An Overview of Machine Learning Methods used in Quantum Error Correction

Adittya Patil

May 6, 2024

Abstract

In the process of Quantum Error Correction, decoding is an integral part of correcting errors as it allows us to obtain accurate estimates of the errors in our quantum computers, which subsequently affect the accuracy of the computations which we perform on these devices. This means that we must have smooth decoders for our error correction schemes, which not only accurately detect errors but also do so in an efficient manner. Several machine learning based decoding models have been introduced recently for error models such as surface codes and Heavy Hexagonal (HH) codes.

1 Introduction

Quantum computing is growing to one of the world's premiere technologies. Naturally, several models of quantum computation have arisen, but each comes with its own tradeoffs. Each model has its own form of error correction. For example, one type which is being worked on by many efforts - and the primary error correcting code which we will be surveying in this work - are surface codes, which act on a 2-dimensional square lattice. Surface codes are the forefront of topological quantum since their popularity comes from their ability to be embedded on different topologies. Because of this, there are several families of surface codes already being implemented on newly-developed hardware, such as planar, toric, and hyperbolic codes.

One integral part of implementing these error correcting codes is having an accurate and efficient decoding method to correct detected errors within the systems. Many popular approaches may not necessarily provide a comprehensive approach to all noise models. For example, Minimum-Weight-Perfect-Matching (MWPM) is an extremely popular decoding algorithm used for finding the most probable set of errors given a set of syndrome measurements. These measurements are recorded from a set of detectors and ultimately output a set of logical X-flip or Z-flip errors. Figure 1 shows a depiction as to what errors MWPM takes into account.

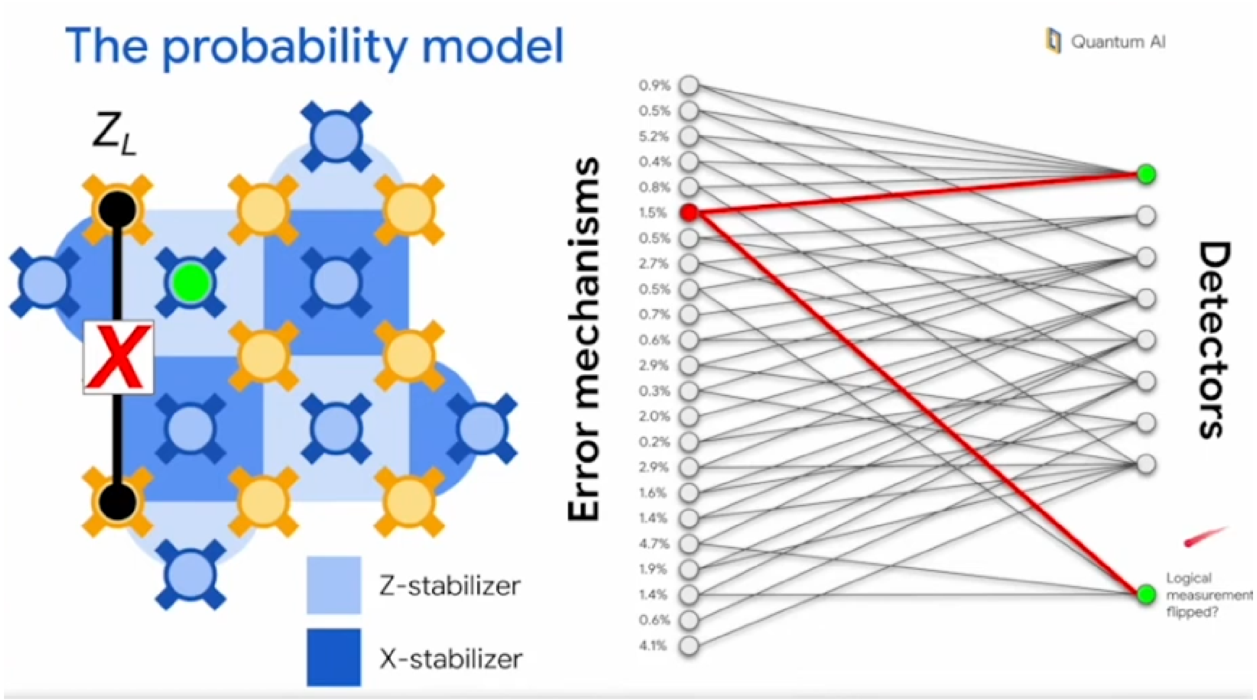


Figure 1: The probability model for the MWPM Decoder. We have a bipartite graph consisting of nodes corresponding to error mechanisms and a set of detectors. In this case, the detectors correspond to pairs of measurements which would agree when no error is present. A specific error mechanism is connected to the detectors which it will activate and a detector will be activated if it is incident to an odd number of occurring error mechanisms. The node in the lower left hand corner signifies if the logical measurement is flipped. In this particular example, an X error on the middle far-left data qubit flips the combined parity of the data qubits on the left hand side outlined in black. Figure taken from [1].

MWPM is considered the standard for surface code decoding because they are considered to have a good balance between the decoding accuracy and speed.

Another similar approach is the Union-Find (UF) decoder, which utilizes the Union-Find (also known as Disjoint-Set) data structure in order to dynamically keep track of and update the estimation of the error while the decoder runs. This allows it to achieve a linear time complexity.

A pitfall of these decoders is that, because they only account for a few different sources of error, identical syndromes can originate from different error sources, meaning decoding algorithms that evaluate syndromes collectively are necessary. Because of this, such methods are the centerpiece for many decoder designs which take additional steps to account for other sources of errors. This potentially can be difficult and introduce additional computational overhead.

Machine learning based decoders aim to solve such problems. In this work, we will survey several different decoder architectures which utilize neural networks for decoding surface codes as well as how they can account for several different sources of noise. Additionally, we will look at how these architectures compare to some of the methods discussed above as well as some of their disadvantages and potential improvements to their design.

2 Materials and Methods

2.1 Transformer Nerual Network

2.1.1 Architecture

The most effective examples so far of machine learning based decoders come in the form of Transformer Neural Networks (TNNs). In this section, we will be analyzing the works of two different research groups who built and tested their own TNN-based decoders. Although their architectures and experiments are very similar, their reasons for developing these decoders are different. Google Quantum AI wants to develop TNN-based decoders because of their ability to incorporate a decoder that draws from multiple types of error. The group from [2] analyzes their solution in the broader regime of machine learning-based decoders, arguing that the Transformer model allows for efficient transfer learning, which allows the decoder to work on multiple code distances without retraining.

One of the centerpieces of Google Quantum AI's QEC efforts is developing a Recurrent, Transformer-Based Neural Network to decode surface codes [3]. The architecture of this network is shown in Figure 2
The key ingredient for decoding syndromes here is the Syndrome Transformer, which updates the decoder state by passing information between stabilizer representations in a learned, structured manner. It combines multi-headed attention from traditional transformer networks with an attention bias as well as spatial convolutions. This system gives the TNN part of the decoder the ability to map which kinds of device-level errors correspond to the respective errors they can cause, thus allowing it to "learn" to modulate the information flow between stabilizer representations based on their physical relationship and type. Because of this, the network is able to adapt to account for many different sources of error so long as the appropriate data from each error source is fed through the TNN. Furthermore, the decoder's state can be processed by a readout network to predict whether an error has occurred or not for real-time quantum computation.

In addition to Google Quantum AI, a group of 8 researchers in [2] built a similar architecture and ran their own independent experiments. The key difference is that these researchers gave more details about how they trained their model. They used a mixed-loss approach during their model training, which combines the loss from the local physical error of each qubit with the loss from global parity prediction Figure depicts a high-level overview of the data processing pipeline used for the decoder.

This group discussed and used the physical error rates of the devices they ran their experiments as a changeable parameter for testing purposes. They ran experiments which related physical error rates to logical error rates on their physical devices in addition to running experiments comparing logical error rate to code distance.

2.1.2 Results and Areas of Improvement

Figure 4 demonstrates two important experiments which Google ran in their testing of their Recurrent TNN. We can see from Plot (A) that the TNN performed better for code distances 3 to 11 compared to both MWPM-Corr and Pymatching, which are two MWPM-based decoders.

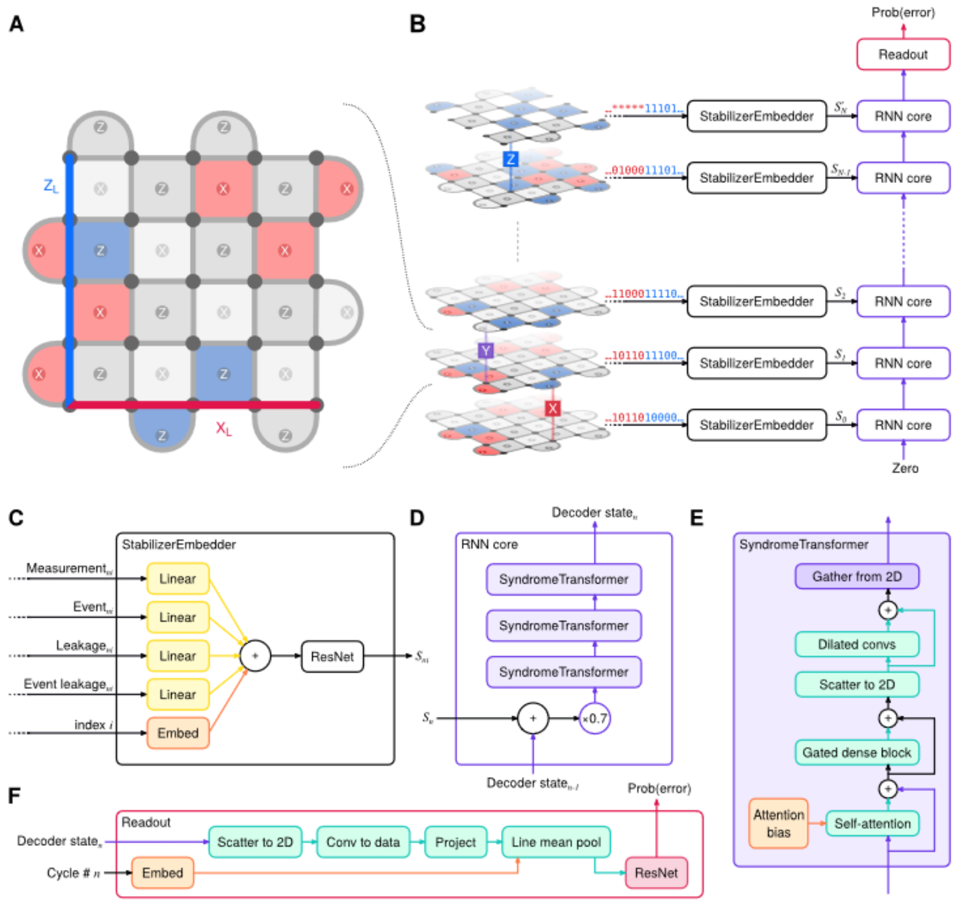
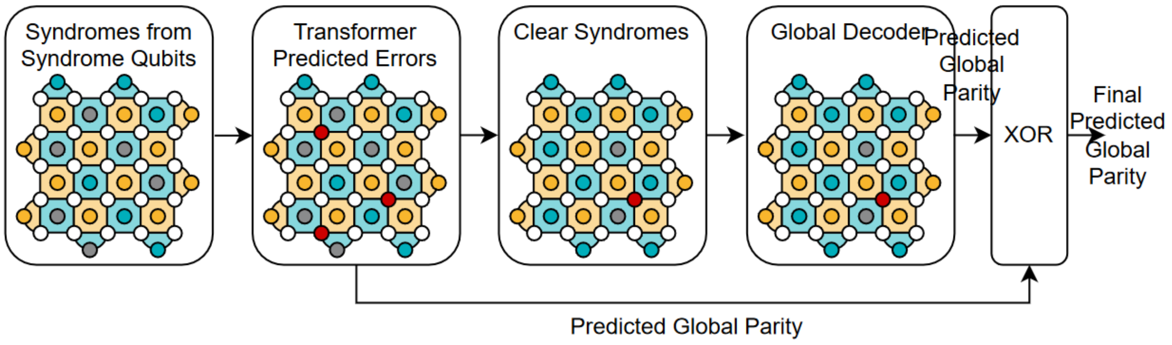


Figure 2: The process of decoding a 5 x 5 rotated surface code using a Recurrent Transformer Neural Architecture. (A) depicts a 5 x 5 rotated surface code, where X and Z stabilizer qubits are labelled by light grey dots or blue/red when a parity check violation is detected. The logical Pauli Operators Z_L and X_L are represented on the left and bottom grid edges, respectively. (B) At each timestep the recurrent network updates the decoder state and incorporates new stabilizers as necessary (C) depicts the creation of an embedding vector S_{ni} for each new stabilizer. (D) The decoder state gets updated through three Syndrome Transformer layers. Each block of the recurrent network combines the decoder state and the stabilizers S_n for one cycle, which is scaled down by 0.7. (E) Each Syndrome Transformer layer updates the stabilizer representations here. (F) Logical errors are predicted from the final decoder state. Diagram from [3].

We can see from Plot (A) that in the soft case, where the TNN is only compared against MWPM-Corr. It outperforms MWPM-Corr for code distances 3-9, even with leakage. At code distance 11, MWPM-Corr overtakes it. In the hard case, The TNN with leakage and without leakage outperform MWPM-Corr and Pymatching. In this case, leakage is incorporated by changing the leakage parameter from zero to another unspecified number and running the decoder again. It can also be noted that, when code distance is increased, the TNN with leakage incorporated actually did slightly better than the TNN without leakage.

In Plot (B), the logical error rates are compared against other MWPM-based decoders



Syndromes from Syndrome Qubits
Transformer Predicted Errors
Clear Syndromes
Global Decoder
Predicted Global Parity
XOR
Final Predicted Global Parity
Predicted Global Parity

Figure 3: A diagram of the data processing pipeline in [2]. for each individual timestep that the decoder runs for. once the Syndrome has been embedded and predicted by the transformer and sent to an XOR function. From here, the syndromes are cleared and processed by a decoder, then sent to the same XOR function. The XOR of these two predicted global parities is computed to output a final predicted global parity.

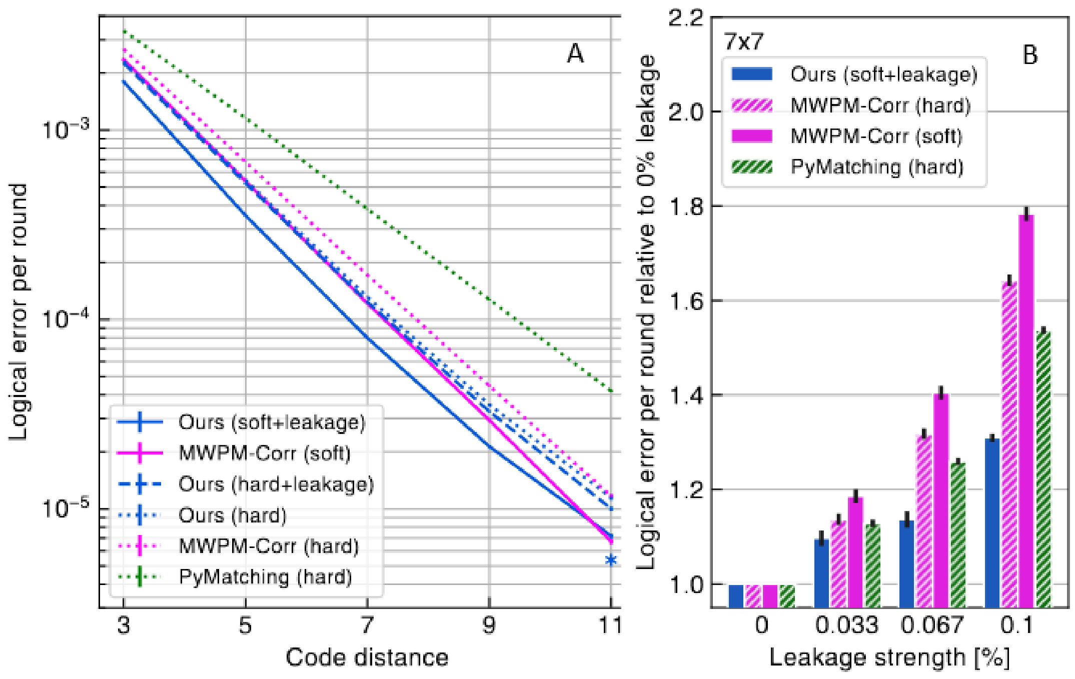
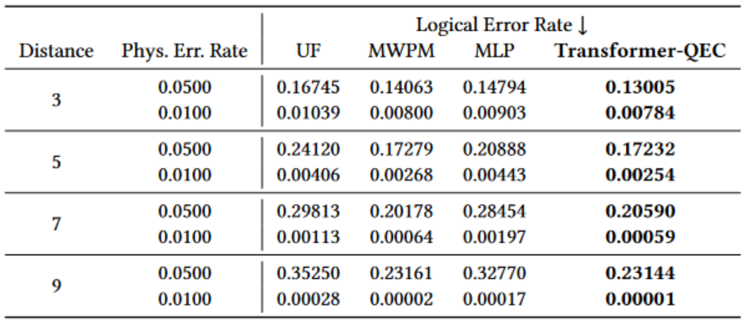


Figure 4: The data generated for these plots is data from an SI1000 transformer [4] with different In-phase and Quadrature (I/Q) noise parameters. (A) Logical Error per Round vs Code Distances from 3 to 11 for Google's TNN decoder and several examples of MWPM decoders. (B) analyzes the impact that leakage at various strengths has on code distances of 7. In this case, soft refers to readout provided by In-phase and Quadrature (I/Q) readout and hard refers to readout determined by thresholded binary values [3].

in terms of difference leakage strengths. In this case, only the soft case with leakage is taken into into account, whereas the Pymatching only uses the hard case as from Plot (A) and MWPM-Corr is tested on both the hard and soft case without any leakage. For this, the TNN consistently outperforms each of the other decoders at each variance of leakage strength.

We can now look at the data retrieved by the Group from [2].



Distance	Phys. Err. Rate	Logical Error Rate ↓			
		UF	MWPM	MLP	**Transformer-QEC**
3	0.0500	0.16745	0.14063	0.14794	**0.13005**
	0.0100	0.01039	0.00800	0.00903	**0.00784**
5	0.0500	0.24120	0.17279	0.20888	**0.17232**
	0.0100	0.00406	0.00268	0.00443	**0.00254**
7	0.0500	0.29813	0.20178	0.28454	**0.20590**
	0.0100	0.00113	0.00064	0.00197	**0.00059**
9	0.0500	0.35250	0.23161	0.32770	**0.23144**
	0.0100	0.00028	0.00002	0.00017	**0.00001**

Figure 5: A comparison of different decoders at code distances of 3, 5, 7, and 9 which include UF, MWPM, MLP, and the Transformer-QEC models. Each of these decoders is tested with two different physical error rate parameters: 0.05 and 0.01.

We can see from the above results that the Transformer-QEC architecture outperforms each of the other decoding models in every case in Figure 5 for each code distance and physical error rate. Additionally, from Figure 6, we can see that as code distance increases, the logical error rate is actually smaller below the threshold.

From these results, the biggest piece of information I would like to understand more from both papers is how much more overhead to TNN networks take in comparison to decoders such as MWPM and UF. In these two works, it has been shown that TNN's can be more accurate compared to traditional decoders. However, if they are only marginally more accurate but require a lot more computational overhead, it may not be worth using them because they would slow down the real-time computation speed by a significant factor. To do this, I would set up experiments which count the number of FLOPs for several rounds of decoding at varying code distances for each decoder while varying parameters such as noise and leakage. Aditionally, I would like to vary parameters such as the number of attention heads that are used in the TNNs or the number of training rounds to see if the overhead decreases and still produces similar results. To do this, I tried looking into codebases and also emailed one of the authors of the Google paper to see if the code can be open-sourced, to which he replied no since the hardware resources they used for testing are closed-source, so the code would not work at all.

One thing I wish I saw more from the group at Google was a discussion comparing the logical error rates to the physical error rates. Although they addressed concerns about including other sources of error such as leakage in their decoding, having a discussion and quantifying results with physical error as a factor is important in being able to directly compare the TNN model with other decoding methods. If I had access to Google's experimental

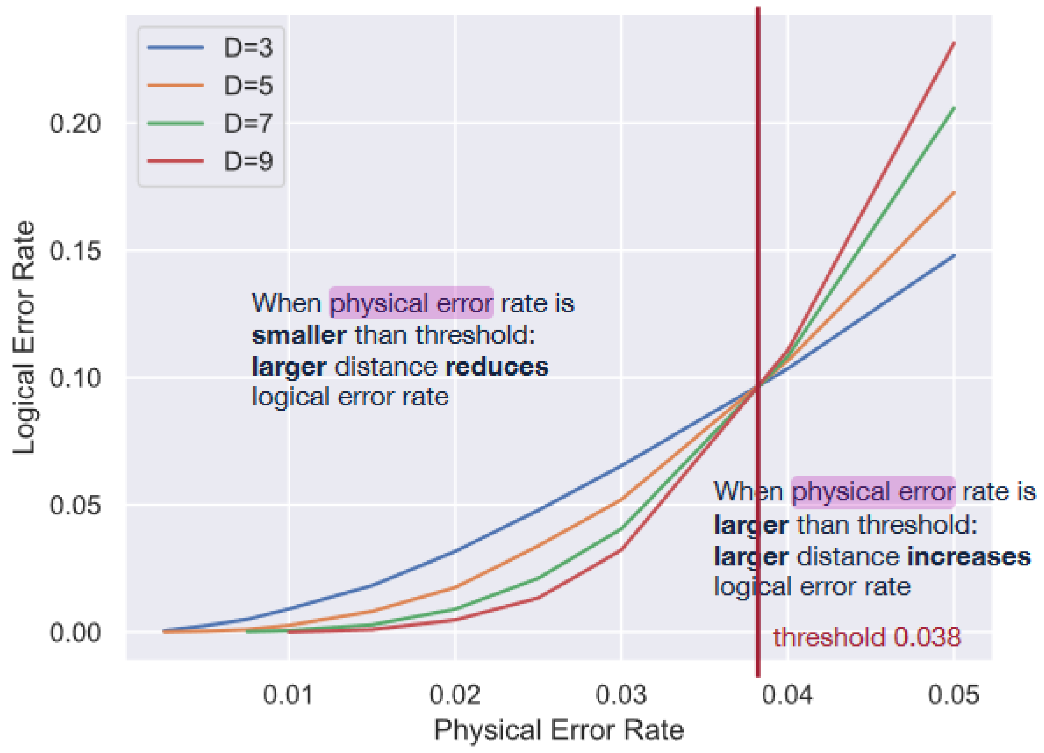


Figure 6: A comparison between Logical Error Rate and Physical Error Rate. Code distances of 3, 5, 7, and 9 are evaluated here and a threshold of 0.038 is determined for when larger code distance increases the logical error rate.

code, I would also run similar results while changing physical error rate as a parameter and testing it against other MWPM decoders similar to their tests with leakage as a parameter and publish these results as part of the paper.

2.2 Artificial Neural Networks for Heavy Hexagonal Codes

2.2.1 Architecture

The paper we will be looking at in this section demonstrates a practical implementation of an Artificial Neural Network (ANN) which decodes the Heavy-Hexagonal (HH) code. The HH Code is subsystem stabilizer code on the Heavy-Hexagonal lattice that combines Bacon-Shor and surface-code stabilizers. It encodes $n = \frac{5d^2-2d-1}{2}$ physical qubits into a logical qubit. In this case, an ANN is a more traditional feed forward neural network. This is unlike the transformer, which has a complex architecture consisting of multiple processing steps within each layer. Figure 7 demonstrates the data processing pipeline that was used in decoding HH codes in [5].
The authors further discuss how the ANN uses dense layers, which means that each neuron within each layer is intricately connected with each neuron in the previous layer. Because of this, the first question I would ask is that if this is necessary since some pathways may never be used due to some detection events not usually occurring, which makes the MWPM

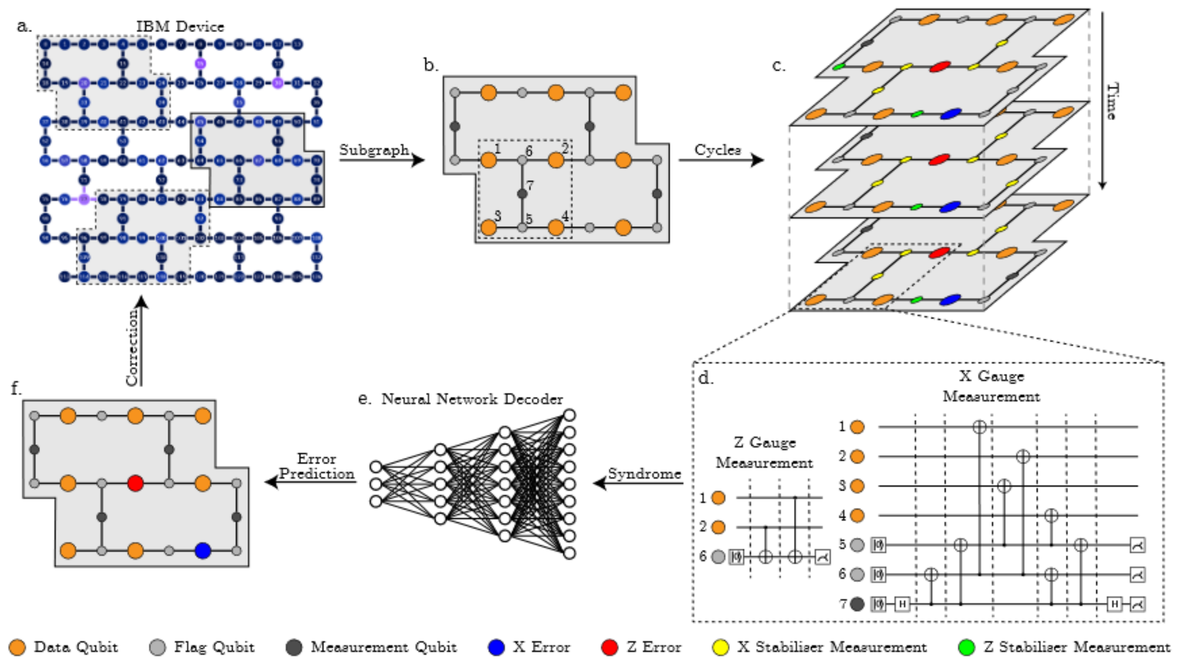


Figure 7: The data processing pipeline for decoding HH codes on several IBM Machines. The Heavy-Hexagonal lattice is the qubit connectivity architecture which several IBM machines use. (A) displays the lattice connectivity of the qubits in these machines. (B) demonstrates the subgraph of HH qubits being inspected for potential errors. (C) Shows the subgraph being inspected over multiple time steps. (D) demonstrates the HH error syndrome measurement, done via the X- and Z-gauge measurements, in the presence of state preparation, readout, and idle qubit errors. (E) demonstrates this data being passed into the ANN for decoding and a prediction of the error(s) which occurred. (F) Shows a possible solution for the corrected error, which is applied to the IBM device. Diagram taken from [5]

model much more effective for its use case because each node in the set of detectors is not connected to each of the error mechanisms. Although the authors did not disclose the specifics of the neural network as well as not having a codebase for this project, I would like to experiment with the number of layers, neurons, and overall parameters that are being used for the network to seek out the most efficient combination.

2.2.2 Results and Areas of Improvement

Figure 8 demonstrates the Logical vs. Physical Error Rates for the tests that were run as well as for specific IBM devices.
This can be seen by each of the horizontal lines on each point, which have a wide range of error. Although not shown in the plots, the threshold for when the overall trend line for code distance 5 is less than code distance 3 is about 0.005. We can see that none of the recorded points are anywhere close to this range. One key point that the authors of this paper make is that the physical error rates on IBM's devices exceed the threshold for restrict the ability

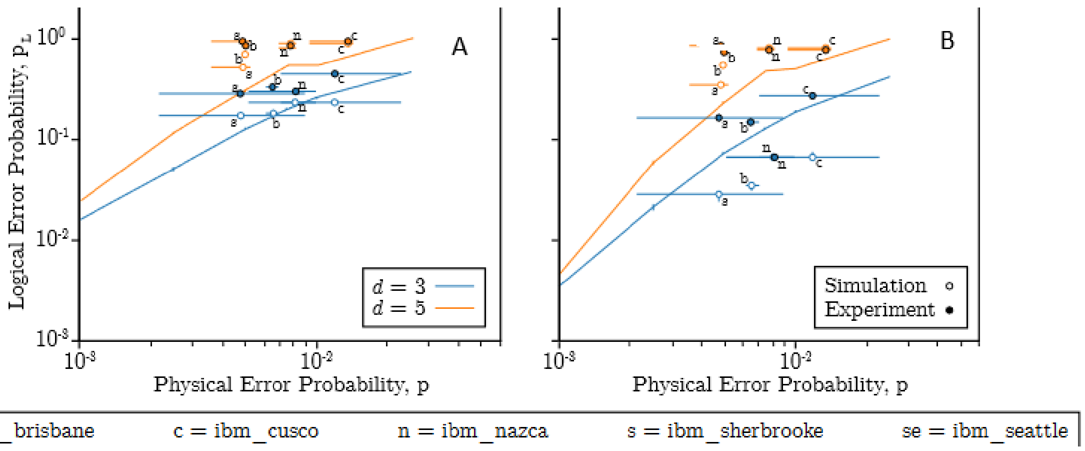


Figure 8: The Logical vs. Physical Error Rates for the tests run for code distances 3 and 5. (A) corresponds to X-type Errors and (B) corresponds to Z-type Errors. Additionally, each of the devices listed on the bottom are plotted from both simulation and experiment. These points symbolize the error models for each of the devices. The horizontal line on each of the points demonstrates the range of overall error rates of each possible sub-graph location.

of the ANN decoder to suppress logical error rates.

3 Conclusion

In this work, we looked at a few different examples of machine learning being used in decoding Quantum Error Correcting Codes. We started off by looking at papers which demonstrate Transformer Neural Network (TNN) architectures to decode Surface codes, which yielded great success in terms of increasing the accuracy of detecting errors and adapting to include other sources of errors in qubits. We then looked at a practical implementation of an Artificial Neural Network (ANN) which was used to decode Heavy Hexagonal (HH) codes. There is still a lot of work to be done in terms of quantifying and improving the efficiency of these methods in order to prove that they are acutally effective in decoding these codes, but significant progress is being made in ensuring that they are accurate.

I would like to thank Professor Jonathan Baker in his guidance for helping me to understand the problem at hand, what is important to consider when evaluating error-correcting methods, and teaching the ECE 382V class, which gave me a different perspective on quantum computing as a whole. I would also like to thank my ECE 382V classmates for pushing and encouraging me to look at things from a different perspective and to develop an understanding for the material presented throughout the semester.

References

[1] Google Quantum AI. Decoding experimental surface code data, Sep 2022.

[2] Hanrui Wang, Pengyu Liu, Kevin Shao, Dantong Li, Jiaqi Gu, David Z. Pan, Yongshan Ding, and Song Han. Transformer-qec: Quantum error correction code decoding with transferable transformers, 2023.

[3] Johannes Bausch, Andrew W Senior, Francisco J H Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, Murphy Yuezhen Niu, Sam Blackwell, George Holland, Dvir Kafri, Juan Atalaya, Craig Gidney, Demis Hassabis, Sergio Boixo, Hartmut Neven, and Pushmeet Kohli. Learning to decode the surface code with a recurrent, transformer-based neural network, 2023.

[4] Tech Power Components.

[5] Brhyeton Hall, Spiro Gicev, and Muhammad Usman. Artificial neural network syndrome decoding on ibm quantum processors, 2023.