Master's Thesis

Emma Storberg

February 23, 2026

Contents

1 **Introduction** **1**

2 **Background** **3**

- 2.1 Grover's Algorithm for Solving Unstructured Search 3
  - 2.1.1 Practical Application and Limitations of Grover's Algorithm 8
- 2.2 Quantum Error Correction for Stable Quantum Memory 9
  - 2.2.1 Toric Surface Code 9
  - 2.2.2 Alternatives to Toric Surface Code 12
- 2.3 Universal Gate Sets for Quantum Computation 13
  - 2.3.1 Choices to Make When Performing Quantum Computations 16
- 2.4 Phase Query Gates in Error Correcting Architecture 17

3 **Methods** **18**

  - 3.0.1 Error Models 19
  - 3.0.2 Software Aids 19
- 3.1 Coding Fault-tolerant Quantum Memory 19
  - 3.1.1 Surface Code 19
  - 3.1.2 BB Code 19
- 3.2 Applying Gates to Error-correcting Patches 20

4 **Results** **21**

5 Conclusion 22

5.1 What is Jacques' full line of argumentation? 22

5.2 Future Work 23

5.3 Declaration of AI Usage 23

Chapter 1

Introduction

In recent years, quantum algorithms have garnered significant attention for their potential to disrupt traditional cryptographic systems. The security of many important communication channels, including the internet as we know it, is ensured by classical cryptographic protocols, particularly those based on the hardness of certain mathematical problems. With the rise of quantum technology, new algorithms have been found that exploit quantum mechanical principles, such as superposition and entanglement, to solve these problems exponentially faster than classical algorithms. Still, the implementation of quantum algorithms for cryptographic purposes continues to be largely theoretical. A quantum computer capable of breaking the most-used algorithms today is yet to be constructed, as there are many practical limitations to doing so. Thus, practical quantum computers (with thousands of qubits and low error rates) remain an area of active research; there is considerable anticipation surrounding the eventual realization of the theoretical promises associated with quantum computing. As we move into the post-quantum era, it will become increasingly important to assess our cryptographic protocols in light of the most recent developments in the field, in order to maintain and protect critical infrastructure and assets.

Brute-force searches are a technique that may be employed to break cryptographic systems that use so-called *symmetric* keys[1]. Performing such a search involves quite simply checking every single possible key option until the right key is found, an inordinately time-consuming process if the key space is large enough. Our current protocols ensure security against these searches by having key lengths large enough that it is considered computationally infeasible for a classical computer to identify one in any reasonable amount of time. A closer look at the popular symmetric-key algorithm *Advanced Encryption Standard* (AES) [2] exemplifies this: It exists in versions that use key sizes of 128, 192 and 256 bits, resulting in search spaces of 2^{128}, 2^{192} and 2^{256} possible keys respectively.

A relevant quantum algorithm here is *Grover's algorithm* [3], in which a procedure is given for a quantum computer to do a brute-force search much faster than classical computers, thereby

---

[1] Symmetric-key algorithms use the same keys for encryption and decryption [1], as opposed to *asymmetric* algorithms, which use separate encryption and decryption keys. This project will focus on the former.

potentially threatening the security of AES. Its asymptotic runtime (meaning the approximate number of operations needed for the algorithm to run as a function of the decryption key length N) is $\mathcal{O}(\sqrt{N})$, compared to the classical approach with $\mathcal{O}(N)$ [4]. However, finding the exact runtime is not as straightforward as the asymptotic estimates, which do not consider constant values, nor do they take into account what kind of parallelization may be possible. Analyzing the true cost of using Grover's algorithm in reality is a nontrivial problem that warrants further exploration, which constitutes the core of this project.

Building a quantum machine in practice is no easy feat. The root of the issue is that quantum information is extremely fragile. The qubits that store information are unstable and very vulnerable to noise that changes the value the qubit holds. We therefore wish to construct circuits of multiple qubits that work together to protect the information from errors, such that they form a single logical qubit that can have its errors located and corrected, making them more stable. This approach, though useful, will often require large numbers of qubits, which poses an engineering challenge to keep them all as stable as possible, facilitate connections between them, and perform computations on the logical qubits, rather than on the physical qubits individually.

Samuel Jacques considers this in his 2024 analysis of Grover's algorithm as a quantum attack on AES [5], where he suggests that due to the error correction models and gate sets currently in widespread use, a physical implementation of Grover's algorithm will never see the light of day on anything resembling current quantum architectures (or maybe ever). This project uses his argumentation as a starting point for further analysis, aiming to tackle some of the assumptions he makes and explore alternatives to some of the runtime bottlenecks he identifies.

Do I need more context about him and his credibility?

I am planning to add a little more info towards the end here about my experiments and method as soon as I make a final decision on the direction I want to take this. It will probably include discussions of other error correcting codes like *BB codes* or *tile codes*. As in, "since Jacques' argumentation bases itself so much on our continued usage of the surface code, what does his analysis look like under the assumption of a better error correcting code?"

Chapter 2

Background

Section 2.1 is adapted from parts of my previous work [6], with revisions and additions for clarity.

2.1 Grover's Algorithm for Solving Unstructured Search

Formally, the computational problem that Grover's algorithm addresses is a quite general problem known as *unstructured search* [7]. It can be described as follows:

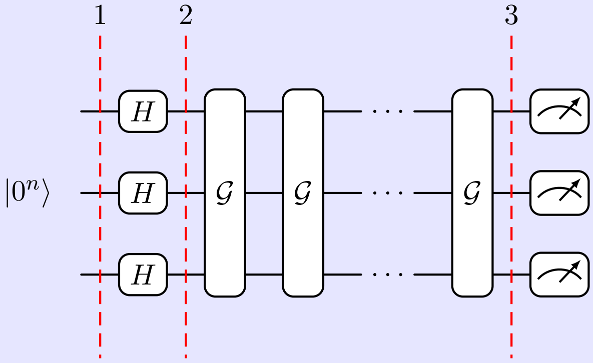
The unstructured search problem

> Let Σ denote the binary alphabet $\{0, 1\}$. Suppose we are given a function $f : \Sigma^n \rightarrow \Sigma$ that we can compute efficiently. We wish to determine the binary string $x \in \Sigma^n$ such that $f(x) = 1$, or "no solution" if no such x exists.

We call this search problem *unstructured* because f can be an arbitrary function, meaning there is no underlying pattern or *promise* [7] that we can take advantage of to systematically search for the solution string x, if it even exists. This means that in the classical approach to solving this problem, we can be sure of the solution only after checking every possible string (of which there are 2^n), thus requiring 2^n queries. As we will see, Grover's algorithm offers a quadratic improvement over classical algorithms, meaning the number of operations needed when we use this quantum algorithm is on the order of the square-root of the number of operations required when solving the same problem classically.

Circuit diagram of Grover's algorithm

The algorithm essentially consists of three main parts: we have initialization of n qubits into an equal superposition (1), followed by t Grover iterates (2), and finally measurement to reveal a candidate solution (3). These three stages are displayed and labeled in the circuit diagram below:



Let's dive into the algorithm. Our first step is using the *Hadamard gate* (denoted by H) [8] on each of the n input qubits. H has no classical equivalent, and using it allows us to take advantage of a strictly quantum phenomenon known as *superposition* [9]. This means the qubit can evaluate to either 0 or 1, and we can apply processing to it as if the qubit holds both values simultaneously. Thus, in finding a binary string of length n, using the Hadamard gate on $|0^n\rangle$ lets us evaluate all 2^n strings in Σ^n concurrently. For convenience, we will use the letter N to refer to the value 2^n from this point on.

Next is the Grover iteration step, represented by sequential application of the Grover operator $\mathcal{G}$ in the diagram. Its definition and circuit diagram are explained in detail on the next page.

Should I reference this more formally?

The Grover operation $\mathcal{G}$

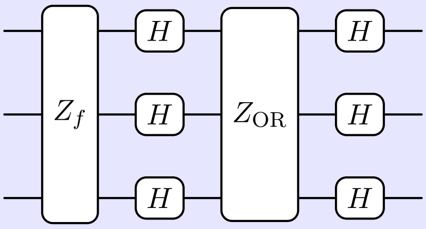
The gate sequence needed for the Grover operation $\mathcal{G}$ is:

$$H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f$$

where Z_f is the phase query gate for the function f, and Z_{OR} is the phase query gate for the n-bit logical OR function. They are defined as follows:

$$Z_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle \qquad \forall x \in \Sigma^n$$
$$Z_{\mathrm{OR}} : \begin{cases} |x\rangle & x = 0^n \qquad \forall x \in \Sigma^n \\ -|x\rangle & x \neq 0^n \qquad \forall x \in \Sigma^n \end{cases}$$

Since the gates are applied from right to left, we apply Z_f first, followed by H, then Z_{OR}, and finally another set of Hadamard gates. The circuit diagram of the Grover operation on three bits looks like this:



Concerning evaluation of the circuit, only the Z_f gate requires a query to f. Thus, the number of iterations t we perform with $\mathcal{G}$ corresponds to how many queries to f we need.

A natural way to use the algorithm for search problems is to choose a number of iterations t of the Grover operation, run the circuit to obtain a candidate solution x, and check if x is a solution (in this case whether $f(x) = 1$). If so, we return x, otherwise we run the algorithm again (possibly with a different value for t) or return "no solution". This sounds simple enough, but still, important questions remain unanswered: What is the "suitable" number of operations t, how do we identify it, and why must it be chosen so precisely?

To understand what is happening here, it is helpful to separate the possible solution strings in Σ^n into two categories, which we will call solutions and nonsolutions. These sets can be described as follows:

$$A_0 = \{x \in \Sigma^n : f(x) = 0\}$$
$$A_1 = \{x \in \Sigma^n : f(x) = 1\}$$

where A_0 contains all the strings that evaluate to 0 when given as input to f, and A_1 has the strings we seek that evaluate to 1. In particular, we are interested in uniform superpositions over these two sets[1], defined as:

$$|A_0\rangle = \frac{1}{\sqrt{|A_0|}} \sum_{x \in A_0} |x\rangle$$
$$|A_1\rangle = \frac{1}{\sqrt{|A_1|}} \sum_{x \in A_1} |x\rangle$$

An important observation here is the fact that we can write the initialized uniform superposition as a linear combination of the states $|A_0\rangle$ and $|A_1\rangle$:

$$H^{\otimes n} |0^n\rangle = \sqrt{\frac{|A_0|}{N}} |A_0\rangle + \sqrt{\frac{|A_1|}{N}} |A_1\rangle$$

The key idea we utilize is this: **The state of the qubit register remains in the subspace spanned by $|A_0\rangle$ and $|A_1\rangle$ after every application of $\mathcal{G}$.** Thus, we would like to process the initialized state $H^{\otimes n} |0^n\rangle$ (with uniform amplitude across all N components) such that through successive iterations, we nudge the amplitude of the solution component $|A_1\rangle$ slightly higher, while lowering the amplitude of the nonsolutions $|A_0\rangle$. As we know, in a quantum state vector, the amplitude of any component squared represents the probability of seeing that component after measurement. Provided we find the suitable number of iterations t, we can push the amplitude of the state corresponding to the solutions arbitrarily high, and consequently see a high probability that this state is measured.

Qiskit [7] calculates the application of $\mathcal{G}$ on the $|A_0\rangle$ and $|A_1\rangle$, resulting in the following linear combinations:

$$\mathcal{G} |A_0\rangle = \frac{|A_0| - |A_1|}{N} |A_0\rangle + \frac{2\sqrt{|A_0| \cdot |A_1|}}{N} |A_1\rangle$$
$$\mathcal{G} |A_1\rangle = -\frac{2\sqrt{|A_0| \cdot |A_1|}}{N} |A_0\rangle + \frac{|A_0| - |A_1|}{N} |A_1\rangle$$

This clearly demonstrates that we are still in the span of $|A_0\rangle$ and $|A_1\rangle$ after an application of $\mathcal{G}$. Moreover, we can describe the action of $\mathcal{G}$ as a rotation matrix, and by once again following the example of Qiskit [7], we can also rewrite it as the square of a somewhat simpler-looking matrix:

$$\begin{bmatrix} \frac{|A_0|-|A_1|}{N} & -\frac{2\sqrt{|A_0|\cdot|A_1|}}{N} \\ \frac{2\sqrt{|A_0|\cdot|A_1|}}{N} & \frac{|A_0|-|A_1|}{N} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{|A_0|}{N}} & -\sqrt{\frac{|A_1|}{N}} \\ \sqrt{\frac{|A_1|}{N}} & \sqrt{\frac{|A_0|}{N}} \end{bmatrix}^2$$

Now it is easier to see the similarity to the general form of a rotation matrix, with $\theta = \sin^{-1}\left(\sqrt{\frac{|A_1|}{N}}\right)$:

$$\begin{bmatrix} \sqrt{\frac{|A_0|}{N}} & -\sqrt{\frac{|A_1|}{N}} \\ \sqrt{\frac{|A_1|}{N}} & \sqrt{\frac{|A_0|}{N}} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

---

[1] Importantly, we will assume from here on that A_0 and A_1 are both nonempty. However, we can still easily see what happens in the last two cases (all strings evaluate to 0, or all strings are solutions), as the former will return no solution, and in the latter case, any string is a solution. Since we are able to handle these special cases, we can safely continue our analysis with this assumption.

Since squaring a rotation is the same thing as applying it twice, we return to the original rotation matrix for $\mathcal{G}$, and write it as:

$$\begin{bmatrix} \frac{|A_0|-|A_1|}{N} & -\frac{2\sqrt{|A_0|\cdot|A_1|}}{N} \\ \frac{2\sqrt{|A_0|\cdot|A_1|}}{N} & \frac{|A_0|-|A_1|}{N} \end{bmatrix} = \begin{bmatrix} \cos 2\theta & -\sin 2\theta \\ \sin 2\theta & \cos 2\theta \end{bmatrix}$$

with θ defined as before. Thus, we have found a rotation matrix that describes the action of $\mathcal{G}$ on the subspace spanned by A_0 and A_1. This also means that the state of the qubit register after initialization can be described as:

$$H^{\otimes n}|0^n\rangle = \sqrt{\frac{|A_0|}{N}}|A_0\rangle + \sqrt{\frac{|A_1|}{N}}|A_1\rangle = \cos\theta|A_0\rangle + \sin\theta|A_1\rangle$$

So, each time we apply $\mathcal{G}$, the state of the register is rotated by 2θ, and we can find a closed-form expression for the state after t applications of $\mathcal{G}$. For brevity, we will henceforth refer to the initialized qubit register $H^{\otimes n}|0^n\rangle$ as $|u\rangle$.

$$\mathcal{G}^t|u\rangle = \cos((2t+1)\theta)|A_0\rangle + \sin((2t+1)\theta)|A_1\rangle$$

This expression makes sense intuitively: 2θ for each application of $\mathcal{G}$, and one additional θ to account for the state we started in.

Now recall that for any quantum state of the form $\alpha|A_0\rangle + \beta|A_1\rangle$, we measure a solution $x \in A_1$ with probability $|\beta|^2$. With the state in question $\mathcal{G}^t|u\rangle = \cos((2t+1)\theta)|A_0\rangle + \sin((2t+1)\theta)|A_1\rangle$, the probability of measuring a solution after t iterations is $\sin^2((2t+1)\theta)$. This is the probability we want to maximize in order to find a solution to the search problem with Grover's algorithm, and as such, we call $|A_1\rangle$ our *target state* [7].

Another consideration is that we would also like to minimize t, as this is the number of queries to f we have to make. Thus, to reach our target state with a probability close to 1 while minimizing t, we aim to find t such that $2(t+1)\theta \approx \frac{\pi}{2}$, since this is the smallest angle where the square of the sine function equals 1. In terms of t, we aim for $t \approx \frac{\pi}{4\theta} - \frac{1}{2}$. We cannot guarantee that this value of t is an integer, so we find the number of iterations t by choosing $t = \lfloor \frac{\pi}{4\theta} \rfloor$. For the algorithm to work properly, it is also important that we do not iterate past the recommended iteration number, lest we avoid "overshooting" our solution by rotating past it. If we simply continue applying $\mathcal{G}$, we will in fact decrease the probability that the solution state is measured, and over time, we will oscillate between solutions and nonsolutions as we rotate the register state around the unit circle [7].

Up to this point, we have studied a general version of Grover's algorithm that can be applied to a number of search problems, ranging from those we know to have a single solution, a larger solution set of a fixed size, or even an unknown number of solutions. We have determined that an iteration number of $t = \lfloor \frac{\pi}{4\theta} \rfloor$ (and no more) will maximize the probability of finding one of these solutions, but what we must also consider is the fact that this way of setting iteration number t also depends on how many solutions there are, as θ depends on the size of the set A_1. Fortunately, since we are considering only the cryptographic application of using the algorithm to search for a single key from some symmetric cryptographic protocol, we can restrict ourselves to the cases where we know there is exactly one solution (or one cryptographic key).

Unique search, as this problem is formally known, is equivalent to the general unstructured search problem, with the additional promise that there is exactly one solution string x. Recall that in the general case, $\theta = \sin^{-1}\left(\sqrt{\frac{|A_1|}{N}}\right)$. Now that we have determined there is only a single solution, we know that $|A_1| = 1$. Thus, for large N, we see that $\theta = \sin^{-1}\left(\sqrt{\frac{1}{N}}\right) \approx \sqrt{\frac{1}{N}}$, since for smaller angles, the sine function starts to look more like the identity function [7]. If we use this approximation of θ in our previously determined expression $t = \lfloor \frac{\pi}{4\theta} \rfloor$, we get $t = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$. Since t represents queries to the function f, we see the quadratic improvement mentioned earlier: Rather than N queries, we are now on the order of the square root of that.

2.1.1 Practical Application and Limitations of Grover's Algorithm

At first glance, the improvements afforded by Grover's algorithm seem substantial. Though not a perfect algorithm in the sense that there is a probabilistic element and some chance of error[2], Grover's clearly has an advantage in runtime complexity over the classical strategy. Therefore, we identify that there is some trade-off between computational efficiency and certainty of outcome when choosing to use Grover's to tackle the unstructured search problem. In terms of correctness, it can be proven analytically that probability of success when using Grover's is always greater than or equal to $1 - \frac{1}{N}$ [7], meaning success is typically likely, particularly as N grows large. We are also free to run the algorithm in its entirety as many times as we choose, and if any one of them succeeds, it will identify a solution, with a high chance of doing so using fewer queries to f than its classical counterpart. With the classical cost on the order of 2^{128} for AES-128, a naive calculation finds the new cost with Grover's square-root speedup to be around 2^{64}, perhaps with some small constant to account for overhead and setting up the quantum circuit. However, this estimation does not take into account a number of practical concerns and considerations.

First, Grover's algorithm, as described in Section 2.1, uses an oracle f, which in our case corresponds to the AES protocol. AES, a complicated protocol in and of itself, must be implemented in the quantum hardware using computationally expensive gates and structures [5, 10], which we introduce and discuss in more detail in Section 2.3. This adds significant cost to the runtime of the algorithm [5], which the commonly cited Big O abstraction does not capture [11]. Exploring circuit designs of AES and their costs is outside of the scope of this project, but it is an active field of research [12, 13, 10] that will be crucial for full asymptotic analysis and practical implementation of Grover's algorithm applied to AES.

Secondly, as pointed out by Jacques, any realistic attack will require parallelization [5], and unfortunately, Grover's algorithm does not parallelize well. If we consider the full key space of N keys and partition it into P subsets, we can assign each of the P partitions to a machine responsible for a search space containing $\frac{N}{P}$ keys. With the square-root speedup over the classical

---

[2] in contrast to the classical runtime we discussed, where the exhaustive search strategy leaves no chance of error whatsoever

method as discussed, we can now find the key in $\mathcal{O}(\sqrt{\frac{N}{P}})$ time. When comparing this to serial Grover's with a runtime of $\mathcal{O}(\sqrt{N})$, we find that parallelizing only reduces the runtime by a factor of $\sqrt{P}$, not P. Worse yet, the additional partitions add an overhead, with the total cost of operations going up to $P \cdot \mathcal{O}(\sqrt{\frac{N}{P}}) = \mathcal{O}(\sqrt{P \cdot N})$. Thus, in our case with AES-128, the bad parallelism of Grover means its cost is certainly higher than the naive estimation of 2^{64}.

The parallelization argument holds for any algorithm on the order of $\sqrt{N}$, and it turns out $\mathcal{O}(\sqrt{N})$ is the best we can do: For the general unstructured search problem, it has been proven that Grover's algorithm is *asymptotically optimal* [14, 15], meaning that as the search space grows, it is not possible to find an algorithm that performs better than on the order of $\sqrt{N}$ in the worst case. Despite its clear limitations, Grover's is thus still a contender to break the security of AES and similar symmetric encryption schemes by quantum computing, motivating the exploration we undertake in this project.

I'm trying to introduce some sort of motivation for working with Grover's despite the pessimistic outlook we discussed in this section, but this last sentence might still be a little unfounded. Thoughts?

2.2 Quantum Error Correction for Stable Quantum Memory

As we briefly touched upon, future quantum computers will certainly require some form of error correction, as quantum systems are prone to noise, causing logical errors during computation. Unlike in a classical computer where the only errors we consider are bit-flip errors (where 0 is interpreted as 1, or vice versa), quantum information can be destroyed by two separate types of errors: bit-flips and phase-flips. This, in combination with quantum hardware being inherently more difficult to work with from an engineering perspective, has the unfortunate result of massively complicating the circuitry required for computation as soon as any practically viable error correction measures are introduced, a topic we will revisit in more detail in Section 2.3.

2.2.1 Toric Surface Code

The standard method of error correction that is used in all existing quantum computers built for fault-tolerance is the *surface code*. Following the discovery of the toric code[3] in 1997 [17], toric surface code has remained a prevalent error correcting architecture because it is based on simple principles and maintains geometric locality[4].

This is my impression, but it's hard to find a specific source.

In essence, surface code works by creating a grid of data qubits and measurement qubits interspersed and defining *stabilizers* that flag bit-flip (X) errors and phase-flip (Z) errors when

---

[3] There is some inconsistency in the literature about what constitutes a *toric code*, a *surface code*, and a *toric surface code*. We adhere to the nomenclature set by IBM [16], where toric codes exist in a grid across the surface of a torus with periodic boundary conditions, and surface codes and toric surface codes are used interchangeably to refer specifically to the portion of a torus grid laid flat with non-periodic boundary conditions.

[4] The only links required in the surface code are between neighboring qubits, as opposed to long-distance connections required by certain other codes that are more challenging from an engineering standpoint.

they occur in specific locations. The error correcting capability lies in measuring the stabilizers across the grid repeatedly, decoding where errors have occured, and applying the appropriate correction based on the stabilizer measurement. The stabilizer formalism can in many ways be viewed as the quantum generalization of parity checks [18, 19], which are often used similarly in classical error correction to identify where and how to restore corrupted data.

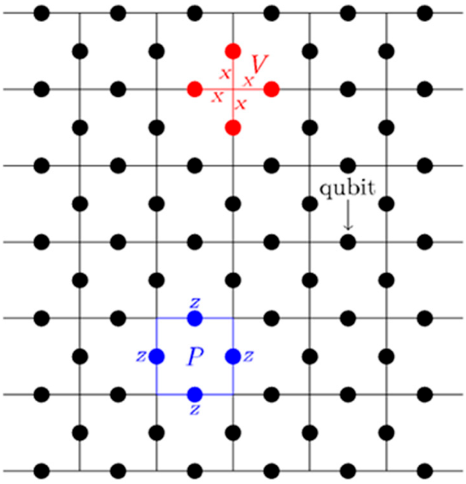


Figure 2.1: A surface code grid with physical data qubits located on the edges, demonstrating a plaquette (Z stabilizer) in blue and a vertex (X stabilizer) in red. Although only one of each is shown here, the entire grid is covered in these stabilizers in a repeating pattern.
Image source: Sayedsalehi et al. [20]

In Fig. 2.1, we show how taking four qubits surrounding a vertex or a square face in the grid (known as a *plaquette*) defines an X or Z stabilizer respectively. Check qubits for measurements in the stabilizers are a crucial element of quantum error correction specifically, as measuring the actual state of data qubits to identify data corruption would destroy their states, and thus the data itself.

I might remake this graphic myself.

The stabilizer formalism allows us to define algebraically the units described visually in Fig. 2.1 with two sets of commuting stabilizer generators. For each vertex v in the grid, we have an X stabilizer

$$S_v^X = \prod_{j \in \text{star}(v)} X_j,$$

which multiplies the Pauli X operator on the four[5] edges touching v. We similarly define a Z stabilizer for each plaquette p as

$$S_p^Z = \prod_{j \in \text{boundary}(p)} Z_j,$$

which multiplies the Pauli Z operator on the edges surrounding p. The groups formed by all S^X and S^Z respectively have elements that are all tensor products of Pauli operators and that

---

[5] or fewer at the grid boundaries

commute. Together, each of these groups has a shared eigenspace constrained by

$$S_i \left|\psi\right\rangle = + \left|\psi\right\rangle \qquad \forall i,$$

where S_i represents a general stabilizer (either X or Z) applied to an appropriate vertex or plaquette, denoted by i. This eigenspace contains the legal code words, or in this case, the states corresponding to logical 0 and logical 1 (henceforth denoted by $\left|0\right\rangle_L$ and $\left|1\right\rangle_L$ respectively).

The Pauli operators satisfy the anticommutation relation $\sigma_j\sigma_k + \sigma_k\sigma_j = 2\delta_{jk}\mathbb{I}_2$, where σ_i are the Pauli operators according to the naming convention $\sigma_1 = X, \sigma_2 = Y, \sigma_3 = Z$, δ_{jk} is the Kronecker delta and $\mathbb{I}_2$ is the 2×2 identity matrix [21]. Thus, if an error operator E commutes with a stabilizer S_i, the eigenvalue of the surrounding qubits' state is unchanged:

$$S_i(E \left|\psi\right\rangle) = E(S_i \left|\psi\right\rangle) = +E \left|\psi\right\rangle .$$

However, if E and S_i anticommute, we will have $S_i(E \left|\psi\right\rangle) = -E(S_i \left|\psi\right\rangle)$ instead, which changes the eigenvalue of the local stabilizer state from $+1$ to -1.

At this point, we can do a measurement of the ancillary check qubits of the stabilizers to determine which ones violate our constraints and flag errors in their neighbors. Thinking back to the notion of parity checks, we consider the result of each stabilizer measurements a remapping of the parity check values 0 and 1 to $+1$ and -1 respectively. This is a very convenient mapping for us, because when we work with Z operators[6], ±1 happen to be the eigenvalues of the eigenstates $\left|0\right\rangle$ and $\left|1\right\rangle$. Thus, ±1 represent a way for us to access information about the state without measuring the state itself. In other words, if we measure the check qubits in the stabilizers and see $\left|0\right\rangle$, we know the data qubits are in the eigenstate $+1$, and we conclude no error has occurred among the neighboring qubits of this stabilizer[7]. However, if we see $\left|1\right\rangle$, we know the surrounding data qubits are in the eigenstate -1, and an error has occurred here. Together, the eigenvalues s_i that we observe at each of the m stabilizer measurements form a string $(s_1, s_2, \ldots, s_{m-1}, s_m) \in \{\pm1\}^m$, called the *syndrome* of the full state. It tells us about what errors are likely to have occurred and where, which we can use to determine what corrections we would like to apply to restore the logical state.

While conceptually simple, the toric surface code has a clear drawback in that it is very qubit-hungry, with estimates predicting a ratio around a 1000 to 1 of physical to logical qubits required for realistic error correction rates [5, 22]. This has sparked further study and innovation in order to uncover more favorable schemes in terms of resource efficiency.

---

[6] A measurement of X is slightly more involved, but still unproblematic. The relation $HXH = Z$ allows us to simply apply H and do a measurement of Z instead.

[7] Strictly speaking, we only know an even number of errors has occurred. For instance, two errors in the neighboring qubits of the stabilizer will also flip its state back into the eigenstate $+1$, and thus be undetectable.

2.2.2 Alternatives to Toric Surface Code

The continued usage of the surface code at the current correction rates is one of the underlying assumptions in Jacques' predictions. Thus, it is interesting for the sake of our analysis to see how the prognosis for Grover's algorithm changes with a different choice of error correcting code. A few promising examples can be found in the *Low Density Parity Check* (LDPC) family of codes, of which the surface code is a special case. In particular, *Bivariate Bicycle code*, also known as BB code, is a strong potential candidate [23]. Though similar to surface codes at a high level of abstraction, BB codes differ in that their stabilizers are not geometrically local, and has therefore been called "surface code with extra long-distance checks" [24]. These checks allow for a number of advantages over the surface code, such as a higher threshold for error correction and lower qubit overhead [23].

Do I need a source for this, and if so, what?

We begin by defining the following shift matrices:

$$x = P_\ell \otimes \mathbb{I}_m, \quad y = \mathbb{I}_\ell \otimes P_m,$$

where $\mathbb{I}_m$ is the $m \times m$ identity matrix and P is a cyclic permutation matrix in the form $\sum_{i=1}^{\ell} |i\rangle \langle i+1|$, which maps $|i+1\rangle$ to $|i\rangle$ [24]. In addition, x and y satisfy $xy = yx$ and $x^\ell = y^m = \mathbb{I}_{\ell \times m}$. We use x and y to represent stepping through the grid in the x- and y-direction respectively.

Can I use m here when I used m to mean something else previously? I'm running out of letters!

Formally, BB code is defined as a pair of $\ell m \times \ell m$ binary matrices $A(x,y), B(x,y)$ that are sums of our shift matrices x and y to integer powers. These matrix polynomials are then used to define parity check matrices H^X and H^Z:

$$H^X = [A|B], \quad H^Z = [B^\intercal | A^\intercal].$$

The dimension of H^X and H^Z is thus $\ell m \times 2\ell m$. For our purposes, $2\ell m = n$, the number of physical qubits, and $\ell m = \frac{n}{2}$, which is the number of vertex and plaquette stabilizers in the grid. We use these two matrices to index which qubits in the grid are to be acted on by stabilizers. Each row contains the qubit indices for a specific stabilizer, and each column is a binary indication of whether a specific qubit in the grid is part of that stabilizer. For instance, in BB code, each row will therefore have six 1s, as each stabilizer acts on six qubits, while the surface code in this notation would have four 1s per row (at most), because each stabilizer acts on only four qubits. The shifting mechanism defined by A and B is the mathematical way to describe the systematic construction of an evenly-spaced pattern repeating throughout the grid. With some appropriate indexing of the grid, we are able to specify the exact qubits to include in each stabilizer, uniquely defining a BB code.

Measuring a stabilizer corresponds to checking whether the associated parity condition set in the H^X and H^Z matrices is satisfied. Specifically, we require that

$$H^X (H^Z)^\intercal = 0.$$

This requirement ensures that the stabilizers commute, so that we can apply similar logic as with the standard surface code to identify the stabilizers for which this constraint does not hold, where the anticommutation relation will let us detect stabilizers in the -1 eigenstate. The syndrome that is produced informs the corrections we make as a result.

I'm thinking about adding more details about the decoding process for surface code and/or BB code, which would go here for BB code. Is that necessary?

It is common to use the notation $[[n, k, d]]$ to specify the parameters of an error correcting code, where n is the number of physical qubits, k is the number of logical qubits we encode, and d is the number of errors the code can correct. IBM, the developer of the BB code, has primarily considered a BB code they call the *gross* code[8], which is a [[144, 12, 12]]-code (though many other variants exist and are perhaps better suited for our purposes — see Chapter 3). In this specification, 144 physical qubits are required to encode 12 qubits, and the code can correct up to 12 errors without issue [23]. With the 1000 to 1 ratio needed for the toric surface code, the gross code (and BB codes in general) offer a clear improvement in this regard. As an additional plus, the connections between qubits in the BB code form a Tanner graph [25], which is a bipartite graph that can be separated into two disjoint sets. In this structure, nodes (qubits) within each set are only connected to members of the other, never to one another. This can be useful when physically constructing this architecture, since it allows the hardware to be split cleanly into two separate levels.

This is where I'll make a subsection discussing tile codes as well if I end up including them. At the moment I'm not sure I'll have time to look into them, so I'll leave it at BB codes.

2.3 Universal Gate Sets for Quantum Computation

Assuming we have created some scheme for fault-tolerant storage of quantum data, we still face a number of significant challenges before we can perform any fault-tolerant computations. In particular, the *Eastin-Knill theorem* [26] states the following:

The Eastin-Knill theorem

For any nontrivial local-error-detecting quantum code, the set of logical unitary product operators is not universal.

I think the key here is that the logical unitary product operators are specifically related to what operators can be expressed and worked with in the stabilizer formalism, which as we know the T-gates cannot.

---

[8] after the term for a dozen dozen, known as a gross

To understand what this means, we need to to delve further into how applying gates to quantum states forms a circuit. Mathematically, we can represent any quantum circuit on n qubits with a single unitary matrix[9]

$$U \in \mathbb{C}^{2^n \times 2^n}.$$

While this representation can certainly be useful, like in classical simulations with very few qubits, collapsing the circuit into one matrix in this way generally does not capture a lot of important behaviors and phenomena we are interested in when studying quantum error correction. In hardware, we never apply a single giant unitary gate; instead, as we have seen, we decompose the computation into smaller, simpler gates. In doing so, we could in principle have our pick from a large selection of possible gates, but if for simplicity we restrict ourselves to implementing only a few different types, we would like the set of gates we choose to be *universal*. A universal gate set can approximate any such unitary U to an arbitrary degree of precision with a finite sequence of gates, an important property for our gate set to have, as without it, there will always be computations we cannot perform.

Put another way, the theorem tells us that we cannot have a universal gate set where every gate is *transversal*, meaning a gate that can transform a logical qubit by applying a gate to each of the n physical qubits:

$$G_L = \bigotimes_{i=1}^{n} G_i.$$

Additionally, transversal gates ensure single-qubit errors do not propagate to multiple qubits within an error correcting patch, making them inherently fault-tolerant: A single error occurring when applying the gate will produce at most one error for the error-correcting patch to correct. This is clearly a desirable property for gates in our chosen gate set to have, but the Eastin-Knill theorem describes a fundamental limitation, in that any gate set we choose capable of approximating arbitrary quantum circuits will contain at least one nontransversal gate.

What is a good source to use for the definition of transversal?

What is the link between transversality (a logical gate is applied by applying the same gate to its constituent physical qubits individually) and being able to manipulate the stabilizers only in order to alter the logical state the patch contains? By which I mean, why is it that there exists a way for me to mess with the stabilizers and achieve the effect of an X, Y, Z or H gate, but not the T gate? And I know rationally that Eastin-Knill is the answer, since at least one gate is hard, but I don't have a conceptual understanding of why there isn't some manipulation of stabilizers corresponding to the application of the T gate.

---

[9] A unitary matrix, also known as simply a unitary, is a norm-preserving matrix $U \in \mathbb{C}^N \times \mathbb{C}^N, N \in \mathbb{N}$ whose conjugate transpose $U^\dagger$ is also its inverse: $U^\dagger U = UU^\dagger = \mathbb{I}_N$.

ChatGPT's answer: The clean way to see the connection is to separate three layers: (i) the stabilizer group that *defines* the code space, (ii) logical Pauli operators as nontrivial elements of the normalizer of that stabilizer group, and (iii) logical gates as automorphisms of the code space. In the surface (toric) code, the stabilizers are local X-type stars and Z-type plaquettes, and the logical operators are noncontractible X- and Z-strings. A logical unitary is implementable by "manipulating stabilizers" precisely when it corresponds to a symmetry or locality-preserving transformation of this Pauli/stabilizer structure.

Transversality fits naturally into this picture. A transversal gate acts independently on each physical qubit, so conjugation by the global unitary reduces to conjugation of each local Pauli. If the single-qubit gate maps Pauli operators to Pauli operators (as Clifford gates do), then each stabilizer generator—being a product of Paulis—gets mapped to another Pauli product. In other words, the stabilizer group is mapped to itself (possibly with relabeling), and the code space is preserved. That is why Clifford gates are compatible with transversal implementations in CSS codes like the surface code: their defining algebraic property is that they normalize the Pauli group.

When you "manipulate stabilizers" in the surface code—via code deformation, lattice symmetries, boundary twists, or relabeling $X \leftrightarrow Z$—you are again performing operations that map stabilizer generators to stabilizer generators. Conceptually, you are implementing an automorphism of the stabilizer group. Any such automorphism necessarily induces a transformation on the logical Pauli operators, and because the logical Pauli operators generate the logical Pauli group, the induced logical gate must be a Clifford. This is the essential restriction: stabilizer-preserving transformations can only produce logical operations that send Paulis to Paulis.

Now consider the T gate. Conjugation by T does not preserve the Pauli group: for example,

$$TXT^{\dagger} = \frac{1}{\sqrt{2}}(X + Y),$$

which is not a Pauli operator but a coherent superposition of Paulis. This already tells you something fundamental. If you tried to implement logical T purely by "messing with stabilizers", you would need a transformation that sends a logical Pauli operator to a non-Pauli operator while still mapping the stabilizer group to itself. But stabilizers, by definition, are Pauli products. Any operation that preserves the stabilizer formalism must map Pauli products to Pauli products under conjugation. There is simply no room inside that algebraic structure to accommodate the non-Pauli images produced by T.

From a more geometric/topological perspective, the surface code is a two-dimensional topological stabilizer code. Any locality-preserving logical gate corresponds to a topological symmetry of the anyon model underlying the code. In the surface code, those symmetries form exactly the Clifford group. The T gate would require a transformation that mixes topological charge sectors in a way that cannot be realized by a locality-preserving deformation of the lattice or stabilizers. So it is not just that "at least one gate must be hard" (Eastin–Knill); it is that the entire stabilizer/topological framework only supports Clifford automorphisms.

Conceptually, then, the answer is this: manipulating stabilizers works precisely when the desired logical gate is an automorphism of the Pauli/stabilizer structure. Clifford gates have that property; T does not. Because T pushes you outside the Pauli algebra under conjugation, it cannot arise from any transformation that merely reshuffles or locally deforms stabilizers. That algebraic incompatibility is the deep reason you cannot realize T by stabi-

2.3.1 Choices to Make When Performing Quantum Computations

Does this section (perhaps rewritten somewhat) fit better in the methods section?

There are a few separate challenges with doing operations on quantum data that we must address in order to reach our goal of a fault-tolerant circuit for Grover's algorithm. Assuming we have made some choice of error correcting code, first, ① we must choose an appropriate universal gate set. Next, ② we need to take the operations we want and determine an equivalent calculation using only the gates in our chosen gate set. Finally, we face another problem entirely in taking our chosen gate set to the quantum hardware and ③ figuring out how the circuit we designed will be implemented on the error-correcting patches corresponding to logical qubits.

With regards to ①, *Clifford+T* is a common choice of universal gate set, fittingly comprised of the *Clifford* and *T-gates*. Clifford gates are unitaries that map Pauli operators to Pauli operators under conjugation, which makes them very well-suited as logical gates in LDPC codes like the ones we have seen. Applying Clifford gates to codes based on X and Z stabilizers respects the stabilizer structure and code space, and since they can be implemented by performing single-qubit operations across all qubits, they are also transversal. The possibility of these simple, single-qubit operations not only grants Clifford gates transversality and built-in fault-tolerance, but it also means they are considered computationally easy to implement.

However, the T-gates, given by

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix},$$

are a different story entirely. A requirement for universality, these gates are neither transversal nor easy computationally; owing to the Eastin-Knill restriction, they necessitate complicated workarounds to be implemented fault-tolerantly (which we return to in Section 2.4). For now, Jacques' argument fundamentally relies on the assumption that we will encode the Grover circuit with the Clifford+T gate set, where the number of T-gates is ultimately the determining factor in his estimation of the time complexity of the algorithm. However, if there were a better way to create the circuit that requires less T-gates (or T-gate equivalents), we could possibly see substantial improvements, since the huge estimated cost of implementing T-gates is a major contributor to why Grover's algorithm works out so poorly in practice.

If it becomes at all interesting to explore other options besides the Clifford+T gate set, I will define and discuss them here. Otherwise, I will elaborate a bit and justify my choice of sticking to Clifford+T.

As for the next challenge, while ② is not necessarily easy in isolation, provided we have our operation expressed as a circuit in some other gate set, there fortunately exist tools such as Synthetiq [27] that can do this optimization and search process for us. In our case with Grover's algorithm, the circuit with freely chosen gates is well known, and Synthetiq can be used to convert it into whatever other limited gate set we would like.

The issue that this project boils down to in many ways is therefore ③. While certain logical operators are known or can be identified by the software tools, others might pose more of a challenge. As highlighted earlier in Section 2.1, the gates we need to perform the full circuit for Grover's algorithm are H, Z_f and Z_{OR}. Of these, H is largely considered computationally easy, taking our standard X and Z corrections and swapping them, since H maps $|0\rangle$ to $|+\rangle$, $|1\rangle$ to $|-\rangle$, and vice versa, leaving us in the realm of the computationally easy Clifford gates. The phase query gates Z_f and Z_{OR} are more problematic, however, and will require the computationally hard T-gates.

Might add a figure name here

2.4 Phase Query Gates in Error Correcting Architecture

TBA after I finish coding this and figure out how it works, because this is where the issues with T-gates arise. Will touch upon T-gate implementation in more detail, magic state distillation for instance.

Chapter 3

Methods

I have more or less recreated the results from Brekke and Trefjord [24], where they compare the storage capabilities of the surface code and BB code for similar numbers of qubits over time.

Since, as we have seen, applying the T-gate does not in any way reduce to stabilizer operations, software such as PanQEC is not able to help us reduce this cost.

Here I could segue into how we need to use the standard matrix form which grows exponentially, but it feels kind of pointless. I have not really made the argument that it is impossible to do this a smarter way than I am doing it.

Potential plan: Do this the hard way with the 16 qubit BB code and the 17 qubit surface code. Go through and make the circuits or at least the individual gates with this lifted unitary matrix approach and compare them.

Unfinished section, not looking for supervisor feedback here yet.

The goal of this project is to simulate the circuitry of Grover's algorithm with two logical qubits, comparing how the circuit looks, the qubit requirements and overall complexity for a few different types of error correcting codes. Recall the expression for the unitary matrix defining an arbitrary quantum circuit:

$$U \in \mathbb{C}^{2^n \times 2^n}.$$

Given a gate set, we obtain U by "lifting" the individual gate matrices to the appropriate dimension $2^n \times 2^n$ and multiplying them in the right order.

Noise is not a single perturbation at the end of our circuit, and other aspects like timing, decoherence and idle errors are completely lost. Addressing all of these things is very complex and not fully understood, but we still wish to use a different approach that takes into account some of them somehow... INTRODUCE TICK MODEL AND SEQUENTIAL APPLICATiON OF GATES!!

Rewrite this sentence; doesn't make sense as is.

change word

3.0.1 Error Models

Currently, the error model used is a time-step system, similar to video game logic that progresses with ticks to simulate time passing. Errors are modeled stochastically, with a fixed probability $p = 0.01$ of occurring on any given qubit at any time step. Further improvements to this error model are discussed in Section 5.2.

3.0.2 Software Aids

The software package PanQEC [28] was used to visualize a few different kinds of error correcting codes. It also has functionality for determining logical operators of various types, which correct errors and perform small single-qubit computations.

As previously discussed, the challenge of doing operations on our fault-tolerantly stored data is multi-faceted. We need to figure out hwo to use gates to perform the computations we want, and then we need to realize those gates in the physical machinery. Fortunately for us, circuits for Grover's algorithm already exist and have even been implemented physically in Noisy Intermeidate-Scale Quantum (NISQ) computing setups (as opposed to fault-tolerant). There is no error correction taking place in the NISQ approach, but we still have a circuit that performs the operation we want.

Synthetiq is another tool, which allows you to take any circuit and express it in terms of another limited gateset, such as Clifford+T (although it does support custom gatesets). HOWEVER; this does not take into account the fault-tolerant aspect of what we're doing, so we need a way to find a fault-tolerant circuit and simulate it.

I am working with Panqec, which is a visualization tool developed by Arthur Pesak. IDeally, the method would be to use Panqec to visualize a couple different types of code, and then figure out ways to do logical operations on whatever I stabilize there.

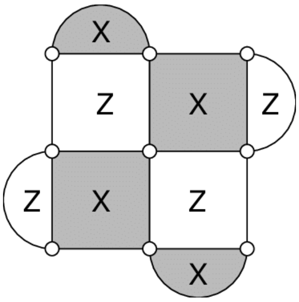
3.1 Coding Fault-tolerant Quantum Memory

3.1.1 Surface Code

The smallest possible surface code requires 17 qubits (or possibly 10, if you reuse one ancillary qubit for measurement rather than having 8 individual ones).

3.1.2 BB Code

Liang and Chen [29] have found all BB codes using less than 160 qubits. The smallest of these, the $[[16, 4, 4]]$-code, is the only option I have considered because otherwise I run into major memory issues.



3.2 Applying Gates to Error-correcting Patches

Chapter 4

Results

Plan: Compare application of specific gates we need in different error correction architecture (surface code, BB code, possibly 5-qubit code as well?). See if I can get a full or partial circuit running. Describe what the circuit might look like, even if I end up not being able to run it.

Should I have a separate discussion section?

Chapter 5

Conclusion

Summarize your findings and future work.

5.1 What is Jacques' full line of argumentation?

This section is not something I'm planning on including in the text, but I just put it here to save it as a reference for myself.

In the normal case when we would use Grover's algorithm, we assume no structure. The argument for there being security in this case is that the number of potential keys is so large that we would not be able to do a brute-force search attack on a classic computer in any reasonable amoutn of time (brute-force being the only option here beceause there is no structure to the problem or the way the keys are determined). The circuit for the quantum algorithm Grover's is such that we may have a speedup on the order of the square root. We also know (from my own previous work) that the runtime of Grover's scales with the number of Grover iterates, which themselves scale with the number of queries to the arbitrary funciton f, or the AES algorithm in our case, which is needed to see if a binary string x is a solution. Each Grover iteration requires running the entire AES encryption circuit, and this is the costly part.

There might also be constant factors here we don't know about, so to see the full picture, we need to know exactly how our f, namely AES, will be implemented in a circuit. Jacques provides a table of estimates of different values on page 20 of the PDF (slide 8 in the deck). The three quantities of apparent interest are Clifford gates (simple, computationally easy gates), T gates (a different type of gate that requires a huge number of qubits) and the depth, which I take to mean the number of gates that have to be applied serially, kind of functioning as a proxy for time (as they cannot be run in parallel). Finally, we have an estimate of a couple thousand logical qubits required for each size of AES in the table, but of course there could be millions of qubits hiding behind this number.

We definitely need some form of error correction, since quantum computations are very prone to noise. But we are orders of magnitude off the number of physical qubits we need in order to be anywhere near the estimated number of necessary logical qubits.

At this point, he introduces NIST's 2017 MAXDEPTH metric as a baseline for how many logical gates/operations the current quantum computing architectures perform serially over certain periods of time, like a year (2^{40}), a decade (2^{64}) or a millenium ($2^{9}6$)[1]. Additionally, he makes clear that any sort of realistic attack on AES would have to be parallelized.

This paper will attempt to tackle just that. What happens if we use something else instead, and what might that be?

5.2 Future Work

Does the decoder we use for correcting errors matter at all and what is the efficiency/accuracy tradeoff?

5.3 Declaration of AI Usage

Rewrite this section into formal text.

I have used AI as an aid in coding and will probably continue to do so.
I use it to learn things sometimes and provide intuitive explanations where I otherwise just have equations to study.
I use it very minimally when writing, pretty much just when I can't remember a certain word or if I've used a word a lot and need a synonym (like a thesaurus).
I use it as a search engine when I have complicated search terms that Google can't find for me.
I ask it to format references from websites etc when there is not a downloadable option provided by the authors.

---

[1] He also points out that this limit does not reflect decoherence concerns, i.e. the quantum state collapsing and quantum data disappearing.

Bibliography

[1] *Symmetric-key algorithms.* Wikipedia. URL: `https://en.wikipedia.org/wiki/Symmetric-key_algorithm`.

[2] National Institute of Standards et al. *Advanced Encryption Standard (AES).* en. 2023-05-09 04:05:00 2023. DOI: `https://doi.org/10.6028/NIST.FIPS.197-upd1`. URL: `https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936594`.

[3] Ronald de Wolf. *Quantum Computing: Lecture Notes.* 2023. arXiv: `1907.09415 [quant-ph]`. URL: `https://arxiv.org/abs/1907.09415`.

[4] Thomas Giechaung Wong. *Introduction to classical and quantum computing.* Omaha, NE: Rooted Grove, 2022. ISBN: 979-8-9855931-1-2.

[5] Samuel Jaques. *Quantum Attacks on AES.* Presentation at the 26th International Conference on Cryptographic Hardware and Embedded Systems (CHES 2024). Available online: `https://ches.iacr.org/2024/Jaques_CHES_2024.pdf`. July 2024.

[6] Emma Storberg. *TEK5550 Mandatory Assignment 1.* Unpublished midterm exam. University of Oslo, 2025.

[7] Qiskit. *Grover's Algorithm | Understanding Quantum Information & Computation - Lesson 08.* Nov. 2023. URL: `https://www.youtube.com/watch?v=hnpjC8WQVrQ`.

[8] Wikipedia contributors. *Quantum logic gate — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Quantum_logic_gate&oldid=1278673600`. [Online; accessed 4-March-2025]. 2025.

[9] Wikipedia contributors. *Quantum superposition — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Quantum_superposition&oldid=1275226563`. [Online; accessed 4-March-2025]. 2025.

[10] Huinan Chen et al. *Quantum circuit for implementing AES S-box with low costs.* 2025. arXiv: `2503.06097 [quant-ph]`. URL: `https://arxiv.org/abs/2503.06097`.

[11] Wikipedia contributors. *Big O notation — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-January-2026]. 2026. URL: `https://en.wikipedia.org/w/index.php?title=Big_O_notation&oldid=1333898274`.

[12] Liao-Liang Jiang et al. "Constructing resource-efficient quantum circuits for AES". In: *Frontiers in Physics* Volume 13 - 2025 (2025). ISSN: 2296-424X. DOI: `10.3389/fphy.2025.1582819`. URL: `https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2025.1582819`.

[13] Qun Liu et al. *Improved Quantum Circuits for AES: Reducing the Depth and the Number of Qubits.* Cryptology ePrint Archive, Paper 2023/1417. 2023. URL: `https://eprint.iacr.org/2023/1417`.

[14] Charles H. Bennett et al. "Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1510–1523. ISSN: 1095-7111. DOI: `10.1137/s0097539796300933`. URL: `http://dx.doi.org/10.1137/S0097539796300933`.

[15] Christof Zalka. "Grover's quantum searching algorithm is optimal". In: *Physical Review A* 60.4 (Oct. 1999), pp. 2746–2751. ISSN: 1094-1622. DOI: `10.1103/physreva.60.2746`. URL: `http://dx.doi.org/10.1103/PhysRevA.60.2746`.

[16] IBM Quantum. *Other code families.* Accessed: 2025-01-28. IBM Quantum Learning. 2025. URL: `https://quantum.cloud.ibm.com/learning/en/courses/foundations-of-quantum-error-correction/quantum-code-constructions/other-code-families`.

[17] A.Yu. Kitaev. "Fault-tolerant quantum computation by anyons". In: *Annals of Physics* 303.1 (Jan. 2003), pp. 2–30. ISSN: 0003-4916. DOI: `10.1016/s0003-4916(02)00018-0`. URL: `http://dx.doi.org/10.1016/S0003-4916(02)00018-0`.

[18] Wikipedia contributors. *Stabilizer code — Wikipedia, The Free Encyclopedia.* [Online; accessed 28-January-2026]. 2026. URL: `https://en.wikipedia.org/w/index.php?title=Stabilizer_code&oldid=1332747113`.

[19] Arthur Pesah. *The stabilizer trilogy I — Stabilizer codes.* Blog post. Jan. 31, 2023. URL: `https://arthurpesah.me/blog/2023-01-31-stabilizer-formalism-1/` (visited on 01/28/2025).

[20] Samira Sayedsalehi et al. "Developing and Analyzing the Defect-Based Surface Codes Using Optimization Algorithms". In: *Quantum Reports* 7.2 (2025). ISSN: 2624-960X. DOI: `10.3390/quantum7020025`. URL: `https://www.mdpi.com/2624-960X/7/2/25`.

[21] Wikipedia contributors. *Pauli matrices — Wikipedia, The Free Encyclopedia.* [Online; accessed 28-January-2026]. 2025. URL: `https://en.wikipedia.org/w/index.php?title=Pauli_matrices&oldid=1327001669`.

[22] Craig Gidney et al. "Yoked surface codes". In: *Nature Communications* 16.1 (May 2025). ISSN: 2041-1723. DOI: `10.1038/s41467-025-59714-1`. URL: `http://dx.doi.org/10.1038/s41467-025-59714-1`.

[23] Sergey Bravyi et al. "High-threshold and low-overhead fault-tolerant quantum memory". In: *Nature* 627.8005 (Mar. 2024), pp. 778–782. ISSN: 1476-4687. DOI: `10.1038/s41586-024-07107-7`. URL: `http://dx.doi.org/10.1038/s41586-024-07107-7`.

[24] Anton Brekke and Brage A. Trefjord. *Sommerprosjekt 2025.* Summer student project report on GitHub. [Online; accessed 28-Jan-2026]. 2025. URL: `https://github.com/Bragit123/QEC/blob/main/rapport.pdf`.

[25] Wikipedia contributors. *Tanner graph — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Tanner_graph&oldid=1296985959`. [Online; accessed 28-January-2026]. 2025.

[26] Bryan Eastin and Emanuel Knill. “Restrictions on Transversal Encoded Quantum Gate Sets”. In: *Physical Review Letters* 102.11 (Mar. 2009). ISSN: 1079-7114. DOI: `10.1103/physrevlett.102.110502`. URL: `http://dx.doi.org/10.1103/PhysRevLett.102.110502`.

[27] eth-sri. *Synthetiq: Fast and Versatile Quantum Circuit Synthesis*. Code base, GitHub repository. 2025. URL: `https://github.com/eth-sri/synthetiq`.

[28] panqec. *panqec: Simulation and visualization of quantum error correcting codes*. Version main. 2021. URL: `https://github.com/panqec/panqec` (visited on 02/18/2026).

[29] Zijian Liang and Yu-An Chen. *Self-dual bivariate bicycle codes with transversal Clifford gates*. 2026. arXiv: `2510.05211 [quant-ph]`. URL: `https://arxiv.org/abs/2510.05211`.