BIM'20 Final Report: Quantum Error Correction Playground

Yue Wu
yue.wu@yale.edu
Yale University

Guojun Chen
guojun.chen@yale.edu
Yale University

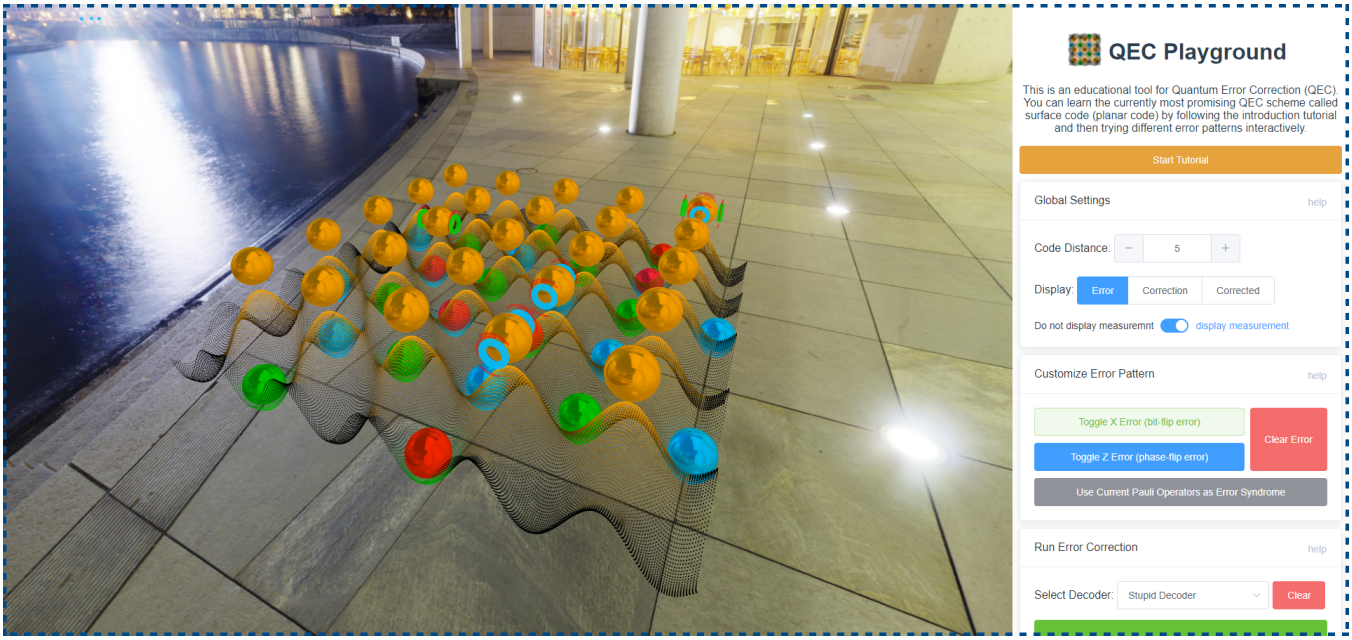


Figure 1: Quantum Error Correction Playground GUI

ABSTRACT

This project is an educational tool to demonstrate how the error correction in quantum computing works using surface code, by letting users customize errors on the physical data qubits and see how the system would recover (if possible) from error states interactively. We investigate several error correction decoders including a naive decoder, a MWPM (Minimum Weight Perfect Matching) decoder and a machine learning based weight-optimized MWPM decoder. Users with various levels of familiarity in quantum computing reported favor in this tool for its fancy GUI, interactive tutorial and sufficient contents. We also demonstrate how this tool can help developers to debug their decoder algorithms by visualizing the correction pattern when some specific pattern of errors occurs.

KEYWORDS

quantum computing, surface code, interactive education, machine learning

1 INTRODUCTION

Quantum computing is an emerging technology that promises exponentially faster processing on some certain tasks than classical computer. Quantum supremacy, thus, is introduced to describe the phenomenon that some tasks can be performed fast and efficiently on quantum computers but will consume years or almost forever on classical computers. Google's Sycamore reaches quantum supremacy in 2019 [2] for the first time in human history. However, currently we don't see any useful applications that can reach quantum supremacy. Apart from the limitation of the poor amount of qubits in the state-of-the-art quantum chip, useful applications also require a reliable quantum computer with long coherent time and extremely low error rate to run complicated quantum algorithms like Shor's algorithm [19] for integer factoring.

Qubits, as the basic unit of quantum computer, stores information and changes its state through quantum gates. There are various sources of errors, *e.g.,* independent qubit errors, quantum gate errors and even measurement errors. These errors happen at a rate of around 0.5%, which is enormously higher than today's classical circuit with error rate of about 1e−17 [14].

Quantum Error Correction (QEC) is thus proposed to build reliable quantum computer on top of unreliable physical components. It can protect quantum states from noise and decoherence effects by adding redundancy to the circuit. Since it's exponentially unlikely that several qubits having errors at the same time, QEC is effective in terms of using polynomial amount of qubits to exponentially lower the error rate. A logical qubit refers to the protected quantum state which is built on several unreliable physical qubits. This is analogous to redundancy codes in classical computing, but requires

BIM '20, Fall term, 2020, New Haven, CT
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

more careful design because error correction must not destroy the superposition state of multiple logical qubits, which is the essence of exponentially faster quantum computing.

Topological quantum code is a special kind of QEC code that uses only geometrically local gates. In this way, physical qubits only interact with nearby qubits so that quantum gates are easy to implement. Surface code is one of the topological quantum codes that has 2D geometry. In this project, we focus on the surface code, more specifically, rotated planar code [15] which optimizes code distance per qubit [5], as shown in Fig. 4.

In this project, we develop an educational tool that help people establish early visualization and understanding of quantum error correction using surface code. It's also helpful to beginners to quantum computing who wants to develop their own surface code decoder by visualizing failed error correction compared to the existing decoders. For the first purpose, we embed interactive tutorials so that users are taught about basic concepts in quantum computing and their corresponding representation in our GUI. The tutorial encourages users to learn by trying out their own settings instead of just following a fixed procedure. For the second purpose, we provide a Minimum Weight Perfect Matching (MWPM) decoder [12] as a baseline, and demonstrate how a decoder implemented by ourselves, called "naive decoder", works sub-optimally under some circumstances. In the same way, one can implement their own QEC decoder and compare with other decoders to have a clue about where to optimize.

Contributions.

- We develop an interactive 3D visualization and simulation tool of quantum surface code using Web technology which is accessible by most devices with modern browser.
- We design an interactive tutorial so that users even without background of quantum computing can learn quantum error correction from this educational tool.
- We implement our own sub-optimal QEC decoder named "naive decoder" and demonstrate how to use this tool to compare with existing decoders and reason why it's sub-optimal.
- We try to use machine learning to train the weights of MWPM decoder, but haven't gotten results currently.

2 BACKGROUND

2.1 Basic Quantum Mechanics

In quantum computing, the qubit or quantum bit is the basic unit of quantum information. One can analogize qubit to the classical bit, which is represented by 0 and 1 in classical computing. For a qubit, there are two computational bases denoted as $|0\rangle$ and $|1\rangle$. They are different from classical bit in that the pure state of a qubit $|\psi\rangle$ can be a superposition of the two bases $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $\alpha, \beta \in \mathbb{C}$. Since qubit pure states $|\psi_0\rangle$ and $|\psi_1\rangle$ are indistinguishable from each other if $|\psi_0\rangle = c|\psi_1\rangle$ given $c \in \mathbb{C}$ [17], one can simply add constrains that $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha \in \mathbb{R}$ to make α, β unique under each quantum state. This can be reparameterized as

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

given $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$. The parameters θ and ϕ can be visualized in spherical coordinates shown in Fig. 2, which is called a Bloch sphere [3]. We use this geometrical representation of single qubit in pure state throughout the project. That means, you can view the spheres in GUI as qubits in Bloch sphere.

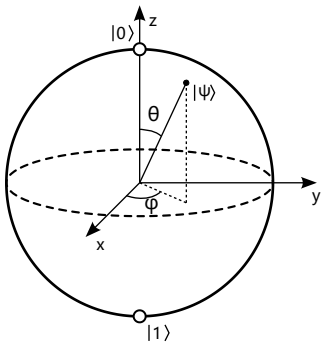


Figure 2: Pure State of a Qubit in Bloch Sphere [20]

Quantum operators can change the state of a qubit, just like classical gates. Analogous to classical NOT gate that maps 0 to 1 and 1 to 0, a Pauli X operator in quantum mechanics maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. Thus, Pauli X operator is also known as bit-flip operator that maps any state $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|1\rangle + \beta|0\rangle$. On the contrary, Pauli Z operator does not have correspondence in classical circuit, which is only introduced in the existence of superposition. Pauli Z operator is also known as phase-flip operator because it changes the relative phase of $|1\rangle$ to $e^{i\pi}|1\rangle = -|1\rangle$, mapping any state $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|0\rangle - \beta|1\rangle$. With Bloch sphere, one can visualize the effect of Pauli operators as rotating the state along its corresponding axis, for example, Pauli X operator rotates the state along X axis and Pauli Z operator rotates the state along Z axis with π angle (180 °). Similarly, Pauli Y operator rotates the state along Y axis, which is equivalent to first applying Z operator then X operator. We use this visualization in our GUI, as shown in Fig. 3.

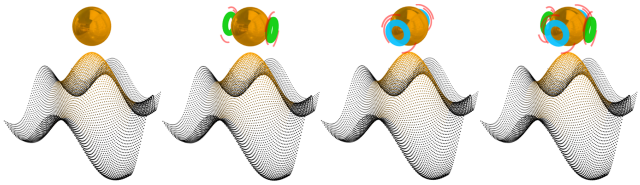


Figure 3: Pauli Operators X (green), Z (blue) and Y (both)

Different from the classical world where measurement is non-destructive to the information, in quantum world measurement usually changes the quantum state by projecting it to a random base of the measurement operator. We can measure the qubit along Z axis, who has the same orthogonal bases $|0\rangle$ and $|1\rangle$ as the computational bases of a qubit. When measuring a qubit along Z axis, it will collapse to one of the orthogonal bases $|0\rangle$ and $|1\rangle$ with probability $|\alpha|^2$ and $|\beta|^2$ respectively. If the measurement result is +1, then after measurement the qubit is in $|0\rangle$ state, otherwise the measurement result is -1 and the qubit is in $|1\rangle$ state after measurement. The measurement of quantum state is destructive in general

because it collapses the superposition state $\alpha|0\rangle + \beta|1\rangle$ to either $|0\rangle$ or $|1\rangle$, losing the information of their relative phase ϕ and relative amplitude $\tan\frac{\theta}{2} = |\frac{\beta}{\alpha}|$.

2.2 Quantum Computing

The exponentially faster quantum computing comes from the superposition of quantum states. If a quantum computer has n qubits, then there are 2^n computational bases $|00..000\rangle$, $|00..001\rangle$, $|00..010\rangle$, ..., $|11..111\rangle$. Unlike a classical circuit with n bits, a quantum circuit with n qubits has 2^n bits information because each of its 2^n computational bases can be individually superposed. The essence of quantum computing is to use these exponentially larger computational bases to search the result in parallel. Quantum algorithms based on quantum walks [6] is proved to give exponential speedups on some tasks by preparing all computational bases and run through a black box to get the result in parallel. Other quantum algorithms based on quantum Fourier transform [13] also perform exponentially faster than classical computers, among which the Shor's algorithm solving the integer factorization problem [19] is famous for its threat to today's widely-used encryption technologies.

2.3 Quantum Error Correction

Quantum error correction (QEC) is used in quantum computing to protect quantum information from errors due to decoherence and other quantum noise. Unlike classical computing, there's high probability of error per qubit per time slice (about 1e-3) in the process of quantum computation (e.g. noisy quantum gates, noisy quantum preparation, and even noisy measurements).

The simplest method of error correction is adding redundancy to qubits, that is to store the qubits information multiple times. When these physical qubit copies disagree in the future, a quantum version repetition code [1] can detect and recover the errors in some circumstances. This is not the same as classical repetition code because direct measurement on each physical qubit will destroy the quantum superposition state. The superposition is the essence of quantum computing supremacy, so a quantum error correction code has to be careful not to destroy the superposition state when doing measurements. Quantum repetition code can only work with single type of Pauli error, so it's not robust to real-world noises composed of all kinds of Pauli errors.

QEC can detect and further recover from errors under certain conditions, but it also introduces more noisy gates and noisy ancilla qubits (used to assist measurement). That means QEC does not necessarily reduce the error rate given more and more physical qubits if the gates and ancilla qubits to implement error correction introduce even more errors than it could recover from.

2.4 Fault Tolerant Quantum Computing

Fault tolerant quantum computing is a step forward from quantum error correction. It requires that given any small error rate requirement ϵ, we can build a logical qubit with at most ϵ error rate with bounded amount of physical qubits. Fault tolerant quantum computing is based on QEC but is more strict in terms of the ability to recover from ubiquitous errors. The fault tolerant quantum computing is still a research front and one need to prove a certain QEC code and it's decoder is fault tolerant mathematically or in experiment, which is not the interest of this project.

2.5 Topological Code: Surface Code

Topological code [4] is a special kind of quantum error correction code that only involves geometrically local gates which means every gate only interacts with adjacent qubits, making it practical to implement. Surface code is one of the topological codes, first proposed as toric code [18] in which every qubit is on the surface of a torus. The surface of torus can be mapped onto a 2D square with periodic boundary condition on top, bottom boundary and left, right boundary. Given the current superconducting quantum chip fabricated in 2D, it's hard to implement toric code only with local gates because of the periodic boundary.

Planar code is a surface code that doesn't require periodic boundary condition, and thus is the most promising one given the current fabrication procedure of superconducting quantum chip. Early version of planar code is a straightforward step from toric code, but it's proven to be sub-optimal in terms of code distance per qubit [5]. Thus, we use the optimized rotated planar code [15] in our project.

The rotated planar code consists of $d \times d$ data qubits. We use only odd number d so that the error correction ability on X and Z Pauli errors is balanced. It has $(d^2-1)/2$ Z stabilizers and $(d^2-1)/2$ X stabilizers, each measures the adjacent 4 or 2 data qubits, according to whether the stabilizer is on the boundary. A demonstration of $d = 3$ rotated planar code is shown in Fig. 4. Each piece of rotated planar code forms a single logical qubit, because there are $d^2 - 1$ stabilizers each locks down a degree of freedom of a data qubit, leaving $d^2 - (d^2 - 1) = 1$ degree of freedom. The goal of quantum error correction is to recover the logical state from random errors, which means the logical state should not be changed after correction. If a logical operator is introduced after correction, then the error correction fails. Logical Pauli X operator and logical Pauli Z operator is shown in Fig. 4 as arrows, meaning that if all the qubits on that arrow has Pauli X or Z error then a logical X or Z operator is introduced respectively. Note that the CX gates and H·CX·H gates are used to implement the measurement with the help of stabilizer ancilla qubits, each is a quantum gate operating on two qubits. We'll not dig into the concrete implementation of measurement using those gates [11] but just take the result that the joint measurement of the adjacent 4 or 2 qubits is feasible.

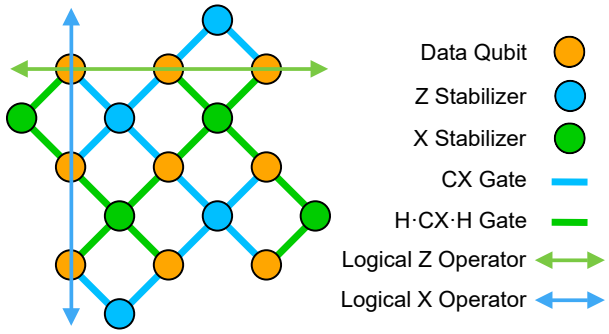


Figure 4: Rotated Planar Code $d = 3$

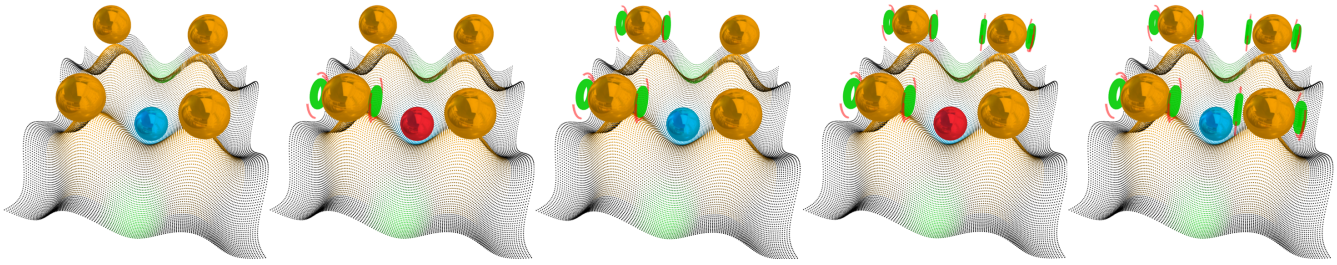


Figure 5: Z Stabilizer Measurement Results (0,1,2,3,4 Errors respectively). Blue for +1 Result, Red for -1 Result.

Take a Z stabilizer with 4 adjacent data qubits as an example, as show in Fig. 5, it measures $Z_1Z_2Z_3Z_4$ of these data qubits. Suppose at first the measurement result is +1, then if one of the four data qubits has a Pauli X error which bit-flip that data qubit, then the measurement result becomes -1 because $1^3 \times (-1)^1 = -1$. However, if two data qubits or all four data qubits have Pauli X errors, then the measurement result is still +1 given that $1^2 \times (-1)^2 = 1$ and $(-1)^4 = 1$, meaning that this stabilizer is not able to detect those errors.

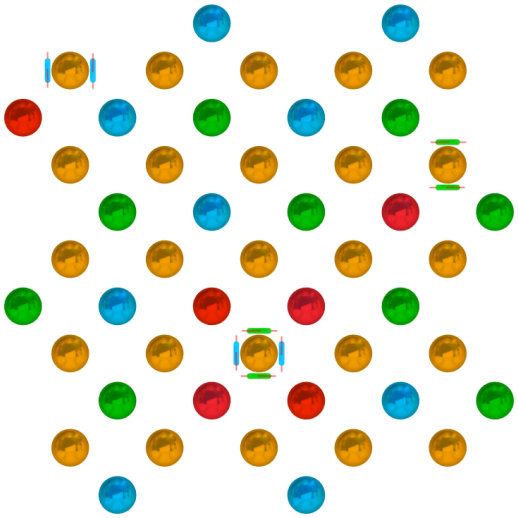


Figure 6: Pauli Errors on Data Qubits (Orange Sphere) and Measurements (Rotated Planar Code $d = 5$). Z stabilizers (blue sphere) and X stabilizers (green sphere) are alternating positioned. Red sphere shows a disagreed measurement of the Z(X) stabilizer, indicating that there are odd numbers of X(Z) errors in the adjacent 4 or 2 data qubits.

The X stabilizer measurements are similar, which only detects odd amounts of Pauli Z errors in the adjacent 4 or 2 data qubits. A demonstration of a random error pattern with both Pauli X errors, Pauli Z errors and Pauli Y errors (having both X and Z errors) is shown in Fig. 6. An error syndrome refers to the stabilizer measurement results. A surface code decoder tries to predict the error pattern from error syndrome and the decoding is successful only if there is no logical operator introduced after the correction and all the stabilizers are back to +1 measurement result after the correction. Note that in practice we only need to remember the errors but do not need to actually correct them by applying quantum gates [15], but we'll not elaborate on it in this project.

3 RELATED WORK

3.1 Surface Code Decoders

Existing works have proposed several decoders for surface code, for example Lookup Table (LUT) decoder [21], Minimum Weight Perfect Matching (MWPM) decoder [9], Machine Learning (ML) decoder [22], Tensor Network (TN) decoder [10] and Union-Find (UF) decoder [8]. We use MWPM decoder as a baseline, and propose a new sub-optimal surface code decoder called "naive decoder" to demonstrate how our tool can help developers to visualize failed error corrections and further reason about why this would happen. MWPM decoder is the default decoder in the tutorial to teach people how quantum error correction works interactively. We also propose an approximate optimization for MWPM decoder by using machine learning to train the weights of MWPM decoder.

3.2 Educational Tools to Quantum Computing

There are several document-based tutorials to quantum computer and a few simulators [7] as well. Unfortunately, None of them is friendly to beginners especially for those who just have interest in quantum error correction but don't want to dig into the details of that. We focus on the quantum error correction and try to illustrate the functionality of fault-tolerant quantum computer in an intuitive and interactive way. As far as we know, this is the first work that even beginners without basic knowledge of quantum computing can get some insights from an interactive tutorial.

4 METHOD

We use Web technologies to implement the GUI (Graphical User Interface), which is portable and accessible on any device with a modern browser. We implement decoders in Rust programming language and it serves as a HTTP server taking error syndromes as input and return the decoded error pattern.

4.1 Web-based GUI

In order to achieve an attractive and user-friendly GUI, we design a 3D visualization system based on WebGL engine that runs in most modern browsers. We optimize the experience for mouse interaction in the 1920×1080 screen, however it also has minimum

support for other screen resolutions. With the ability to zoom in/out, rotate and move around, users are provided flexibility to change the view and focus on any part of the system. When the mouse is hovering over a data qubit, it will jump up and change the color from orange to silver, indicating that user can interact with it by clicking it, as shown in Fig. 7. All the interactions have smooth animations to improve user experience.

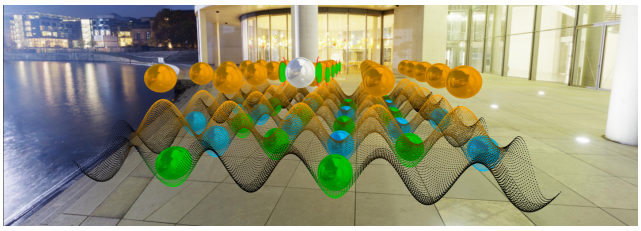


Figure 7: 3D GUI and Mouse Interaction

It also has a side panel on the right providing all the settings and commands, as shown in Fig. 1. The control is implemented based on Vue.js library to reduce the code to implement view updating. We also implement the error correction checker that examines whether the decoder corrects those errors successfully and prompts up the reason of the failed error correction to help developers to quickly know which logical error is introduced or whether the stabilizer recovery failed.

4.2 Decoder Backend

We implemented Minimum Weight Perfect Matching (MWPM) decoder [12] as a baseline, and propose a new sub-optimal decoder called "naive decoder" and an optimization for MWPM decoder using machine learning.

4.2.1 Naive Decoder. Before proposing a decoder, we need to determine the requirement of a successful decoder. First, according to §2.5, decoding is successful only if there is no logical operator introduced after the correction and all the stabilizers are back to +1 measurement result. The latter requirement is equivalent to the case where data qubit errors is composed of multiple closed loops or paths connecting the two boundaries. This is because the stabilizers are still measured +1 after correction if and only if the data qubits only have some logical operators or some stabilizer operators, where logical operator does not change the value of stabilizer measurement, and stabilizer operator itself will not change the stabilizer measurements, just like a Pauli Z operator on the qubit will not change the result of Z measurement. This property does not come for free but is the deliberate design of surface code topology, however we'll not dig into it here. Since the errors can be viewed as chains connecting -1 stabilizers or the boundary, we can conclude that the correction pattern is also chains connecting -1 stabilizers or the boundary, so that after the correction there is only closed loops or paths connecting the two boundaries.

This converts the decoding problem to a perfect matching problem. We need to find a perfect matching of -1 stabilizer, in the special condition that they're free to connect to boundaries as many times as they want. We consider each stabilizer as a node, where some nodes are connecting to boundary and some nodes having errors meaning that they need to be matched. This leads to the idea of naive decoder, that each error node spreads over the graph at a constant speed, and stops once it meets other spreading node or the boundary. Once they meet, they're mark matched and removed from the graph. It's guaranteed that this algorithm terminates after d rounds given any code distance d, because the spreading area grows at constant rate of 1 per round and will definitely meet boundary after that. The error correction pattern is constructed once the matching is done, because we can record the source when spreading to a new node. When some two nodes meet, we can source back to the original error node individually and this is one of the minimum path between the matched nodes (or node to boundary).

This decoder is straightforward but has critical limitations. It only considers local optimal but not global optimal, meaning that it sometimes find a perfect matching that is much less likely than a global optimal one. We'll demonstrate this suboptimality in §5.2.

4.2.2 MWPM Decoder. We implemented this decoder as part of the course assignment in APHY 660 Quantum Information & Computation, 2020, Fall term. Although it's not the contribution of this course project, it's helpful to outline the implementation details in order to introduce our new weight-optimized MWPM decoder based on machine learning.

MWPM stands for minimum weight perfect matching, with existing algorithms [12] running at polynomial time. According to §4.2.1, surface code decoder also try to solve a perfect matching, with the special case of boundary. We try to transform the problem of decoding into a MWPM problem by designing the structure of the graph and its weights. To do so, we first consider the structure of the graph. This graph is not the same as in §4.2.1, because every node in the graph will be exactly matched to another node after running the MWPM algorithm. The error stabilizers are directly mapped to nodes in the graph, but we need to consider the boundary so that they can be matched to error nodes as many times as they want. A single boundary is not enough in this condition, so we add exactly the same amount of boundary nodes as the error nodes to the graph. Each boundary node only connects to one error node, vice versa. Each boundary node connects to every other boundary node, so that if any boundary node is not used to match with error node, it is able to match another boundary node. Each error node connects to every other error nodes. In this way, the amount of nodes in the graph will be even, and thus perfect matching always exists. The structure of the graph is shown in Fig. 8.

Next we consider the weights of the graph. Although there are multiple copies of boundary node, they are inherently the same. Thus, every edge connecting two boundary nodes is of weight 0 so that they are connected for free. Every other edge in the graph has some positive weights, depending on the probability of matching the two node. We can assume the errors on each data qubit is independent and identically distributed with the same error rate p, so that the minimum path connecting the two nodes has the probability of p^l where l is the amount of data qubits on this path. The total probability of all errors are $\prod_i p^{l_i} = p^{\sum_i l_i}$ over all matching paths. We want this probability to be maximum, so that we find an error pattern with maximum probability to occur. If we use $-\log p^l = l \times (-\log p)$ as the weight of each edge, then a MWPM

algorithm will find the error pattern with maximum probability. Note that $-\log p$ is a positive constant, so we can just ignore it by dividing it on all weights. The final graph has weights of integer 0 or l_i.

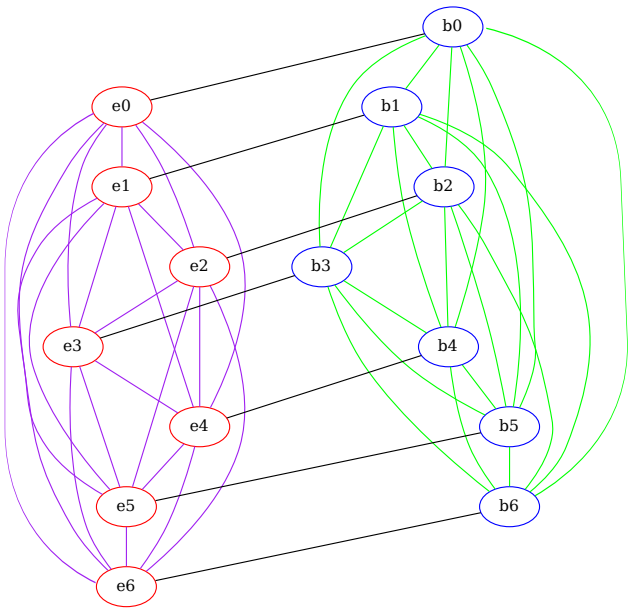


Figure 8: MWPM Graph Structure with 7 Error Stabilizers. $b*$ for Boundary Nodes, $e*$ for Error Nodes.

Note that finding the error pattern with maximum probability does not exactly fit the goal of maximize the probability to decode successfully. This is because some less probable ones may, in sum, be more probable than an error pattern with maximum probability. The maximum likelihood decoder fits the goal exactly but has too large complexity. However, we can see from result that MWPM decoder performs good with random errors in §5.3. We can conclude from experimental result that MWPM is a successful decoder in surface code.

4.2.3 Weight-Optimized MWPM Decoder. Given the knowledge that MWPM decoder is a successful one in surface code, we are inspired by [16] that embedding algorithmic priors to machine learning model can get better result than directly train a network. In our case, since MWPM decoder is a good one, we can use machine learning to train the parameters of this algorithm, more specifically, train the weights of the graph.

For optimizing the weights of MWPM decoder, we first plan to use Keras and some dense layers to train it. The targets for the network are the weights of MWPM, but the loss function and gradient function will not be a simple mean square error and typical gradient between target and predicted values. We have to define our own loss function and gradient function.

We consider using the error rate of decoding result as the loss function which means the predicted targets in each epoch will go through a full decoding process. Since the error rate of decoding is only related to the predicted value but irrelevant to the truth value, the truth target value is non-essential. Hence, over-fitting is not a problem in our case, since we don't have a typical training data set, our goal is making the loss, which is the error rate, as low as possible. Besides, we choose the gradient function to be the partial derivative of loss function with respect to each weight value.

Unfortunately, Keras only supports custom loss functions that consist of pure tensor operations and our decoding algorithm apparently can not be represented by tensor operations, we have to choose another machine learning frame. We end up with a simple customized gradient descent frame. Though putting a lot of time to optimize the error rate calculation, each epoch run still takes too long to get some promising results.

5 EXPERIMENTS

As an educational tool, we evaluate our work in three aspects. First, we ask several users having various levels of familiarity with quantum computing to follow the tutorial and see how they learn from this project. We also ask them for potential improvement of the system. Second, we try to use this tool to explore the sub-optimal behavior of naive decoder to get some early understanding of it. Third, we evaluate our decoder performance by running simulations and get the relationship between physical error rate and logical error rate under different code distances.

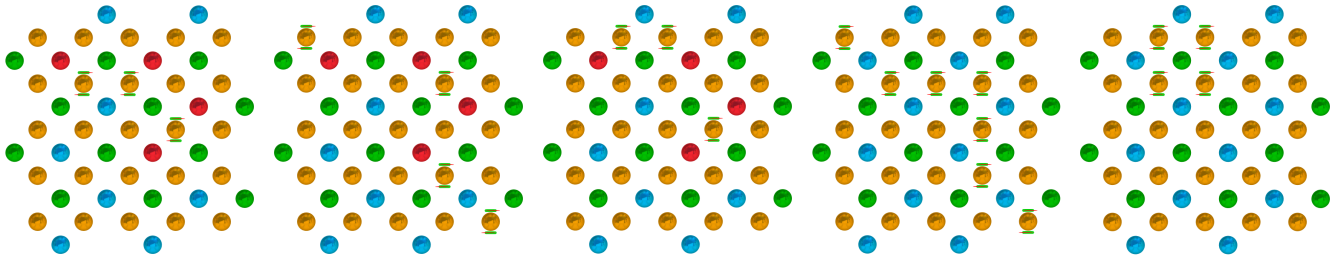
5.1 Tutorial Perception

We asked 8 people to use this interactive tutorial, among which 2 people are not familiar with quantum computing, 3 people have medium level familiarity to quantum computing and 3 people are familiar with quantum computing. Each of them expresses their usage experience of the tutorial, and also suggest some improvements.

Not Familiar. They expressed that the GUI is nice and interesting. One person said that because of the limitation of knowledge of quantum mechanics, he didn't understand too much about the decoder. Another person spent some time to get familiar with the error display mode and finally learn it by himself. It makes sense that one without knowledge of quantum mechanics cannot learn everything well within just one tutorial in several minutes, but we're glad that they all successfully finished the tutorial and have some preliminary understanding of quantum error correction code.

Medium Familiarity. They all expressed favor in this tool, both GUI and its content. One person said that it's a fantastic tool and he's going to "play around with this on my own time just to play with everything and learn more". He thought "this does a very good job walking you through the tool and teaching you the specifics about decoding", and he also like that "you have the option to read through all the quantum stuff or just go straight into the decoding". Another person major in physics but without knowledge about quantum computing said that the content is sufficient for beginners to quantum computing, and it's very meaningful in education.

Familiar. They're impressed by the fancy GUI based on Web and also the interactive tutorials to introduce everything clearly. Since they have already read plenty of papers about quantum error correction and surface code, they are surprised by how this tool can introduce it clearly in just several minutes with funny interactions. One person said "you have given lot of effort in making it", indeed,



(a) Error Pattern (b) Naive Correction (c) MWPM Correction (d) Naive Corrected (e) MWPM Corrected

Figure 9: Comparison between Naive Decoder and MWPM Decoder.

we spent lots of time to convey the core information of surface code without introducing too much underlying physics. This is only feasible with step-by-step interactive tutorial that maps abstract concepts to the concrete representation of the 3D model. Another one shows curiosity about decoding process which is not included in the tutorial. For expert users, this educational tool is impressive but doesn't provide enough depth into the hard problem of decoding algorithm.

5.2 Decoder Behavior Exploration

We demonstrate how this tool can work beyond just an educational tool, in assisting developers to explore their new proposed surface code decoders. We use naive decoder as an example, and find its limitations by comparing it with MWPM decoder. After testing several error patterns, we found a special pattern that causes decoding error with naive decoder but works fine with MWPM decoder, as shown in Fig. 9.

Given the error pattern in Fig. 9a, we can see that there are four error points (red sphere). Naive decoder mistakenly matches the left-top error point to the boundary in Fig. 9b, but MWPM decoder matches correctly in Fig. 9c. The naive decoder introduces a logical Pauli X operator after correction in Fig. 9d because there is a sequence of Pauli X operator connecting the left boundary to the right boundary. The MWPM decoder on the other hand, corrects those errors successfully without introducing any logical operators, as shown in Fig. 9e. With the help of this visualization, we understand that this happens because naive decoder depends heavily on the search sequence. When the top-left error points hits the left boundary, it matches with it without considering the other part of the system. Although this error points reaches its minimum path, the other part of the errors end up with using longer path to do the perfect matching, which is less likely in practice. As see the result, the solution of naive decoder in Fig. 9b predicted 4 data qubit errors but MWPM predicts 3 in Fig. 9e. In a word, naive decoder tries to find a local optimal but MWPM considers global optimal.

5.3 Decoder Performance

We evaluate the performance of naive decoder compared to MWPM decoder. To do so, we add i.i.d. (independent and identically distributed) Pauli X errors to the data qubits and count the failed error correction cases of Z stabilizers. Because the decoding of Z stabilizers and X stabilizers are independent, the ultimate error rate would be no more than 2 times the error rate with single type of errors.

The error rate of naive decoder is shown in Fig. 10. It shows some abnormal behaviors when code distance increasing from 3 to 5 and from 9 to 11. When the code distance changing from 3 to 5, logical error rate becomes even higher, which is not expected because larger code should come with lower error rate when physical error rate is below the fault tolerant threshold. The improvement from 9 to 11 is abnormally small, which is much smaller than the improvement from 7 to 9. Apart from these abnormal behaviors, the critical problem is that the logical error rate decreases too slow with code distance, making it impractical in making a fault tolerant quantum computer. We do not see any clue indicating that the logical error rate is decreasing exponentially with code distance, so this decoder may not be able to provide arbitrary low error rate practically. This suboptimality matches with our early exploration of the tutorial.

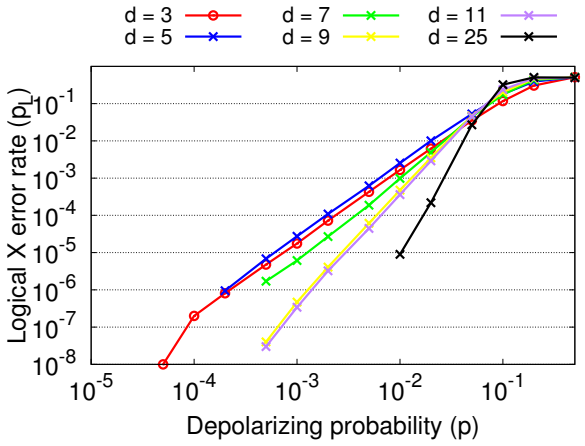


Figure 10: Naive Decoder Performance

The error rate of MWPM decoder is shown in Fig. 11. The implementation of this MWPM is a preliminary one using a python graph library. To take advantage of existing library to the maximum, we add much more nodes to the graph without considering the special attributes of surface code, and thus the complexity becomes high (but still polynomial). This makes our implementation of MWPM decoder only feasible for small code distances. From the data of small code distances, we can find that the logical error rate is indeed decreasing exponentially with code distance, which is the correct behavior of a fault tolerant decoder. Compare with naive decoder, it

can solve the matching problem optimally and thus behaves better in some common cases.

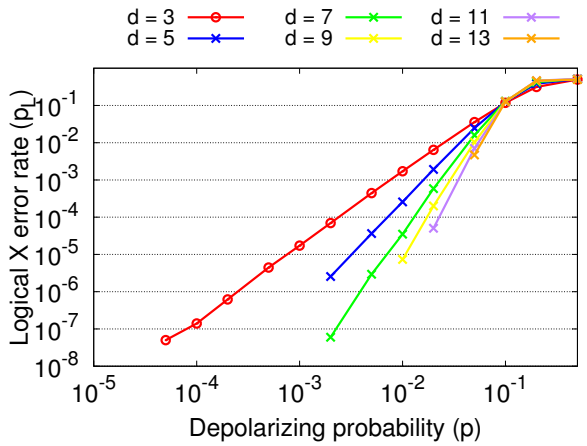


Figure 11: MWPM Decoder Performance

Currently we don't have results for weight-optimized MWPM decoder based on machine learning, because the training time is unacceptably long to get. We'll leave this as future work.

6 CONCLUSION AND FUTURE WORK

We propose an educational tool to demonstrate how surface code as an quantum error correction code works by leading users through an interactive tutorial. This approach is appreciated by people with various levels of familiarity in quantum computing. This educational tool also helps us visualize and explore our new sub-optimal "naive decoder" by comparing with the baseline MWPM decoder. We further evaluate the decoder performance and it matches exactly with the early exploration of the decoder.

We plan to further improve this educational tool by taking suggestions from our early users, for example, adjusting the layout of tutorial messages and elaborating on the detailed decoder algorithm descriptions. We also leave the weight-optimized MWPM decoder as future work so that we can know whether this approach is feasible or not.

REFERENCES

[1] [n.d.]. Quantum Error Correction. https://en.wikipedia.org/wiki/Quantum_error_correction. Accessed on 2020-12-06.
[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.
[3] Felix Bloch. 1946. Nuclear induction. *Physical review* 70, 7-8 (1946), 460.
[4] Héctor Bombín. 2013. An introduction to topological quantum codes. *arXiv preprint arXiv:1311.0277* (2013).
[5] Héctor Bombín and Miguel A Martin-Delgado. 2007. Optimal resources for topological two-dimensional stabilizer codes: Comparative study. *Physical Review A* 76, 1 (2007), 012305.
[6] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. 2003. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 59–68.
[7] Andrew Cross. 2018. The IBM Q experience and QISKit open-source quantum computing software. *APS* 2018 (2018), L58–003.
[8] Nicolas Delfosse and Naomi H Nickerson. 2017. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218* (2017).
[9] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. 2002. Topological quantum memory. *J. Math. Phys.* 43, 9 (2002), 4452–4505.
[10] Andrew J Ferris and David Poulin. 2014. Tensor networks and quantum error correction. *Physical review letters* 113, 3 (2014), 030501.
[11] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* 86, 3 (2012), 032324.
[12] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. 2012. Towards practical classical processing for the surface code. *Physical review letters* 108, 18 (2012), 180501.
[13] Lisa Hales and Sean Hallgren. 2000. An improved quantum Fourier transform algorithm and applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 515–525.
[14] Kevin Hartnett. 2019. Quantum supremacy is coming: here's what you should know. https://www.quantamagazine.org/quantum-supremacy-is-coming-heres-what-you-should-know-20190718/. Accessed on 2020-12-03.
[15] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. 2012. Surface code quantum computing by lattice surgery. *New Journal of Physics* 14, 12 (2012), 123011.
[16] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. 2018. Differentiable particle filters: End-to-end learning with algorithmic priors. *arXiv preprint arXiv:1805.11122* (2018).
[17] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. 2007. *An introduction to quantum computing*. Oxford university press.
[18] A Yu Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of Physics* 303, 1 (2003), 2–30.
[19] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.
[20] etc. Smite-Meister. 2009. Bloch sphere. https://en.wikipedia.org/wiki/Bloch_sphere. Accessed on 2020-12-05.
[21] Yu Tomita and Krysta M Svore. 2014. Low-distance surface codes under realistic quantum noise. *Physical Review A* 90, 6 (2014), 062320.
[22] Giacomo Torlai and Roger G Melko. 2017. Neural decoder for topological codes. *Physical review letters* 119, 3 (2017), 030501.