimate of what error occurred, denoted by bit-string c: specifically,

$$\mathcal{D}(|s\rangle) = |c\rangle\,. \tag{2.14}$$

Error	Syndrome
I and $X_1X_2X_3$	'00'
X_1 and X_2X_3	'10'
X_2 and X_1X_3	'11'
X_3 and X_1X_2	'01'

Table 2.2: Syndrome of bit-flip errors on the 3-qubit repetition code. This table is computable using the method of Example 4. The quotation marks signify that the syndrome is a bit string, and here I (no error) is the 3-qubit identity. Finally, notice that the syndromes are doubly degenerate.

Non-trivial decoding algorithms are generally required because syndromes are not necessarily structured such that the errors they imply are immediately apparent.

Example 5 (Look-Up-Table Decoder for the 3-Qubit Repetition Code)**.** The simplest example of a decoder is the *look-up-table decoder*, a two-column table with each syndrome in one column, and the corrections they imply in the other. Table 2.2 and Figure 2.1b give look-up-table decoders for the 3-qubit repetition code, where their estimate of the error is the lowest *weight* operation corresponding to each syndrome. The weight of an operator is the number of Pauli operators it contains, so (recalling that here errors occur independently) this decoder gives us the highest likelihood of identifying the error and hence the highest chance of returning the state to its original error-free form.

One thing to note is that look-up-table decoders scale poorly: for an arbitrary $[[n, k, d]]$ code, the decoder will contain 2^{n-k} rows. This exponential memory overhead is hard to store and search over. As such, more sophisticated methods for decoding have been developed to preserve a good deal of accuracy in identifying the most likely error while reducing the process' time- and memory-complexity. We return to this topic in Subsection 3.1.1 and Section 3.2.

2.1.6 Correction

Now that the decoder has identified the error with bit-string c, it is time to correct. Because the errors we are considering are Pauli operators (*Hermitian*), to correct $\mathcal{C}$ is to apply the error estimate $\tilde{E}'$. That is,

$$\mathcal{C}(E' \left|\psi\right\rangle_L \otimes \left|c\right\rangle) = \tilde{E}'E' \left|\psi\right\rangle_L \tag{2.15}$$

where E' is the error on the logical state $\left|\psi\right\rangle_L$ and $\tilde{E}'$ is the decoder's estimate of the error E'. We say that the procedure succeeds when $\tilde{E}'E'$ is an eigenoperator (the identity operator, for example) of the encoded information $\left|\psi\right\rangle_L$ and fails otherwise, as then there would be a residual (non-trivial) operation on the state.

Example 6 (Error Correction on the 3-Qubit Repetition Code)**.** Putting all the above steps together, we have a protocol capable of correcting a single-qubit Pauli-X error on any of its qubits. To show an instance of this, recall from Equation 2.11 that after the logical operation, the encoded state is

$$\alpha \left|111\right\rangle + \beta \left|000\right\rangle . \tag{2.16}$$

Suppose an X_2 error occurs, mapping this state to

$$\alpha |101\rangle + \beta |010\rangle . \tag{2.17}$$

Then syndrome extraction, using Example 4, checks that the Z-eigenstates of neighbouring qubits matches, and, in this case, returns the syndrome '11'. From this syndrome, the decoder in Figure 2.1b speculates that the error that occurred was X_2. So X_2 is applied to the state to give back

$$\alpha |111\rangle + \beta |000\rangle : \tag{2.18}$$

the original logical state before the error, Equation 2.16. The process for correcting X_1 and X_3 (and I) is similar. Hence error correction is indeed successful in preserving the logical information under a single bit-flip error. Error correction repeated frequently - in this model - thus permits the recovery of the logical state with high probability if the error rate p is low: uncorrectable errors now have probability at least $O(p^2)$ (versus p without error correction).

2.2 Idea of Fault Tolerance

We can identify two issues with quantum error correction using the 3-qubit repetition code example from the previous section.

1. The code used does not independently correct both a phase- and bit-flip error, as is required to correct an arbitrary unitary error. Though, this was done for simplicity in introducing the subject. There do exist codes that facilitate the correction of bit- and phase-flip errors independently, for example, the *Shor code*, which is a $[[9, 1, 3]]$ code [1].

2. The error correction model is contrived. In practice, noise is not conveniently confined to after computation and before syndrome extraction: it is present throughout the whole circuit. Removing error correction's unphysical assumption on error locations carries two subtle but important consequences:

   (a) Consider an X error at the yellow lightning bolt in Figure 2.1b. This gives an X_2 error on the encoded state and the syndrome of '01', which implies an X_3 correction. In this case, due to a single error and error correction, the encoded state has been left with two errors; which is problematic because a correctable error has become uncorrectable. That is, we potentially lose our protection against errors and, in fact, make things worse by using error correction.

   (b) What's more, errors can potentially propagate through multi-qubit gates (see Figure 2.2). Errors that grow in weight by this mechanism present the same issue: they can map an innocuous situation of a correctable error to a scenario where the computation fails.

These issues with error correction motivate the design of protocols that hinder the spread of errors. Such protocols are said to be *fault tolerant*. Fault tolerance preserves a desired level of error correction even after relaxing the unphysical assumption on where errors can occur, making it of more practical relevance for robust quantum computing. (Noting that error correction still has a place in *quantum memories* and *quantum communication*, where the data spends much of its time idling or in transmission, so error correction's assumption on the location of errors is often a suitable approximation [7, 8].)

Figure 2.2: *Heisenberg picture* of quantum computing. One can calculate these diagrams by considering how Paulis commute with $CNOT$s. For instance, the first diagram is calculated as $(X \otimes I)CNOT_{1,2} = CNOT_{1,2}(X \otimes X)$. All the operators here are *Hermitian*, so the diagrams are readable left-to-right or right-to-left. Notice the highlights on the qubit wires, which serve as a visual aid to see 'how' the Paulis propagate. Similar relations hold for other multi-qubit gates.

2.3 Definition of Fault Tolerance

Fault-tolerant models dub any location where an error might spawn as a *potential fault location* and any such location that introduces an error as a *fault*. *Error models* govern potential fault locations by determining the probabilities of errors a fault produces. Typically, there will be a potential fault location on each qubit after each gate, with identity gates applied to qubits in memory to capture memory errors. *Ideal* operations are without faults and will prove useful despite being unphysical.

A circuit giving the same output distribution of logical information as some other circuit is said to *simulate* that circuit. A fault-tolerant circuit simulates, with high probability, some ideal circuit in the presence of noise. More precisely, adopting a definition similar to those in [9, 10, 11],

Definition 1 (Fault Tolerance)**.** A level-l fault-tolerant encoding $FT^{(l)}$ is a mapping from an ideal logical circuit $C^{(0)}$ to a level-l fault-tolerant version of that circuit $C^{(l)}$: a circuit that simulates the ideal circuit $C^{(0)}$ to some desired level of accuracy $\epsilon > 0$, as controlled by parameter l, under a non-trivial error model $\mathcal{E}$ and for any level l larger or equal to some cutoff l_0. That is, a circuit construction aiming for fault tolerance,

$$FT^{(l)}(C^{(0)}) = C^{(l)}, \tag{2.19}$$

is indeed fault tolerant if it satisfies

$$||(FT^{(l)})^{-1} \circ \mathcal{E} \circ FT^{(l)}(C^{(0)}) - C^{(0)}|| \leq \epsilon \tag{2.20}$$

for any $\epsilon > 0$, any logical circuit $C^{(0)}$, and any $l \geq l_0$. A circuit C here is the probability distribution of the outputted states of said circuit, and so the operations $FT^{(l)}$, $\mathcal{E}$, and $(FT^{(l)})^{-1}$ change the distribution of outputted states of their input accordingly. To be clear, $\mathcal{E}$ updates the states to be consistent with some given error model, and $(FT^{(l)})^{-1}$ performs ideal error correction and then undoes the level-l fault-tolerant mapping ideally.

The importance of fault tolerance is captured by this definition: the ability to increase the level l of a fault-tolerant encoding $FT^{(l)}$ allows for an arbitrarily-close-to-ideal simulation of any circuit under a given error model. The parameter l is needed because there will be some (decreasingly small with increasing level l) probability of a malignant error for any finitely-sized code.

3

Fault-Tolerant Constructions and Thresholds

As a road map for this chapter, I will present a fault-tolerant construction for simulating ideal circuits, the *threshold theorem*, and how to compute *thresholds*.

3.1 A Fault Tolerant Construction

In this section, I will provide the three remaining ingredients needed to present a version of the threshold theorem: the assumptions; *concatenation*, which scales the level l of our fault-tolerant construction; and *gadgets*, used in our fault-tolerant mapping $FT^{(l)}$.

3.1.1 Assumptions

Here, I explain the assumptions underpinning my essay's version of the threshold theorem. I mark assumptions central to a threshold's existence with a '‡' symbol, and I include further assumptions - not essential to a threshold's existence - that simplify the analysis to follow.

The first group of assumptions focuses on gates. Begin by assuming the existence of

1.1.‡ *parallel operations*: operations in consecutive timesteps - not acting on the same qubit/s - can be performed simultaneously.

Where a time step is, say, the time of the slowest operation. The crux of this assumption is necessary for a threshold: if we have n qubits and each one succumbs to errors with an unacceptably high probability after t time steps, we must operate on at least n/t qubits per time step to have a chance to compute reliably [7]. So, as the number of qubits in the computation n increases (for a fixed maximum number of acceptable timesteps t), the number of qubits acted on in each time step must increase at least linearly in qubit count n for robust computation. Thus, if we cannot at least operate on some constant fraction of qubits per time step, the probability of a successful simulation vanishes. In practice, parallel computations are possible, albeit more challenging, evidenced by parallel gates generally being accompanied by higher error rates [12].

We also use

1.2. *long-range multi-qubit gates*: any set of qubits can be acted on by multi-qubit gates regardless of their physical location in a quantum device.

While thresholds exist for fault-tolerant procedures that only require short-range interactions (in low-dimensional geometries) [11, 13], we postulate this assumption because it is convenient to apply multi-qubit gates however we need. In reality, not all quantum platforms guarantee connectivity between all qubits, though improving connectivity is a focus of experimental efforts [3]. Indeed, connectivity would need to be considered in analyzing a fault-tolerant scheme on a particular quantum device. Another factor to consider is that, given they are performable at all, long-range entangling operations often come with higher error rates [14].

The next group of assumptions center around mid-computation *preparations* and measurements. In particular, we require

2.1.‡ *on demand ancillas*: during the computation, fresh ancilla states can be initialized.

Without this assumption, qubits (now prepared well in advance) are more likely to have errors by the time they are used, reducing the computation's reliability. Exacerbating this issue is the fault-tolerant circuit's size blow-up, where the circuit's growth will be so large in the infinite limit that without specific techniques to resist errors in the ancillas, erroneous ancilla qubits become inevitable. In fact, without this assumption, fault tolerance would require the circuit to grow exponentially [15]. Note we cannot avoid this issue: ancilla qubits are necessary for quantum error correction, following from the interpretation that syndrome extraction is needed to move the entropy of noisy information onto ancilla qubits which are then removed from the system to leave the information with (hopefully) less noise.

Architectures can allow ancilla initialization during the computation [16, 17]. However, doing so comes with challenges. For instance, initializing qubits can take an unpredictable amount of time - due to relying on *post-selection* for example - in which case the computation may either need to wait in memory until ancillas are ready or the ancillas will need to be prepared well in advance [16, 18]. In either scenario, there is an increased likelihood of errors. This has led to experimental efforts focused on reducing the state preparation time, reusing qubits, and making preparation take a consistent amount of time [17, 18]. A more rigorous analysis would need to consider hardware-dependent factors like this.

To simplify matters, we also want

2.2. *mid-circuit measurements*: qubits can be measured thought the computation.

This assumption is unnecessary because error correction does not require measurements - let alone mid-circuit measurements [19, 20]. However, we make this assumption anyway because measurement-free error-correction schemes are suspected to have substantially lower thresholds [19], and allowing mid-circuit measurements will help with our analysis. Furthermore, measurements during the computation are physically possible [4, 16]. Something to consider is that measurements can take a relatively long time. However, we can design fault-tolerant circuits such that the threshold is not significantly affected [19].

The following two assumptions constrain the classical computations supporting our quantum computation. The first is

3.1. *ideal classical computing*: all classical computations are error-free.

This assumption is unnecessary for a (*polylog*-overhead) threshold as although errors in our classical computing processes would add more potential fault locations (reducing the threshold), they could be combated with *classical error correction* if necessary [1, 21]. Nevertheless, we take this assumption because it is convenient and physically motivated: classical computations are robust [21].

Additionally, we ask for

3.2. *free classical computing*: classical computations take zero time.

This assumption is not totally realistic in practice, and - ironically - if it were to hold, there would be no need for quantum computers. Despite this, the assumption of free classical computation is not always necessary for a (low-cost) threshold to exist, as waiting for classical computations 'only' adds more potential fault locations where the quantum computer must wait for a classically computed result. Sufficiently fast and good-scaling classical computations will not add so many potential fault locations that they prevent a threshold.

Still, an entire sub-field of quantum error correction focuses on improving the runtime of decoding algorithms without considerably impacting their ability to identify errors accurately [22, 23, 24]. Noting that a less accurate decoder cannot correct as many errors, causing the number of malignant errors to grow and the threshold to drop. There are also many other places where classical processing may slow down a quantum computation [16, 25, 26, 27], and work has gone into scheduling quantum computations around classical computation's limitations and finding other ways to avoid excessive waiting [16].

The last lot of assumptions restrict the noise models we can consider. The most important of these assumptions is

4.1.‡ *locally decaying noise*: the probability of a specific f faulty locations in a circuit is $\leq p^f$ where p is the probability of any one fault.

For there to be a threshold, the probability of multiple specific faults occurring in the same time step needs to be at least exponentially suppressed in the number of specific errors they cause. Otherwise growing the circuit larger - as fault tolerance requires - can have a vanishing ability to succeed. Implicit is also that the physical error rate p is small, so that error correction is not overwhelmed with errors; p is (near-)independent of the computation's size, so we can safely scale our computation; and p characterizes all errors in the system. This last requirement is unnecessary as multi-parameter thresholds exist [28, 29], but - for simplicity - we will not consider these here.

Experimentalists are still in the process of determining how physically realistic this assumption is. It is difficult to demonstrate in practice as it inherently requires scaling to large qubit counts - pushing the bounds of devices currently available and going well beyond what was possible in the past. Nevertheless, recent results indicate that this assumption perhaps holds sufficiently well for fault tolerance to be effective (at least on small scales) [30].

While not necessary, we will only consider

4.2. *stochastic noise*: any specific set of f faulty locations occur with a probability p_f.

This assumption says that errors are chosen randomly and independently from one another, i.e. a fixed number of faults occur in any specific set of potential fault locations with the same probability. However, this assumption is not entirely accurate in practice [31, 32]. This is made worse by our model assuming that errors occur after gates, while - in reality - gates are

continuous evolutions. Errors amid gates can lead to correlated errors called *crosstalk* errors, which do not fit this assumption. Furthermore, correlated errors can arise from other sources, like *cosmic rays* [30]. Results indicate that *non-Markovian* environments more accurately model noise, where a Hamiltonian is used to couple a system to an *environmental bath*. In the case of a non-Markovian environment, a threshold still exists under appropriate assumptions [33].

Finally, we

4.3. *ignore leakage errors*: errors do not move any qubit outside of its two-dimensional subspace.

This assumption confines the amplitudes of all states to be within the computational subspace of however many (assumedly two-dimensional) qubits are being used. So errors map computational subspace states to other computational subspace states; consequently, Paulis (and the identity) can model errors. In reality however, 'qubits' are generally implemented by quantum systems with more than two dimensions. While this can have advantages in other areas of quantum computing, it means that our assumption is not implicitly satisfied - as noise can allow physical 'qubits' to access levels outside of their computational subspace [7]. These *leakage errors* are prevalent in many quantum devices and are very damaging to quantum error correction as they spread errors throughout the hardware and can corrupt syndrome measurements [34]. Thresholds still exist in the presence of leakage errors but require special techniques that effectively transform them into Pauli errors; for instance, passively through *Knill syndrome extraction* [26] or actively using *leakage-reduction units* [35]. Note however, any additional expense carried by these techniques reduces thresholds.

3.1.2 Concatenation

From the definition of fault tolerance, Definition 1, for arbitrarily accurate computations, a fault-tolerant construction requires the ability to be scaled using a parameter l. We do this by *concatenating*: growing a fault-tolerant protocol's underlying error-correcting code from smaller ones. More precisely,

Definition 2 (Concatenation)**.** Concatenation is the process of separately encoding each of the n physical qubits of an $[[n, k, d]]$ error-correcting code into an $[[n', k', d']]$ error-correcting code, as illustrated in Figure 3.1.

Indeed we can produce an arbitrarily-high-distance code by repeatedly concatenating an $[[n, 1, 3]]$ code with itself, as argued in Figure 3.1. In fact, self-concatenating an $[[n, 1, 3]]$ code will be sufficient to produce a threshold, so, for the sake of simplicity, I restrict all further discussion to just a single $[[n, 1, 3]]$ code; the arguments I present do generalize to other code families though [11]. Naturally, we set the level l of the fault-tolerant protocol to be the number of times this code is concatenated with itself, so we can correct at least exponentially more errors with each level l, per Figure 3.1.

However, as we continue to concatenate, the size of our circuit also grows: with each concatenation, we must encode operations in the circuit over exponentially more physical qubits. So, while our ability to correct errors increases with the level of concatenation, so too does the number of potential faults. Seeing how these two competing factors balance is a challenging task that forms much of the remainder of the essay.

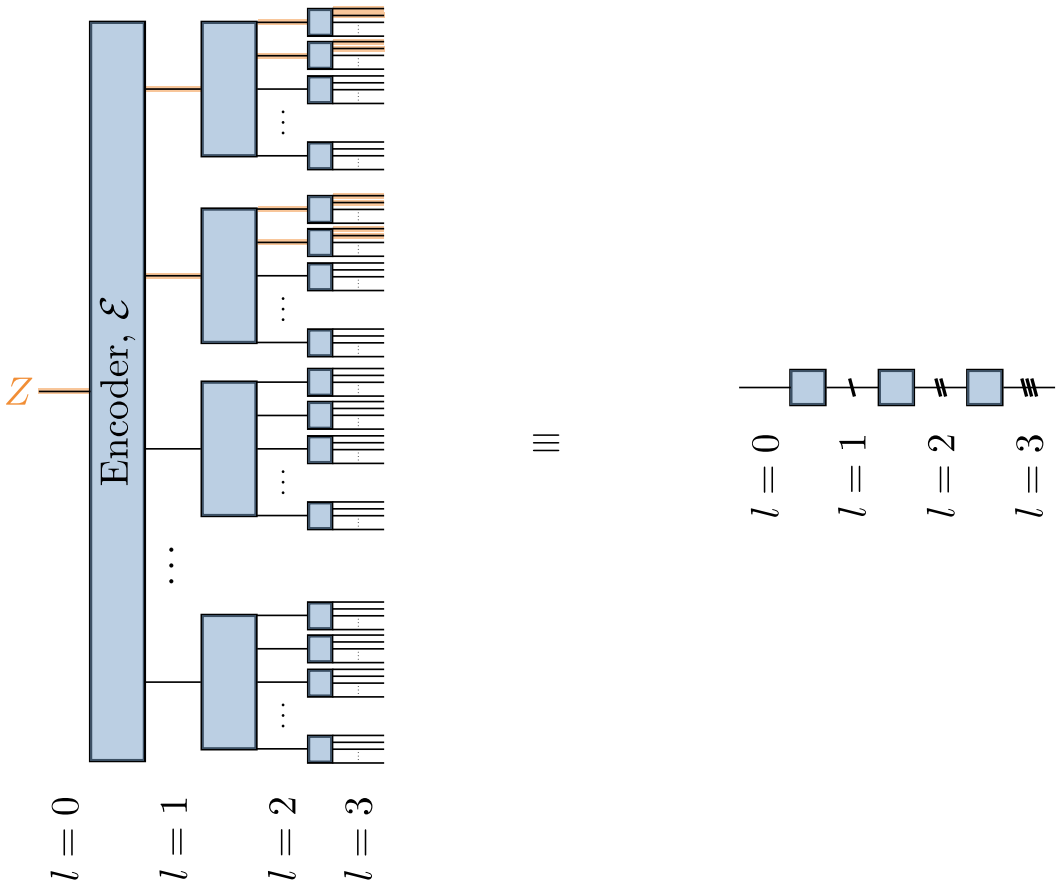


Figure 3.1: Self-concatenation of an $[[n, 1, d]]$ code. The blue boxes are the code's encoding circuit. I show the level l on each layer of concatenation. The left- and right-hand side diagrams are equivalent as we introduce the notation that w back-slashes on a qubit wire denote w iterations of concatenation. Notice the Z on the left-hand side diagram and how it propagates through the encoding circuit, shown as highlights on the relevant qubits. This is a visual representation of how concatenation impacts distance. Without loss of generality, assume that this Z gives the lowest-weight logical Pauli of the code (otherwise, select the operator that indeed gives the lowest-weight logical Pauli, i.e. Pauli-X or Pauli-Y). The first encoding will propagate this Pauli to the code's corresponding logical Pauli per [6]. Hence the operator propagates to a Pauli string of weight equal to the distance d. These d constituting Paulis then similarly propagate through the next layer of encoding to Pauli strings of weight at least d each, giving the smallest logical Pauli a weight of at least d^2. All other logical Pauli operators at level 1 must similarly give a minimum weight logical Pauli of $\geq d^2$ by level 2, so the *level-2 code* has a distance of $\geq d^2$. Apply this argument recursively to achieve a distance-$\geq d^l$ code by level l of concatenation. We can also see using this visual aid that the number of qubits at each level grows by a factor of n and that no additional logical operators have been created through concatenating, consistent with no information being added to the system; accordingly, the concatenated code similarly encodes just $k = 1$ logical qubit. So concatenating an $[[n, 1, d]]$ code $l \geq 1$ times produces an $[[n^l, 1, \geq d^l]]$ code. Note that this argument straightforwardly generalizes to other code constructions.

3.1.3 Fault-Tolerant Gadgets

We are yet to fix precisely how in our construction $FT^{(l)}$ maps the logical circuit we want to simulate $C^{(0)}$ to its fault-tolerant version $C^{(l)}$.

The schematic we select for this is to decompose the circuit we wish to fault-tolerantly simulate $C^{(0)}$ into its fundamental components (preparing qubits, performing logical gates, and taking measurements), and then reconstruct it by replacing each of these components with some *fault-tolerant gadgets*, defined on our level-l concatenated code. And, because we want to correct errors, we also insert fault-tolerant error correction gadgets. We place constraints on these gadgets such that the simulating circuit satisfies the definition of fault tolerance, Definition 1. The basic idea is that we construct gadgets such that, for a code capable of correcting t errors with certainty, there are $\leq t$ outgoing errors on each encoded block, assuming the gadget encountered $\leq t$ total errors and faults. We require this constraint on the propagation of errors so that a circuit remains in a correctable regime if the number of faults and errors was correctable to begin with. Further, we require that gadgets accurately simulate their operation when $\leq t$ errors and faults occur. We make no such constraint when there are $> t$ errors as the code cannot hope to correct all errors of this weight, so the computation could fail anyway. Although not essential for this essay, these definitions of fault-tolerant gadgets are made more exact in [10, 11]. Note, a gadget's construction in terms of gates will depend on the code and is not necessarily unique: though non-unique gadget implementations may not be equivalent in their robustness against noise.

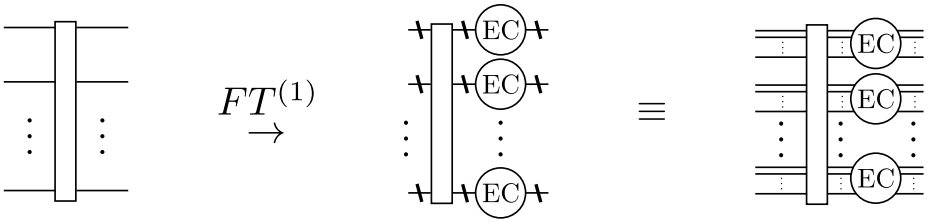
In particular, we encode preparation locations as

(3.A)

where we have placed a *level-l* fault-tolerant error correction gadget (the circle labeled EC) after the preparation location. These level-l fault-tolerant error correction gadgets consist of decoding, correction, and a fault-tolerant version of syndrome extraction. Note that to be in line with the fault-tolerance literature, I have switched the circuit notation of qubit preparation, and I will do the same with measurements later.

Many techniques with distinct advantages and disadvantages are available for constructing fault-tolerant error correction gadgets from gates. While these details are not required to understand the remainder of the essay, an interested reader may find [36] helpful for further reading. Moreover, fault-tolerant state preparation can be done - for example - by using a variation of fault-tolerant error correction to encode into the level-l code (measure the code's stabilizers and Z logical operators to project onto the *code space* and then attempt to correct errors or discard the state and try again if it gives a non-trivial syndrome) and then perform level-l fault-tolerant gates discussed below to produce the desired state [11].

We encode gates as follows,

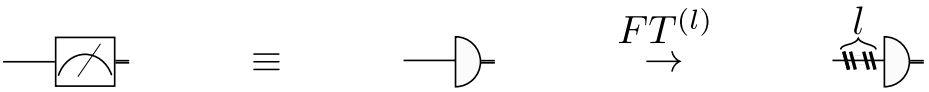


(3.B)

where level-l gadgets are constructed by replacing all level-0 gadgets (physical operations) constituting a level-$(l-1)$ gadget with level-1 gadgets. So one only needs to define a level-1 gate gadget for each gate in a universal gate set to find any level gate gadget and perform arbitrary fault-tolerant quantum computations.

The simplest implementation for fault-tolerant logical operations is *transversal* gates: gates that act bitwise and hence limit the propagation of errors between encoded blocks [1]. I show a transversal $CNOT$ in Diagram 3.G. Unfortunately, a transversal universal gate set is forbidden [37], so more expensive methods for fault-tolerant computing are required; including, for example, switching code mid-computation to a code with the same distance and the desired operation being transversal [38].

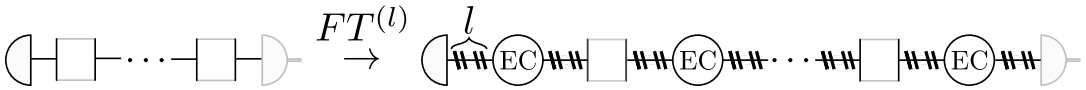
Finally, we encode measurements using the mapping



(3.C)

Fault-tolerant syndrome extraction is essentially nothing more than a fault-tolerant measurement (of stabilizer operators), so we can adapt syndrome extraction techniques to perform level-l fault-tolerant measurements of other operators, like the level-l code's logical operators. The logical state can also be operated on with fault-tolerant gates before the measurements to take arbitrary fault-tolerant measurements.

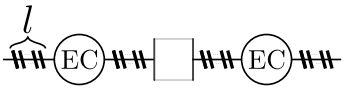
Putting these gadgets altogether, a qubit in a circuit under a level-l fault-tolerant mapping becomes



(3.D)

where the top and bottom of gates are greyed out to denote their potential connection to other qubits, and the measurement is greyed out because it is not necessarily required.

Constructing our fault-tolerant mapping $FT^{(l)}$ in this way has four significant advantages. (1) Frequent error correction reduces the likelihood of an unrecoverable accumulation of errors. (2) Gadgets make it easier to find fault-tolerant constructions: to satisfy the sweeping definition of fault-tolerance, Definition 1, (we will show) one needs to construct just a finite number of gadgets, as opposed to an entire circuit. (Note that not all protocols must be gadget-based to meet the definition of fault tolerance: Definition 1.) (3) The recursive construction of gadgets leads to a self-similarity between different levels that we will exploit to find thresholds. (4) The fault-tolerant circuit exhibits a periodic structure, comprising of some repeated *unit cells* (often called *extended rectangles* in the literature), which makes proving results on the circuit as a whole more tractable. We will choose the unit cell to be



(3.E)

at level-l. (The other unit cells are preparation followed by error correction and error correction followed by measurement.) Per Diagram 3.D, unit cells overlap. This is intended to simplify the threshold theorem's proof; in short, overlapping unit cells leave no room for errors to occur between unit cells where there is no error correction immediately following to

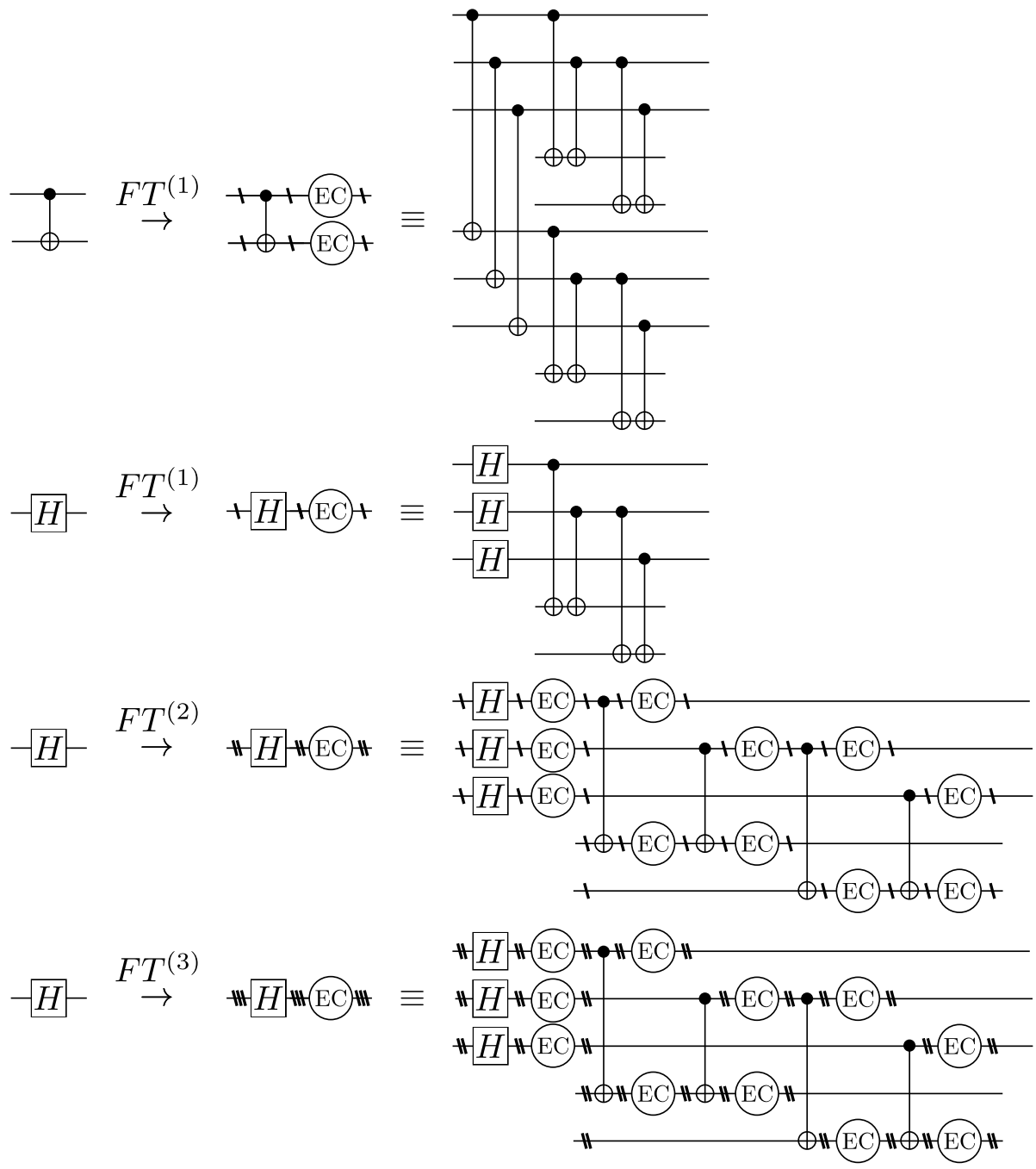


Figure 3.2: Pedagogical example showing the self-similarity of gadgets. The H gates are *Hadamards*, $H \equiv \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. In our construction, we must explicitly define the level-1 gadgets we require. We do this in the first two diagrams (non-fault-tolerantly and without memory errors) for a $CNOT$ and a Hadamard on the 3-qubit repetition code. Notice that the recursive definition of gadgets can equivalently be stated as a level-l gadget is constructed by replacing each gate in a level-1 gadget with its level-$(l-1)$ gadget [10]. With this, we construct the level-2 Hadamard gadget for this code, shown in the third diagram. One could plug in the circuit elements from the first two diagrams to find this large level-2 gadget in terms of physical gates and qubits. We then also show the level-3 Hadamard gadget in the fourth diagram, which one could similarly plug in the lower-level Hadamard and $CNOT$ gadgets for to find the yet larger level-3 gadget in terms of physical gates and qubits. (An enthusiastic reader attempting this might notice that the gadget's growth is not upper-bounded by powers of the constant factor we might naively expect from Lemma 2, but this is not an issue as it results from the Hadamard gadget relying on $CNOT$s whose gadgets grow faster with level l.) Continuing in this way, see that all levels of the Hadamard gadget look similar to its level-1 gadget.

correct said errors, so the number of potential fault locations in a unit cell does not exceed that which is contained within the unit cell itself [10, 11].

A threshold theorem based on gadgets and unit cells critically relies on two lemmas, each respectively answering how often unit cells *fail* and how gadgets grow with concatenation. Consider first the former: how often do unit cells fail,

Lemma 1: Unit Cell Failure Probability

A level-l unit cell will fail with probability at most

$$(cp)^{2^l}/c \tag{3.1}$$

where p is the physical error rate, and c is a combinatoric constant for the number of ways to select two potential fault locations in the unit cell (and the threshold's inverse, as we will see). Also, fail here means the unit cell has experienced more faults than the selected code, concatenated l times, can correct with certainty.

Note: the unit cell with the most potential error locations will give an upper bound on the failure rate of all unit cells.

This lemma is central to the threshold theorem, and its rigorous proof is intricate [10]. I will sketch the argument to avoid getting bogged down in details.

Sketch of proof: recall that our construction of fault tolerance is built on the recursive definition that a level-l gadget is constructed by replacing each level-0 gadget in a level-$(l-1)$ gadget with a level-1 gadget. This gives a *self-similarity* between gadgets of different levels l, where they look like their level-1 gadget when viewed from level $l-1$, as illustrated in Figure 3.2.

The idea is then (approximately), because the code we are concatenating is distance-3, a level-l gadget fails when two or more of its constituting level-$(l-1)$ gadgets fail. In a more intuitive, hand-wavy sense: we perform error correction in the level-1 constituting gadgets first; they fail when there are two or more errors on the physical qubits, and when they fail, they produce (due to their constraints) up to what is a logical error on the level-1 concatenated code which then - with other level-1 encoded qubits - is fed into level-2 error correction (without level-1 error correction checks); this level-2 error correction fails if it encounters two level-1 logical errors (at level-2 concatenation we can correct two errors that are logical errors in the level-1 code), which occurs with the highest probability (fewest physical errors) when two of the level-1 gadgets fail; and so on.

Therefore, a level-l gadget will fail with probability

$$P_l \leq \sum_{i=2}^{L_l} \binom{L_l}{i} P_{l-1}^i (1-P_{l-1})^i \leq \binom{L_l}{2} P_{l-1}^2 \tag{3.2}$$

where L_j is the number of potential fault locations at level-j (e.g. typically one on each encoded block after each level-$(j-1)$ gadget), and P_j is the probability of a *level-j error* (i.e. an error composed of enough physical errors to overwhelm a level-j error correction gadget) at any of the L_{j+1} potential fault locations. The first bound holds because we know we succeed with certainty when there is just one error at that level. The second bound holds due to a non-instructive proof using hypergeometric functions, or perhaps more intuitively, we enforce two errors at that level on any of that level's potential fault locations and do not mind what happens at the other potential fault locations [39]. (Note that one may achieve a tighter bound on the failure rate here

by calculating only the malignant errors that cause a unit cell to fail; our criterion is pessimistic as not all pairs of errors will necessarily cause failure.)

Realizing that $L_j = L_{j'} \equiv L \ \forall j, j' \geq 1$ due to the self-similarity of the levels mentioned before, we find

$$P_l \leq \binom{L}{2} P_{l-1}^2 \tag{3.3}$$

which can be recursively reduced down to level-1, where we consider errors afflicting physical qubits with probability $P_0 \equiv p$. Solving this relation in terms of the physical error rate p, we find an upper bound on the failure rate of a level-l unit cell to be

$$P_l \leq (cp)^{2^l}/c \tag{3.4}$$

where $c \equiv \binom{L}{2}$.

So a unit cell's failure rate will be reduced (when the error rate p is below some computable threshold value $P_{th} \equiv c^{-1}$) doubly-exponentially in the level of concatenation. The second thing to consider is how these gadgets scale with concatenation,

Lemma 2: Concatenation's Impact on Gadget Size

A level-l gadget will have an upper bound on its number of qubits, depth, potential fault locations, and gate count of G^l for some constant G.

Proof: each level-1 gadget will have a constant number of qubits (or gates or depth or potential fault locations). Let the largest of these across all level-1 gadgets be denoted by the constant G. The result then follows directly from the recursive construction of higher-level gadgets.

So gadgets grow exponentially in their level of concatenation. With these Lemmas, I can now present the threshold theorem.

3.2 Threshold Theorem

The threshold value, and in fact the very existence of a threshold at all, depends strongly on the fault-tolerant construction used and the assumptions made. Indeed, other variations of the threshold theorem exist - based on different constructions and different assumptions - and they can offer higher or lower threshold values and different implementation overheads. I will begin by sketching a proof for the threshold theorem [7, 10, 11, 20] using the construction presented so far: fault-tolerant gadgets, self-concatenated $[[n, 1, 3]]$ codes, and our assumptions. I will then show another fault-tolerant protocol - in less detail - that indeed offers a threshold at a more favorable qubit cost asymptotically [9, 13, 24].

So, the threshold theorem based on the fault-tolerant construction I have presented,

Theorem 1: Threshold Theorem

Any ideal circuit $C^{(0)}$ may be simulated fault tolerantly to arbitrarily small accuracy $\epsilon > 0$ when the error rate p is less than some threshold P_{th} using a circuit $C^{(l)}$ that has a depth, qubit count, gate count, and potential fault location count that is at most

$$O\left(polylog(N/\epsilon)\right) \tag{3.5}$$

times larger than the ideal circuit $C^{(0)}$, where polylog is logarithm raised to some power; N is the number of potential fault locations in a noisy version of $C^{(0)}$; and the level l depends on the desired simulation accuracy ϵ.

Sketch of proof: following [10, 11] closely, we begin by assigning $\{p^{(\text{ideal})}\}$ and $\{p^{(\text{actual})}\}$ to be the probability distribution of our simulating circuit's output states without noise and with noise respectively. Quantify the error by the *absolute-value norm* of the difference between these distributions:

$$\epsilon \equiv ||p^{(\text{actual})} - p^{(\text{ideal})}|| = \sum_i |p_i^{(\text{actual})} - p_i^{(\text{ideal})}|. \tag{3.6}$$

The simulation cannot be guaranteed to be without logical errors if any of its constituting unit cells fail (although this seems intuitive, it is intricate to prove [10, 11]). Thus, using Lemma 1, the level-l simulation's failure probability has the bound

$$\tilde{P}_l \leq N(cp)^{2^l}/c \tag{3.7}$$

where p is the physical error rate, c is a constant (the maximum number of ways to select two errors across all level-1 unit cells), and N is the number of potential fault locations in a noisy version of the ideal circuit (and hence also at least the number of locations that the highest-level unit cells in the simulating circuit can fail at i.e. produce more errors on an encoded block than can be corrected).

Now consider the probability of any one of a level-l simulating circuit's noisy states,

$$p_i^{(\text{actual})} = (1 - \tilde{P}_l)p_i^{(\text{ideal})} + \tilde{P}_l p_i^{(\text{fail})} \tag{3.8}$$

where $\{p^{(\text{fail})}\}$ is some probability distribution for the simulating circuit when it fails. Using this, the statistical error, Equation 3.6, can be rewritten as

$$\epsilon = \tilde{P}_l \sum_i |p_i^{(\text{fail})} - p_i^{(\text{ideal})}| = \tilde{P}_l ||p^{(\text{fail})} - p^{(\text{ideal})}|| \leq 2\tilde{P}_l \leq 2N(cp)^{2^l}/c \tag{3.9}$$

where in the first inequality, we used that the absolute-value norm of the difference between two probability distributions cannot exceed 2, and in the second inequality, we used our bound on $\tilde{P}_l$ from Equation 3.7. (See that the error in our distribution ϵ is at least doubly-exponentially suppressed in the level l of concatenation only when the error rate is below threshold: $p < P_{th} \equiv 1/c$.)

Rearrange the above expression to find that for a level-l simulation to achieve at least some desired level of accuracy ϵ we require a level of concatenation l that satisfies

$$2^l \geq \frac{\log(2N/\epsilon c)}{\log(1/cp)}. \tag{3.10}$$

Now consider the smallest level of concatenation w that achieves this accuracy ϵ, which - per the above inequality - must satisfy

$$w = \left\lceil \log\left(\frac{\log(2N/\epsilon c)}{\log(1/cp)}\right)\right\rceil. \tag{3.11}$$

(Note: we examine just level w because we cannot consider all levels l that satisfy the condition to achieve an accuracy ϵ at the same time - because we want to find the cost to achieve this level of accuracy and there is no upper bound on the number of concatenations l so the maximum size of the circuit is similarly unbounded from above.)

Using Lemma 2, if the maximum qubit count (or depth or potential fault location count or gate count) over all level-1 gadgets is denoted by constant G, then the lowest-level gadget that achieves an accuracy ϵ (i.e. a level-w gadget) will have a qubit count (or depth or potential fault location count or gate count) of

$$G' \leq G^w = (2^w)^{\log(G)}. \tag{3.12}$$

Because the error rate p and the number of pairs of potential fault locations c are fixed constants for a given regime, we can combine the above two expressions to find that the qubit count, depth, potential fault location count, and gate count of a gadget used in a simulating circuit achieving accuracy ϵ is no more than $O(\text{polylog}(N/\epsilon))$ times larger than its corresponding physical operation (i.e. its corresponding level-0 gadget in say an ideal circuit). So, each of these quantities (qubit count, depth, potential fault location count, and gate count) in a simulating circuit achieving accuracy ϵ will also respectively grow by a factor of

$$O(\text{polylog}(N/\epsilon)) \tag{3.13}$$

compared to the ideal circuit (albeit a generally different factor for each quantity).

It may not be clear why polylog scaling gadgets yield polylog scaling simulating circuits, so in this paragraph, I will explain why explicitly. For gates, this follows from each gate in the ideal circuit corresponding to a gadget whose gate counts scales by at most this polylog factor. An analogous argument applies to the scaling of potential fault locations. The depth scales by this polylog factor because parallel gates in the ideal circuit remain parallel after l levels of concatenation (up to the depth of the largest gadget, which scales polylogarithmically). The number of qubits needed scale by this polylog factor also, assuming that qubits are reusable: per our assumptions, we can run as many gadgets in parallel as there are qubits in the ideal circuit, so our ancilla qubits must be able to support at most the gadget requiring the greatest qubit count of any of the gadgets at any one time acting on each qubit in the ideal circuit. The ancilla qubits of each of those gadgets, which grows by the polylog factor, can then be reused in the next layer of gadgets, giving our result. Moreover, see that we can further reduce the qubit overhead at the expense of our circuit's parallelism; we discuss a remarkable extension of this idea shortly. Note that if qubits are not reusable, the number of qubits will grow by a factor dependent on the maximum number of gadgets acting on any one qubit of the simulating circuit, as this gives an upper bounded factor on how many more qubits we need per qubit in the ideal computation.

Comments: see that the number of potential fault locations in a noisy copy of the ideal circuit N is upper bounded by depth (or gate count in negligible-memory-error-rate regimes) times the qubit count of the ideal circuit, which gives a bound in terms of different parameters. Finally, notice that the accuracy ϵ is analogous to that of the fault-tolerance definition, Definition 1, and so this polylog-scaling construction indeed satisfies the definition of fault tolerance - when the error rate is below threshold.

In words, the threshold theorem says that arbitrarily accurate quantum computations are achievable at a polylogarithmic cost, provided that the error rate is below a certain threshold value. Achieving sub-threshold error rates is naturally a target for experimental endeavors; after accomplishing such an error rate, the threshold theorem says the last ingredient required for reliable quantum computations is more qubits at the same error rate.

While the polylogarithmic scaling in overhead implied by the threshold theorem seems modest, in practice - outside an asymptotic limit, the overhead will not necessarily be as favorable: the big-O generally conceals significant constant factors. For low-threshold protocols, these constant factors can be on the order of hundreds, while in protocols with optimized thresholds, these constant factors reach billions [8, 13]; it is also worth pointing out explicitly that fault-tolerant protocols with higher thresholds tend to also have larger constant factors in the overhead. However, this is still an active field of study [11].

While our presentation of the threshold theorem relies on concatenated codes that encode a single logical qubit, it does not need to. In fact, there can be advantages to considering families of codes that encode multiple logical qubits per encoded block [7, 13]. *Constant-rate* codes provide a remarkable illustration of this point. A code's *rate* [1] is defined as the ratio between logical k and physical n qubits in the code, so a constant rate code has a non-zero rate in the asymptotic limit of infinitely many physical qubits. By exploiting constant-rate *low-density parity check* (LDPC) *codes* (which should really be called constant-rate LDPC code families) - characterized by having sparse stabilizer checks and satisfying some additional properties [13], it has been shown that - using the same assumptions as in Subsection 3.1.1 - there exists a fault-tolerant scheme that provides a threshold result at a qubit cost that is only a (potentially very small) constant factor bigger than the ideal circuit [13]. The insight gained here is that the polylog cost in qubit overhead for fault-tolerance is due to codes with a vanishing rate [13, 24]. Note that a lower bound on the number of qubits necessary for fault tolerance [9] implies that fault-tolerant protocols with constant qubit overhead are possible only on *families of quantum circuits* - defining a quantum algorithm we want to run - with a depth that grows sub-exponentially in qubit count. Furthermore, this construction heavily relies on reused qubits, making it incompatible with a low depth - potentially necessitating worse thresholds, particularly in a regime with high memory error rates. Conversely, it has also been shown that thresholds exist for fault-tolerant protocols where instead the depth scales by a (potentially small) constant factor [13].

The asymptotically low qubit overhead afforded by this construction is appealing for it potentially makes fault tolerance cheaper in non-asymptotic circuits, helping address the limited qubit count expected in nearer-term quantum devices. However, it relies on the assumption of free classical computing: Subsection 3.1.1. To relax this unrealistic assumption, this construction uses a subclass of constant-rate LDPC codes called *expander codes*, which are equipped with a decoding algorithm that is sufficiently fast (constant time-complexity, in fact) and corrects enough errors for a (non-zero) threshold [24]. However, LDPC codes - and expander codes by extension - have drawbacks. They often have complicated logical operators and are non-local - making logical gates and syndrome extraction more physically challenging [13]. In fact, on this last point, an LDPC code embedded in finite dimensions cannot be local while simultaneously maintaining a constant rate with suitable distance scaling [8, 40, 41]. This drawback has led to the non-locality of LDPC codes being subject to much study and places an upper bound on the effectiveness of (LDPC) codes implementable on connectivity-limited hardware, currently limiting the practicality of this idea.

3.3 Computing Thresholds

In this section, I aim to make thresholds more concrete: we will discuss how to calculate thresholds, the values of thresholds in practice, and how thresholds 'look'.

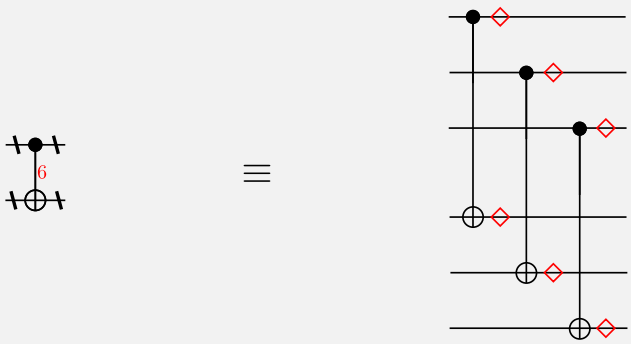
Working again in the framework of concatenation-based fault tolerance, we can calculate a scheme's threshold analytically by counting the number of potential fault locations in the largest level-1 unit cell, per Lemma 1. To get a taste of this, consider an example on the 3-qubit repetition code.

Example 7 (Threshold of the 3-Qubit Repetition Code)**.** Assume here an error model where single-qubit gates have a single potential fault location that acts on the same qubit as the gate and that two-qubit gates have two independent potential fault locations, one on each of the qubits acted on. Because this example is purely pedagogical, we will simplify matters by further imposing the assumptions that only X errors occur, that measurements are fault-free, and memory errors are negligible.

Consider the unit cell of a $CNOT$ on the 3-qubit repetition code. As is typical, the unit cell mapping requires the $CNOT$ to be replaced with the code's logical $CNOT$, pre- and post-luded by error correction on each encoded block. That is, (ignoring any part of the diagrams in red for now)

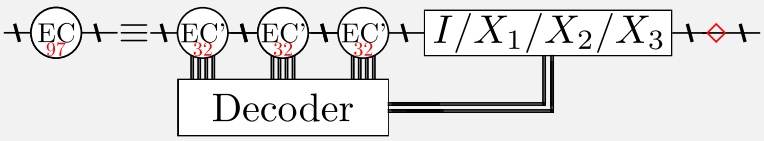
(3.F)

where the logical $CNOT$ is



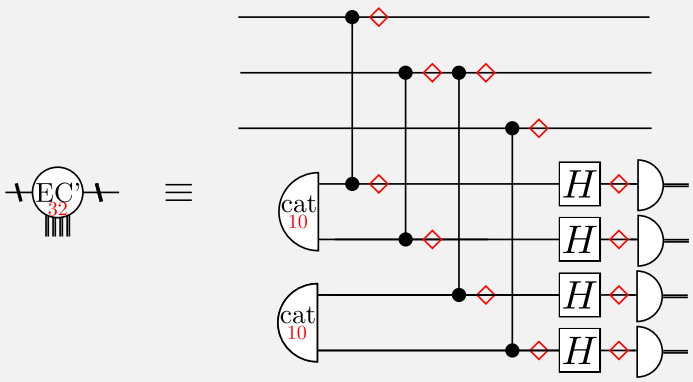
(3.G)

and said to be transversal. In this construction, we use *Shor syndrome extraction* [42], which breaks fault-tolerant error correction into steps



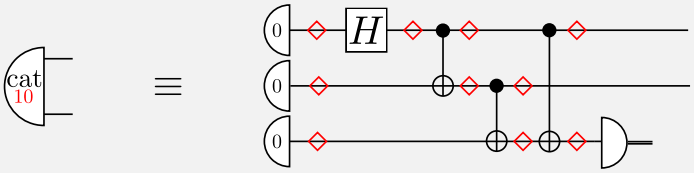
(3.H)

where the $I/X_1/X_2/X_3$ gate applies any one of these four operations conditioned on the decoder's output, and each of these sub-steps is defined as



(3.I)

and where the gates with two black dots - say on qubits i and j - connected by a vertical line are $CZ_{i,j} \equiv |0\rangle_i \langle 0|_i \otimes I_j + |1\rangle_i \langle 1|_i \otimes Z_j$ with implicit identities on other qubits. These sub-error-correction-steps rely on the fault-tolerant preparation of *cat states*, $|\text{cat}\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$, which we generate using the circuit below.



(3.J)

Error correction is broken into these sub-steps to verify the syndrome before correcting. While - for brevity - I will not explain in more detail how this construction is fault-tolerant, a reader can check that indeed this procedure will not fail for any single-qubit X error, the degree of error correction that this code is rated for.

Turning our attention back to the threshold, we can see in the above diagrams each potential fault location marked by a red diamond. Also, in red on each gate is the number of potential fault locations contained in that process. Counting, we find 394 potential fault locations in the unit cell. If we pick errors to occur at any two of these fault locations, the procedure does not necessarily succeed because it can only correct with certainty a single (X) error. So an upper bound on the number of ways this procedure can fail is $\frac{394 \times 393}{2} = 77421$, where we divide by 2 to avoid double counting. Therefore, from Lemma 1, the threshold is $P_{th} \geq \frac{1}{77421} \sim 1.3 \times 10^{-5}$ - assuming that a $CNOT$ has the most potential fault locations out of any unit cell we want to use in this error model. And so, as long as the error rate per potential fault location is below this threshold, scaling the construction to higher levels l of concatenation will yield arbitrarily accurate simulations.

While illustrative, this example's threshold relies on unrealistic assumptions about the nature of errors. Relaxing these assumptions and instead using a *surface code* [43] - a family of LDPC codes (so we are no longer relying on concatenation to scale our code) made popular for practical implementation by its nearest-neighbor locality in a two-dimensional Euclidean geometry and giving high thresholds - achieves a threshold of $\sim 6 \times 10^{-3}$ under a noise model where X, Y, and Z errors are all equally likely [2]. (Stressing the assumptions of a model is critical when presenting its threshold. The procedure achieving this threshold relies on assumptions similar to Subsection 3.1.1. Differences are: the scheme only requires limited long-range interactions and parallel gates, though it uses many parallel measurements. And the free-classical-computations assumption can be partially relaxed because, while the scheme uses a time-efficient algorithm, it assumes that the decoder and any other supporting classical computations - for which there can be many in this *measurement-based*

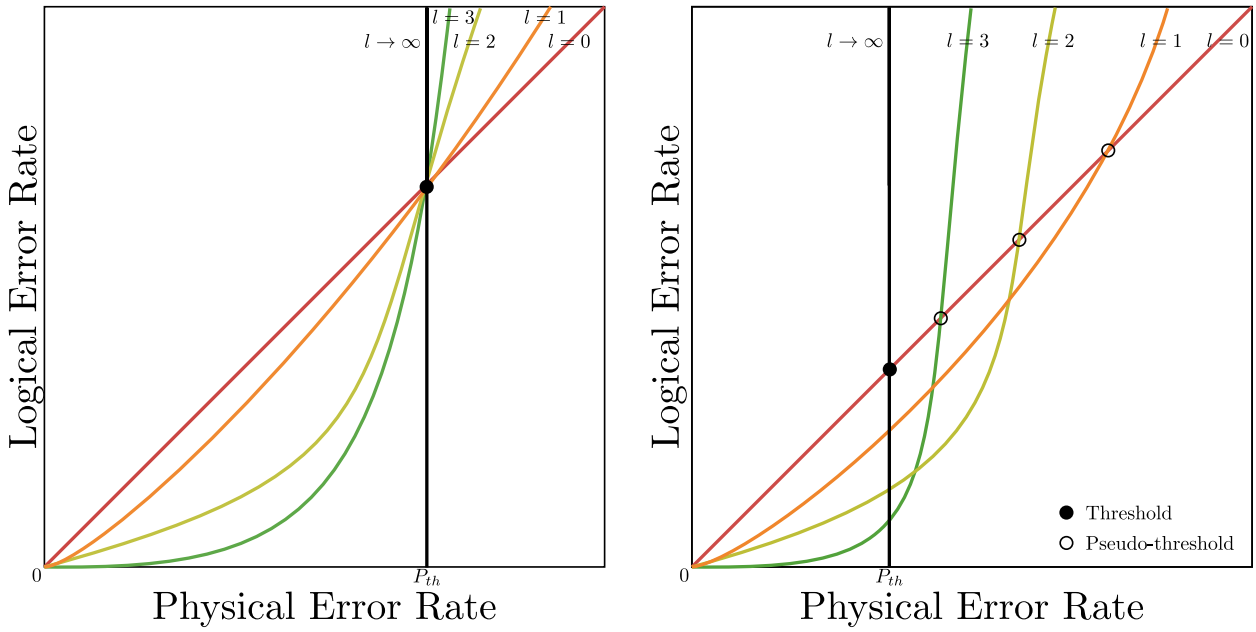


Figure 3.3: Numerical thresholds. Adapted from [29]. The left graph corresponds to a generally contrived scenario where the threshold is the physical error rate for which all levels l of fault tolerance give the same logical error rate. Increasing the level while below threshold suppresses errors while the opposite is true above-threshold. The right shows a more realistic pseudo-threshold regime.

[25] approach - take no time.) While error rates are approaching this threshold, the surface code (or any other code) is yet to be seen on the scale needed for useful quantum computing [30].

Our example also highlights the difficulty of analytically computing threshold bounds. This exercise in counting is not always practical, particularly for the more sophisticated constructions we are usually interested in for implementations. Instead, we often elect to estimate thresholds with numerical simulations. In short, this is done by programming in a classical computer the desired error correction procedure and error model, accordingly applying errors using *Monte Carlo methods*, and then checking if the error correction procedure succeeded in fixing the error. Over many iterations at select error rates, we find an estimate of the error correction procedure's (logical) failure rate; using *statistical standard error*, achieving an uncertainty $0 < \delta \leq 1$ requires $\sim \delta^{-2}$ iterations. By repeating this over different physical error rates, one can plot a graph of logical error rate vs physical error rate. If the failure probability in Equation 3.4 is an equality, repeating this process for an additional fault-tolerance level l yields an estimate of the threshold: the physical error rate at which different levels fail with the same probability, illustrated on the left of Figure 3.3. In a general and more physical case, the failure probability of our construction in Equation 3.4 (or from other constructions) remains an inequality due to asimilarities that shift the proportion of malignant errors in different fault-tolerant levels l; for example, from more complicated error models - such as errors occurring with distinct probabilities for particular operations, from elaborate rules for scaling the fault-tolerant procedure - like non-self-similar gadgets, or particular decoders. In this case, instead, *pseudo-thresholds* arise: the physical error rates where level-$l \geq 1$ curves intersect the level-0 curve, shown on the right side of Figure 3.3. While being below the level-w pseudo-threshold means level-w fault tolerance offers protection, unlike the threshold, it does not guarantee that scaling the level l will be advantageous. To find the actual threshold, one must examine the asymptotic level of concatenation, where the pseudo-thresholds converge to the true threshold. Using Pauli

errors and Clifford circuits, as we often can in error correction [6], these simulations are time-efficient due to the *Gottesman-Knill theorem* [44, 45]; however, more general circuits and error models can make these simulations intractable.

Finally, to elaborate on the practical cost of useful and robust quantum computations, it is estimated that breaking modern encryption protocols using a quantum computer, currently an intractable problem on classical computers, will require 8 hours and $\sim 2 \times 10^7$ qubits with below-threshold error rates [46]. On this, fault-tolerant protocols' overheads can be sensitive to how far below threshold the error rate is. This is important for practical purposes given the current and likely continued expense of qubits. For instance, in a large surface code, an error rate of an order of magnitude below threshold can reduce the qubit overhead also by an order of magnitude [47].

4

Conclusion

I aimed to show in this essay that due to the threshold theorem, fault-tolerant methods provide a path forward on the mission of building noise-robust quantum computers. Indeed, this essay introduces fault tolerance as a way of mapping some ideal circuit that performs some desired computation to a new simulating circuit that allows for any desired level of accuracy in a noisy environment; presents a fault-tolerant construction that leads to a version of the threshold theorem; and discusses the threshold theorem and its implications.

There is still much work to be done on fault tolerance. The main two goals for experimentalists should unsurprisingly be to achieve error rates (well) below threshold and add more qubits to quantum devices while retaining sub-threshold error rates. A more specific research direction for experimentalists is to comprehensively investigate if the threshold theorem's assumptions are indeed satisfied in real devices and study quantum noise more generally. Neither of these have received as much attention as they deserve, likely because it is difficult to do so on our currently very limited quantum hardware. However, they are nonetheless important. The first of these two suggestions is important because the future of quantum computing is believed to rely on the threshold theorem, which itself relies critically on these assumptions. The second is important because higher threshold protocols have been found using insights into quantum noise's properties [28].

Regarding future research for theorists, the aim is to design protocols that increase thresholds and reduce overheads. I will briefly discuss some of the specific frontiers of fault-tolerance research that I think are most exciting. One such direction is to try and consolidate the seeming trade-off between depth and qubit count in fault-tolerant protocols, discussed in [13]. This would involve designing a fault-tolerant protocol that simultaneously achieves (near-)constant qubit count and depth scaling. I believe a starting point for this work includes *measurement-* [25] or *fusion-based* [16] quantum computing, as these models naturally have less of a distinction between space and time [48]. Moving on, because LDPC codes have shown promise in efficient fault tolerance, it would also make sense to try and address their biggest drawback: their non-locality; for example, one may consider how to layout qubits in a two-dimensional architecture to minimize the long-range interactions between qubits. Work has recently begun towards this goal [49, 50]. Additionally, it is believed that LDPC codes admit more efficient fault-tolerant logical gates than they currently have, which could also be the subject of further research [8]. Work could also focus

on optimizing fault-tolerance's overhead for specific hardware. There is, for example, a slew of work in this area built around the aforementioned fusion-based model on *photonic devices* [16]. Another emerging idea, as realized by *flag qubits* [51, 52], is that instead of passively preventing error propagation, fault-tolerant protocols can consider where errors occur in the circuit in both time and space. For example, a naive non-fault-tolerant method of syndrome extraction combined with flag qubits is fault tolerant as the flag qubits allow us to figure out how the errors probably propagated during syndrome extraction and actively correct them accordingly [51, 52]. Perhaps this idea can be applied to potentially reduce the cost of other fault-tolerant procedures too - maybe by following the ideas proposed in [8]. In particular, it would be interesting to see if flag qubits could help make the efficient non-fault-tolerant encoding circuits I proposed in [6] fault tolerant; to achieve lower-overhead encoding.

On the outlook of fault-tolerant quantum computing, I have hopefully conveyed in this essay the immense cost of fault tolerance and the relatively low (but not impossibly low) thresholds it achieves; these results are despite many simplifying assumptions, and fault tolerance's outlook with these assumptions relaxed is yet starker. Large-scale fault-tolerant quantum computation is surely many years away, $\sim O(10)$ years in my view. That said, recent results [30] suggest that we may be on the cusp of small-scale fault tolerance. I believe these early implementations of fault tolerance will dramatically accelerate fault-tolerance theory research by giving insights into errors that can be exploited, as we have already seen in [28]. That is, I conjecture that while full-scale fault tolerance may be decades away, fault-tolerant quantum computing will face its most pivotal period over the next few years.

References

[1] M. Nielsen and I. Chuang. Quantum Computation and Quantum Information. *Cambridge University Press*, 2000.

[2] A. Fowler, A. Stephens, and P. Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80(5), 2009.

[3] S. Debnath, N. Linke, C. Figgatt, K. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016.

[4] E. Deist, Y. Lu, J. Ho, M. Pasha, J. Zeiher, Z. Yan, and D. Stamper-Kurn. Mid-Circuit Cavity Measurement in a Neutral Atom Array. *Physical Review Letters*, 129:203602, 2022.

[5] D. Gottesman. Stabilizer Codes and Quantum Error Correction. *Caltech (PhD Thesis)*, 1997.

[6] M. Gorman. Ancilla-free preparation of quantum error-correcting codes. *The University of Queensland (Honours Thesis - preparing for publication)*, 2022.

[7] E. Knill, R. Laflamme, and W. Zurek. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):365–384, 1998.

[8] D. Gottesman. Opportunities and Challenges in Fault-Tolerant Quantum Computation. *arXiv:2210.15844*, 2022.

[9] O. Fawzi, A. Müller-Hermes, and A. Shayeghi. A Lower Bound on the Space Overhead of Fault-Tolerant Quantum Computation. *Schloss Dagstuhl - Leibniz-Zentrum für Informatik*, 2022.

[10] P. Aliferis, D. Gottesman, and J. Preskill. Quantum Accuracy Threshold for Concatenated Distance-3 Codes. *Quantum Information and Computation*, 6(2):97–165, 2006.

[11] D. Gottesman. An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation. *arXiv:0904.2557*, 2009.

[12] C. Figgatt, A. Ostrander, N. Linke, K. Landsman, D. Zhu, D. Maslov, and C. Monroe. Parallel entangling operations on a universal ion-trap quantum computer. *Nature*, 572(7769):368–372, 2019.

[13] Daniel Gottesman. Fault-Tolerant Quantum Computation with Constant Overhead. *Quantum Information and Computation*, 14(15–16):1338–1372, 2014.

[14] A. Noiri, K. Takeda, T. Nakajima, T. Kobayashi, A. Sammak, G. Scappucci, and S. Tarucha. A shuttling-based two-qubit logic gate for linking distant silicon quantum processors. *Nature Communications*, 13(1), 2022.

[15] D. Aharonov, M. Ben-Or, R. Impagliazzo, and N. Nisan. Limitations of Noisy Reversible Computation. *arXiv:9611028*, 1996.

[16] S. Bartolucci, P. Birchall, H. Bombin, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant, F. Pastawski, T. Rudolph, and C. Sparrow. Fusion-based quantum computation. *Nature Communications*, 2023.

[17] M. DeCross, E. Chertkov, M. Kohagen, and M. Foss-Feig. Qubit-reuse compilation with mid-circuit measurement and reset. *arXiv:2210.08039*, 2022.

[18] B. Barber, N. Gillespie, and J. Taylor. Post-selection-free preparation of high-quality physical qubits. *arXiv:2209.05391*, 2022.

[19] D. DiVincenzo and P. Aliferis. Effective Fault-Tolerant Quantum Computation with Slow Measurements. *Physical Review Letters*, 98:020501, 2007.

[20] D. Aharonov and M. Ben-Or. Fault-Tolerant Quantum Computation with Constant Error. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, page 176–188, 1997.

[21] T. Szkopek, V. Roychowdhury, D. Antoniadis, and J. Damoulakis. Physical Fault Tolerance of Nanoelectronics. *Physical Review Letters*, 106:176801, 2011.

[22] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009.

[23] N. Delfosse and N. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, 2021.

[24] O. Fawzi, A. Grospellier, and A. Leverrier. Constant Overhead Quantum Fault Tolerance with Quantum Expander Codes. *Communications of the ACM*, 64(1):106–114, 2020.

[25] H. Briegel, D. Browne, W. Dür, R. Raussendorf, and M. Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

[26] E. Knill. Scalable quantum computing in the presence of large detected-error rates. *Physical Review A*, 71:042322, 2005.

[27] T. Tansuwannont and K. Brown. Adaptive syndrome measurements for Shor-style error correction. *arXiv:2208.05601*, 2022.

[28] J. Ataides, D. Tuckett, S. Bartlett, S. Flammia, and B. Brown. The XZZX surface code. *Nature Communications*, 12(1), 2021.

[29] K. Svore, A. Cross, I. Chuang, and A. Aho. A Flow-Map Model for Analyzing Pseudothresholds in Fault-Tolerant Quantum Computing. *Quantum Information and Computation*, 6(3):193–212, 2006.

[30] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614:676–681, 2023.

[31] F. Pollock L. Hollenberg K. Modi G. White, C. Hill. Demonstration of non-Markovian process characterisation and control on a quantum processor. *Nature Communications*, 11(6301), 2020.

[32] J. Morris, F. Pollock, and K. Modi. Quantifying non-Markovian Memory in a Superconducting Quantum Computer. *Open Systems and Information Dynamics*, 29(02), 2022.

[33] B. Terhal and G. Burkard. Fault-tolerant quantum computation for local non-Markovian noise. *Physical Review A*, 71:012336, 2005.

[34] N. Brown, A. Cross, and K. Brown. Critical faults of leakage errors on the surface code. *2020 IEEE International Conference on Quantum Computing and Engineering*, pages 286–294, 2020.

[35] P. Aliferis and B. Terhal. Fault-Tolerant Quantum Computation for Local Leakage Faults. *Quantum Information and Computation*, 7(1):139–156, 2007.

[36] M. Gorman and T. Farrelly. A review of syndrome extraction in quantum error correction. *The University of Queensland (preparing for publication)*, 2022.

[37] B. Eastin and E. Knill. Restrictions on Transversal Encoded Quantum Gate Sets. *Physical Review Letters*, 102(11), 2009.

[38] T. Jochym-O'Connor and R. Laflamme. Using Concatenated Quantum Codes for Universal Fault-Tolerant Quantum Gates. *Physical Review Letters*, 112(1), 2014.

[39] J. Preskill. Quantum Computation. *California Institute of Technology (notes)*, 2021.

[40] S. Bravyi, D. Poulin, and B. Terhal. Tradeoffs for Reliable Quantum Information Storage in 2D Systems. *Physical Review Letters*, 104(5), 2010.

[41] N. Baspin and A. Krishna. Quantifying Nonlocality: How Outperforming Local Quantum Codes Is Expensive. *Physical Review Letters*, 129(5), 2022.

[42] P. Shor. Fault-tolerant quantum computation. *arXiv:9605011*, 1997.

[43] A. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[44] D. Gottesman. The Heisenberg Representation of Quantum Computers. *arXiv:9807006*, 1998.

[45] S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), 2004.

[46] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[47] A. Fowler, M. Mariantoni, J. Martinis, and A. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86:032324, 2012.

[48] N. Nickerson and H. Bombín. Measurement based fault tolerance beyond foliation. *arXiv:1810.09621*, 2018.

[49] N. Berthusen and D. Gottesman. work in progress. 2023.

[50] C. Pattison, A. Krishna, and J. Preskill. Hierarchical memories: Simulating quantum LDPC codes with local gates. *arXiv:2303.04798*, 2023.

[51] R. Chao and B. Reichardt. Quantum Error Correction with Only Two Extra Qubits. *Physical Review Letters*, 121:050502, 2018.

[52] R. Chao and B. Reichardt. Flag Fault-Tolerant Error Correction for any Stabilizer Code. *PRX Quantum*, 1:010302, 2020.