Decoding Small Surface Codes with Feedforward Neural Networks

Savvas Varsamopoulos,[1,2] Ben Criger,[1,2] and Koen Bertels[1,2]
[1]*Computer Engineering, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands*
[2]*QuTech, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands*
(Dated: October 16, 2017)

Surface codes reach high error thresholds when decoded with known algorithms, but the decoding time will likely exceed the available time budget, especially for near-term implementations. To decrease the decoding time, we reduce the decoding problem to a classification problem that a feedforward neural network can solve. We investigate quantum error correction and fault tolerance at small code distances using neural network-based decoders, demonstrating that the neural network can generalize to inputs that were not provided during training and that they can reach similar or better decoding performance compared to previous algorithms. We conclude by discussing the time required by a feedforward neural network decoder in hardware.

I. INTRODUCTION

Quantum computing has emerged as a solution to accelerate various calculations using systems governed by quantum mechanics. Such calculations are believed to take exponential time to perform using classical computers. Initial applications where quantum computing will be useful are simulation of quantum physics [1], cryptanalysis [2, 3] and unstructured search [4], and there is a growing set of other quantum algorithms [5].

Simple quantum algorithms have been shown to scale better than classical algorithms [6–8] for small test cases, though larger computers are required to solve real-world problems. The main obstacle to scalability is that the required quantum operations (state preparations, single- and two-qubit unitary gates, and measurements) are subject to noise, therefore quantum algorithms cannot run with perfect fidelity. This requires quantum computers to use active error correction [9, 10] to achieve scalability, which in turn requires a classical co-processor to infer which corrections to make, given a stream of measurement results as input. If this co-processor is slow, performance of the quantum computer may be degraded (though recent results [11] suggest that this may be mitigated).

The remainder of this paper is organized as follows. In Section II, we outline the relevant aspects of quantum error correction and fault tolerance. We discuss the need for a fast classical co-processor in Section III. In Section IV, we give a brief summary of existing techniques to perform decoding quickly, and follow this in Section V with the introduction of a new technique based on feedforward neural networks. We examine the accuracy of the proposed decoder in Section VI, and conclude by discussing its speed in Section VII.

II. QUANTUM ERROR CORRECTION

While it is often possible to decrease the amount of noise affecting a quantum operation using advanced control techniques [12, 13], their analog nature suggests that some imperfection will always remain. This has driven the development of algorithmic techniques to protect quantum states and computations from noise, which are called *quantum error correction* and *fault tolerance*, respectively.

Quantum error correction replaces unprotected qubit states (e.g. $|0\rangle$, $|1\rangle$) with specially encoded multi-qubit states (typically called $|\bar{0}\rangle$, $|\bar{1}\rangle$), accompanied by a set of *logical operators* $\bar{U}$

transforming states in the subspace defined by $|\bar{0}\rangle$ and $|\bar{1}\rangle$. Quantum codes typically ensure that random operations E acting on fewer than d qubits cannot transform one encoded state into another ($\langle\bar{0}| E |\bar{1}\rangle = 0$), where d is called the *code distance* [14, 15]. Typically, these random operations are taken to be *Pauli operators*, whose names and effects on single-qubit states are given below:

$$X : \begin{array}{c} |0\rangle \mapsto |1\rangle \\ |1\rangle \mapsto |0\rangle \end{array}, \; Z : \begin{array}{c} |0\rangle \mapsto |0\rangle \\ |1\rangle \mapsto -|1\rangle \end{array}, \; Y : \begin{array}{c} |0\rangle \mapsto i\,|1\rangle \\ |1\rangle \mapsto -i\,|0\rangle \end{array} \tag{1}$$

These operators form a convenient basis for the space of possible errors; codes which can correct these errors on a subset of qubits can correct arbitrary errors on the same subset [15, Chapter 2].

Often, the encoded states are chosen to be in the mutual $+1$ eigenspace of a set of multi-qubit Pauli operators, called *stabilisers*, resulting in a *stabiliser code*. Projective measurements of the stabilisers result in output bits, called *syndromes*, which are used by a reliable classical co-processor to determine which error has occurred, a process called *decoding*. The use of stabiliser codes can effectively reduce the probability of error from transmitting a qubit through a noisy channel (though *logical errors* can still occur, acting as $\bar{X}$, $\bar{Z}$, or $\bar{Y}$). This reduction in the probability of error is obtained when operations are perfect, however, quantum error correction is not enough on its own to guarantee that computation can be performed with a low probability of error when using noisy operations.

To suppress errors from the physical operations themselves, it is necessary to design logical operations which act directly on encoded states (i.e. without first transforming the encoded states to bare qubit states), in such a way that random errors affecting physical operations are likely to result in correctable errors with respect to the underlying code. Operations which have this property are called fault-tolerant [16]. Fault-tolerant syndrome measurements can be applied repeatedly to correct time-dependent errors, which occur continuously as computation proceeds. There are many schemes for attaining fault tolerance, based on different families of quantum codes, and using different techniques for ensuring noise from imperfect state preparation and stabiliser measurement remains suppressable.

Each fault tolerance scheme has a *threshold error rate*, beneath which there exists a code in the associated code family which can suppress errors to exponential accuracy, using a polynomially-large number of qubits and operations [17]. Each code in such a family also typically has a *pseudo-threshold*, an error rate at which encoded operations using that specific code provide higher accuracy than is possible using bare qubits/operations. These figures of merit are used to characterize fault tolerance schemes, and are especially important when considering near-term implementations of these schemes.

One scheme which has a relatively high threshold error rate uses *surface codes* [18–21], stabiliser codes whose stabilisers are supported on qubits which are adjacent on a 2D square tiling (see Figure 1). This approach also allows the use of exclusively planar connections between qubits, and uses at most four connections between each qubit and its neighbours (see Figure 2). These features make surface codes especially attractive for near-term implementation.

To complete such an implementation and analyse its performance, it is also necessary to specify the method by which surface codes are to be decoded. Syndromes obtained by measuring surface code stabilisers have a special mathematical structure, which leads to a polynomial-time decoding algorithm. These syndromes occur at the endpoints of continuous one-dimensional chains of errors if stabiliser measurement is performed with perfect operations, and differences between consecutive syndromes occur at the endpoints of one-dimensional chains of data/measurement errors if realistically noisy operations are used (see Figure 3). If error rates are low, then the smallest error which conforms with the syndrome is likely a valid correction. To find it requires the classical co-processor to minimize the sum of the lengths of chains connecting pairs of syndrome changes, a problem known as *minimum-weight perfect matching* [25]. This problem can be solved

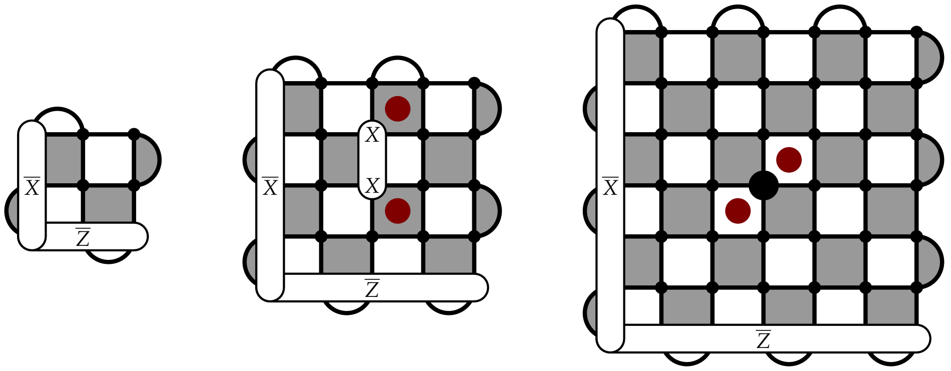


FIG. 1. Surface codes with distances 3, 5, and 7, respectively. Data qubits are placed at the corners of the square tiles, on which the stabilisers are supported. White and grey squares support stabilisers of the form $X^{\otimes 4}$ and $Z^{\otimes 4}$, respectively. White and grey semi-circles support stabilisers of the form $X^{\otimes 2}$ and $Z^{\otimes 2}$, respectively. Logical operators are supported on continuous paths that cross the tiling from side to side; shorter paths form detectable errors (syndromes shown in red). Ancilla qubits placed inside the tiles can be coupled to neighbouring data qubits and measured to effect indirect stabiliser measurement.

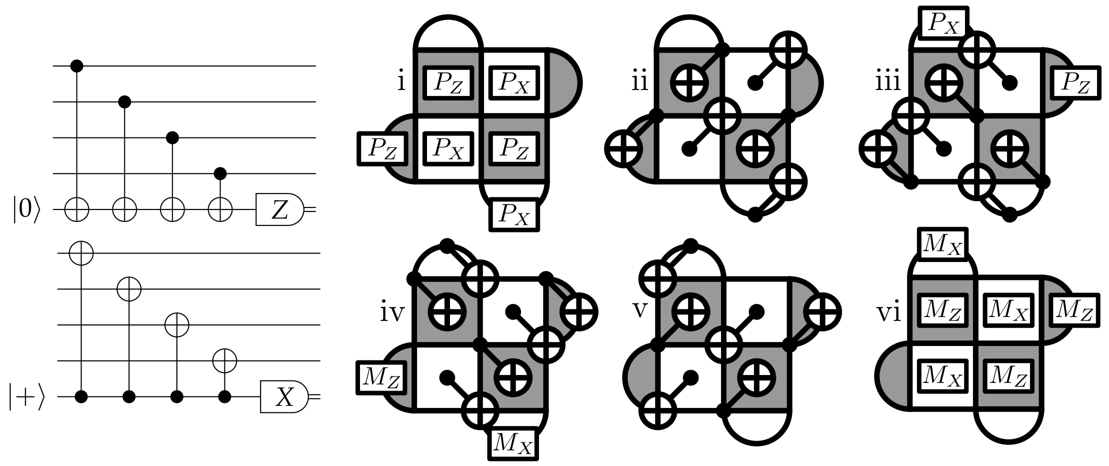


FIG. 2. Stabiliser measurement circuit for the distance-3 surface code [22–24]. Left: Measurement circuit for individual Z tiles (top) and X tiles (bottom), including an ancilla qubit to be placed at the center of each tile. Ancilla qubits are prepared in the $+1$-eigenstate of the appropriate basis, four CNOT gates are executed, and the ancilla qubits are measured in the appropriate basis. Right: Interleaving of separate stabiliser measurements, including late preparation and early measurement for weight-two stabilisers.

using the *Blossom algorithm* [26, 27]. This algorithm produces accurate corrections for the surface code, but has a complexity which scales super-linearly with respect to d. This is an obstacle to using the Blossom algorithm for decoding surface codes in practice, for reasons which we explain in the following section.

III. NEED FOR FAST DECODING

Projective measurement of the logical qubits and classical feedforward of the measurement values are key ingredients in universal fault-tolerant quantum computing. To calculate the bit which we feed forward, we need to decode. Thus, it is necessary to correct errors frequently during a computation.

While the decoding takes place at the classical co-processor, we could either continue running

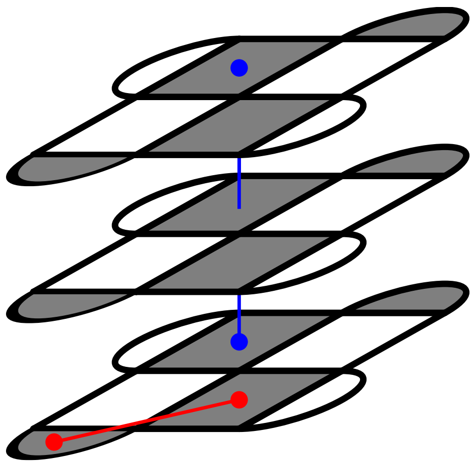


FIG. 3. Three consecutive rounds of surface code measurement arranged in a $2+1$-dimensional lattice. Errors on data qubits result in horizontally-separated changes in the syndrome record, measurement errors result in vertically-separated changes.

rounds of syndrome measurement or stop and wait for the decoding to be concluded. If we stop the computation, errors will build up until they become uncorrectable. This takes an amount of time which depends on the implementation in question ($\sim 10\,\mu$s in current superconducting circuits, for example [28]). On the other hand, if we continue measuring syndromes, we will build a *backlog* of data that produces a more difficult decoding problem in the future. The ideal case would be a decoder that decodes d rounds of syndrome measurement in less time than the time needed to perform the measurements themselves. In superconducting circuits, the time for a single round of syndrome measurement is 800 ns [29].

There are many techniques that provide high performance decoding. In the following section, we summarize some of them.

IV. RELATED WORK

To decrease decoding time when correcting time-dependent errors, the "overlapping recovery" method was introduced in [22]. This method divides the measurement record into *windows*, defined as a set of $\sim d$ consecutive error correction cycles. In the overlapping recovery technique, syndrome changes are matched either to each other (pairwise) or to a time boundary placed immediately after the last round of syndrome measurement. At the next window, the syndrome changes matched to the time boundary are forwarded to the following window, in order to identify chains of errors which cross the boundary. This reduces the backlog problem mentioned earlier, by allowing the decoding problem to be solved incrementally.

To further reduce the backlog, Fowler [30] has parallelized the Blossom algorithm, using message-passing between local processors to replace slow subroutines. This technique produces accurate corrections, resulting in a high threshold error rate, and is scalable to large code distances. However, in the near future, only small code distances will be experimentally viable, so it is likely that a heuristic approach will perform well.

One such approach is taken in [24]. In this paper, the authors have designed a heuristic-based decoder that resembles the parallelized MWPM decoding for a distance-3 Surface Code with a window of 3 error correction cycles. The simple structure of this heuristic algorithm makes it easily programmable to hardware, decreasing the decoding time. The main drawback of this algorithm is that it cannot easily be extended to higher code distances, so an alternate method is required.

Currently, machine learning techniques are being explored as possible alternate decoding techniques for both classical LDPC codes [31, 32] and quantum codes, independently of the need for high-speed decoding. One such technique is being used in [33]. The authors of this paper use a stochastic neural network (or Boltzmann machine) to decode stabilizer codes. They optimize the neural network to fit a dataset that includes the errors and their respective syndromes. The network then models the probability distribution of the errors in the dataset and generates prospective recovery error chains when a syndrome is input. Many networks are produced for a variety of physical error probabilities p, so when an error syndrome is obtained, a random recovery chain of errors is sampled from the distribution corresponding to the known value of p. While this method was as accurate as MWPM decoding for simple error models, repeated sampling is required in order to produce an error that conforms with the syndrome, which takes unknown time.

To achieve high accuracy in bounded time, we use a simpler machine learning technique, the feed-forward neural network, which we introduce and apply to the decoding problem in the next section.

V. NEURAL NETWORK DECODER

To apply machine learning techniques to surface code decoding, we first reduce the decoding problem to a well-studied problem in machine learning; classification. Classification problems consist of a set of (generally high-dimensional) inputs, each of which is associated with a (generally low-dimensional) label. The goal is to optimize the assignment of known labels to known inputs (a process called *training*) so that unknown inputs can also be correctly labeled.

To reduce the decoding problem to a classification problem, we decompose an error E into three multi-qubit Pauli operators:

$$E = S \cdot C \cdot L, \tag{2}$$

where S is a stabiliser, C is any fixed Pauli which produces the syndrome $\vec{s}$ (also known as a *pure error* [34]), and L is a logical Pauli operator of the surface code, see Figure 4. Any decoder which

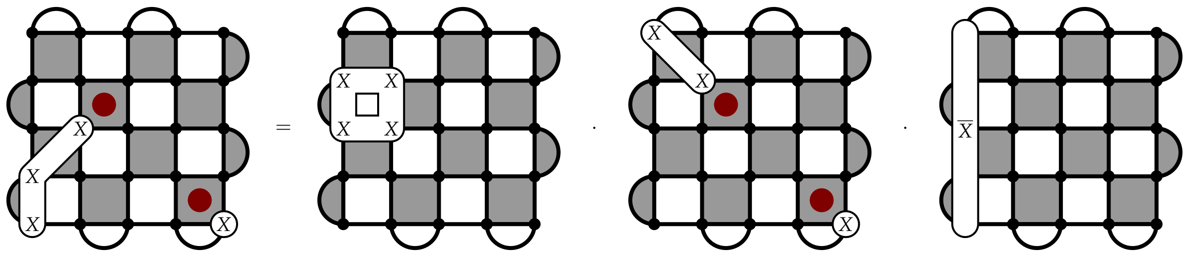


FIG. 4. A surface code error E decomposed into three components; a stabiliser S, a fixed Pauli C which produces the same syndrome as E, and a logical operator L.

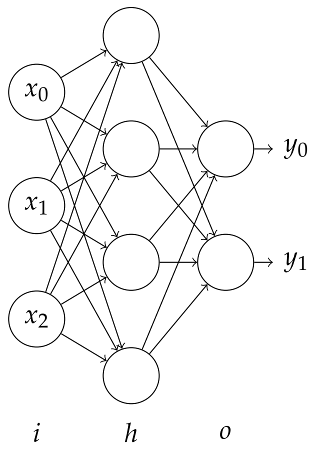
provides a correction $E' = S' \cdot C \cdot L$, in which the stabiliser in the correction is different from that in the actual error, does not lead to a logical error. This implies that S can be assigned arbitrarily with no impact on decoder accuracy. Also, it is possible to produce a pure error by parallel table look-up, since each bit of the syndrome can be assigned a unique pure error, independently of the other bits. We call the apparatus that produces this error the *simple decoder*. Since pure errors can be determined quickly in this fashion, the neural network only has to identify L, which can take one of four values; $\hat{\mathbb{1}}$, $\bar{X}$, $\bar{Y}$, or $\bar{Z}$. These four values can be used as labels in a classification problem.

To solve this problem, we use feed-forward neural networks, which are widely regarded as the simplest machine learning technique [35]. A feed-forward neural net can be described graphically or functionally, see Figure 5. The artificial neurons which comprise the network are devices which store a length-m internal array of *weights* $\vec{w}$, as well as a *bias* b. Upon receiving a length-m input $\vec{x}$, they calculate $(1+\exp(-(\vec{w}\cdot\vec{x}+b)))^{-1}$ and either broadcast the result to a subsequent layer of neurons, or output the result directly if the neuron in question is part of the *output layer*. Such output is typically denoted $\vec{y}$, and its accuracy is measured using an approptiate cost function.

A typical cost function, which we use in this work, is the average cross-entropy:

$$\langle H(p,y)\rangle \propto -\sum_{(\vec{p},\vec{x})\in T} \vec{p}\cdot\ln(\vec{y}(\vec{x})), \tag{3}$$

where T is the training set, consisting of desired ('target') distributions $\vec{p}$ and input values $\vec{x}$. The cross-entropy is a measure of the divergence between two probability (or frequency) distributions, differing from the Kullback-Leibler divergence [36] only in terms that depend solely on $\vec{p}$. To minimize the cross-entropy, we use stochastic gradient descent, as implemented in the Tensorflow library [37]. To produce a training set, we use direct sampling at a single physical error probability, where the Blossom algorithm produces a logical error rate of $\sim 25\%$. This physical error probability is chosen so that a large variety of error syndromes can be produced while still ensuring that correction is possible. For small surface codes, it is possible to sample the entire set of possible syndromes, we limit the size of the training set to at most 10^6 samples for larger codes. This training set size provides relatively fast training and high accuracy, as seen in Section VI.



$$\vec{y} = \sigma\left(\hat{W}_o\sigma\left(\hat{W}_h\vec{x}+\vec{b}_h\right)+\vec{b}_o\right)$$

FIG. 5. The graphical and functional descriptions of a feed-forward neural network. In the graphical description (left), inputs x_j are passed to neurons in a *hidden layer*, and each of these neurons outputs $\sigma(\vec{w}\cdot\vec{x}+b)$, where $\vec{w}$ and b are a local set of weights and a bias, and $\sigma(x)$ is a non-linear *activation function* (we use $\sigma(x) = (1+\exp(-x))^{-1}$ for all neurons considered in this work). The final outputs y_k can be rounded to $\{0, 1\}$, and interpreted as a class label. In the functional picture, the weights and biases are assembled into matrices and vectors, respectively, allowing the output vector to be expressed as a composition of functions acting on the input vector.

In the following section, we compare the performance of our decoder to the performance of Blossom and the performance of the partial lookup table (PLUT), which contains the error syndromes and corrections from the training set. In the event that the input syndrome is not in the training set, the PLUT decoder multiplies the correction from the simple decoder by the most frequent logical operator, $\hat{\mathbb{1}}$. The comparison in terms of performance is based on the logical error rate of each decoder for specific code distances and error models.

VI. RESULTS

In the proposed decoder, we provide the error syndrome to both the simple decoder and the neural network. As presented in Table 1, the size of the input for the neural network is equal to the number of required syndrome bits, depending on the error model, and only one hidden layer was used for all networks. The number of nodes in the hidden layer was decided based on the performance of the neural network during training and testing.

QEC Error Models			
Code Capacity Noise			
Code distance	Input nodes	Hidden nodes	Output nodes
3	4	10	2
5	12	90	2
7	24	512	2
Depolarizing Noise			
3	8	128	4
5	24	660	4
7	48	256	4

FT Error Models			
Phenomenological Noise			
Code distance	Input nodes	Hidden nodes	Output nodes
3	16	768	4
Depolarizing & Measurement Noise			
3	32	768	4
Circuit Noise			
3	32	704	4

TABLE I. Layer sizes for the neural networks used throughout this work. The number of input nodes is determined by the number of syndromes in the quantum error correction scenario, using only X (or Z) syndrome bits for independent X/Z errors, and all syndrome bits for depolarizing errors. For fault tolerance error models, d rounds of measurement are followed by readout of the data qubits, and calculated stabiliser eigenvalues are included in the input. The output layer is restricted to two nodes for independent X/Z errors, since logical X/Z errors are also independent. In all other scenarios, four nodes are used to discriminate between $\hat{\mathbb{1}}$, $\bar{X}$, $\bar{Y}$, and $\bar{Z}$. The number of nodes in the hidden layer is determined by analysing the performance of the resulting decoder empirically.

We test the proposed decoder against Blossom and the PLUT decoder for two classes of error models, called quantum error correction (QEC) and fault tolerance (FT). Quantum error correction (QEC) error models approximate noise only on data qubits and fault tolerance (FT) error models approximate noise on all qubits, gates and operations, therefore requiring multiple rounds of measurement to find all errors. The code capacity model inserts only X or only Z errors with probability p in the data qubits. The depolarizing model places $X/Y/Z$ errors with equal probability, $p/3$, on the data qubits. For these error models only one cycle of error correction is required to find all errors.

For fault tolerance error models, the probability of an error occurring on a qubit and the probability of a measurement error is the same, therefore the minimum number of rounds of measurement is taken to be d. Instead of data qubit and measurement errors, the circuit noise model assumes that all operations and gates are noisy. Each single-qubit gate is followed by depolarizing noise with probability $p/3$ and each two-qubit gate is followed by a two-bit depolarizing map where each non-$\hat{\mathbb{1}} \otimes \hat{\mathbb{1}}$ two-bit Pauli has probability $p/15$. Preparation and measurement locations fail with probability p, resulting in a prepared -1-eigenstate or measurement error, respectively.

In our simulations, more than 10^6 error correction cycles were run per point, and each point has a confidence interval of 99.9%. The percentage of the most frequent error syndromes that were used as training cases for the QEC error models were 100% ($d = 3$), 72.46% ($d = 5$), 2.75% ($d = 7$), see figure 6, and 100% ($d = 3$), 0.98% ($d = 5$), 3×10^{-7}% ($d = 7$), see figure 7, for code capacity and depolarizing models respectively. The percentage of the most frequent error syndromes that were used as training cases for the fault tolerance error models were 30.09%, 0.022% and 0.01% for the code capacity (see figure 8 top), the depolarizing (see figure 8 middle), and the circuit model (see figure 8 bottom), respectively. The performance of our decoder was compared to the Blossom algorithm and the PLUT decoder.

In the QEC error models, see figure 6 and figure 7, we observe a clear trend. In both error models, as the distance increases the performance of our decoder remains similar to Blossom, and becomes much better than the PLUT-based decoder. This demonstrates that the neural networks of our decoder can successfully correct error syndromes that were not included in training. At small code distances, almost all possible error syndromes were used in training, resulting in identical performance from both the PLUT and our decoder. However, going to larger distances while using a small set of error syndromes for training, leads to sub-optimal decoding by the PLUT decoder.

It is known that, for the code capacity error model, Blossom can reach near-optimal accuracy, therefore it is sufficient for our decoder to reach similar accuracy. There are correctable errors (with weight ≤ 3) in distance 7 that are not included in the training set and the neural network is not generalizing correctly. Therefore, the performance is $\sim 1\%$ worse than Blossom's. However, for the depolarizing error model, Blossom is known to misidentify Y errors, since it performs the decoding for X and Z errors separately, treating a Y error as two distinct errors. Thus, if we train our decoder to take Y errors into account as weight-1 errors, the performance will be better than Blossom's. In the depolarizing model, there are still a few weight 3 errors that are being mis-identified, however the existence of higher weight errors in the training set, that are being corrected properly, account for the sligthly better performance compared to the Blossom decoder.

In the fault tolerance scenario, see figure 8, due to the small code distance, all decoders reach a similar level of performance. Specifically, for the code capacity and the depolarizing error model, only a small amount of error syndromes was necessary to reach Blossom's performance. The circuit metric required more syndromes, however slightly better performance was achieved in this case as well.

It is encouraging that the neural network based decoder can achieve similar performance to Blossom. However, the main reason that such a design is proposed is to accelerate the decoding time. In the following section, we provide an estimation of the speed of the neural network based decoder in hardware, and discuss the implications for future research.

VII. DISCUSSION AND CONCLUSION

To estimate runtime scaling of a neural network decoder of the type introduced above, we first note that the runtime is dictated by the sizes of the input, hidden and output layers. The input

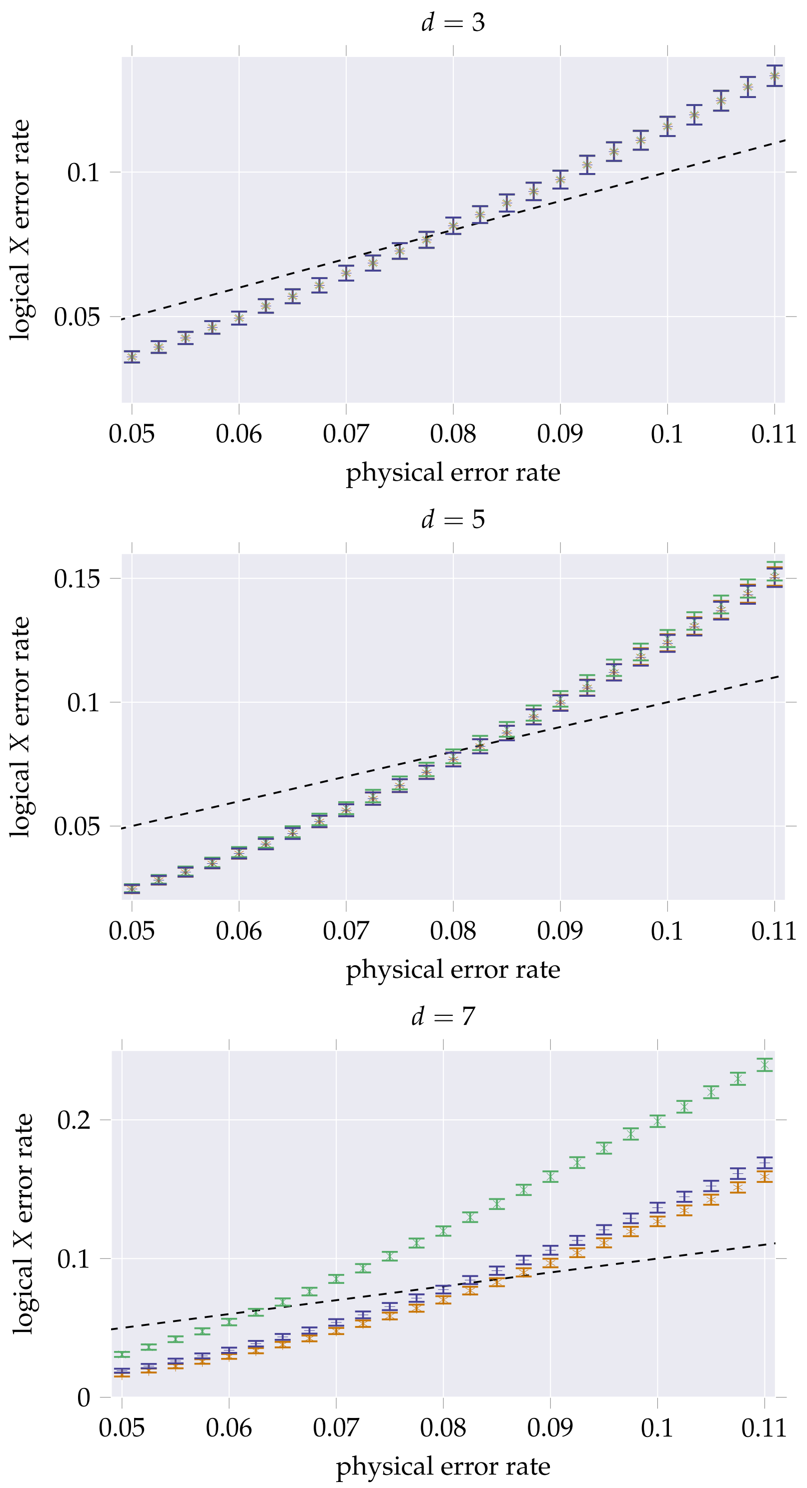


FIG. 6. Code capacity error model without measurement errors for Surface Code distances 3, 5 and 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green), and un-encoded qubit (black dashed line).

layer grows quadratically with the distance of the surface code in question, and the size of the output layer is constant. The size of the hidden layer, however, is selected through trial and error,

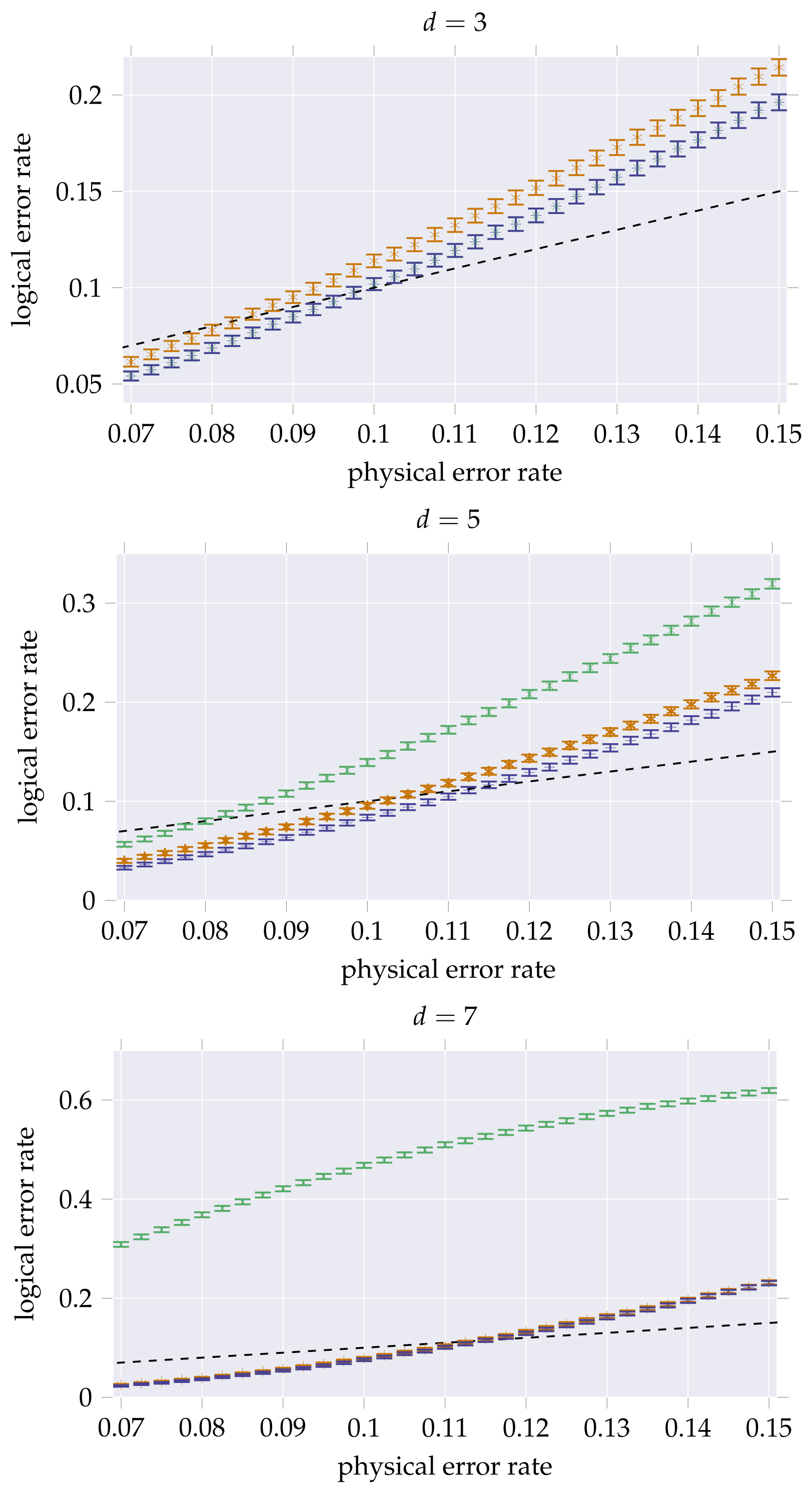
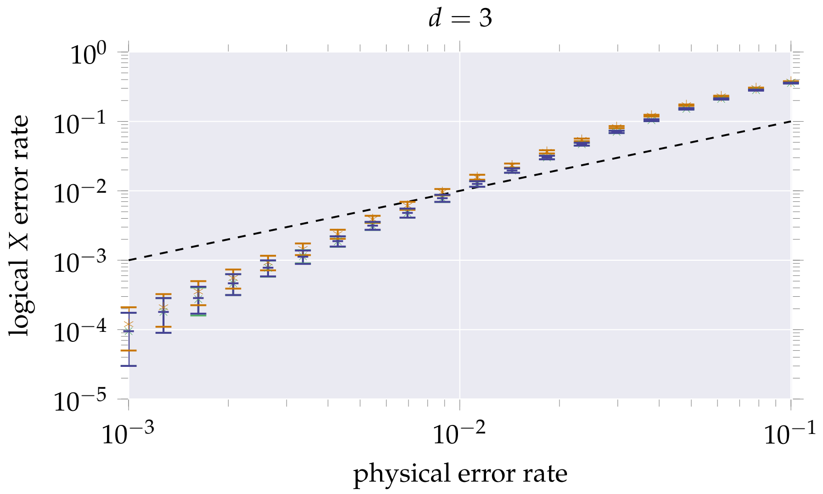
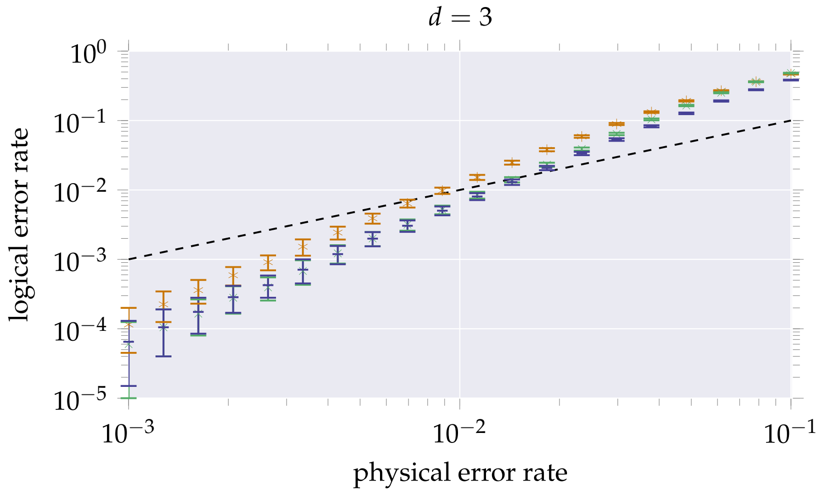


FIG. 7. Depolarizing error model without measurement errors for Surface Code distances 3, 5 and 7. Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green), and un-encoded qubit (black dashed line).

so it cannot be predicted a priori.

Given a hidden layer size, we can estimate the required runtime, using the graphical descrip-





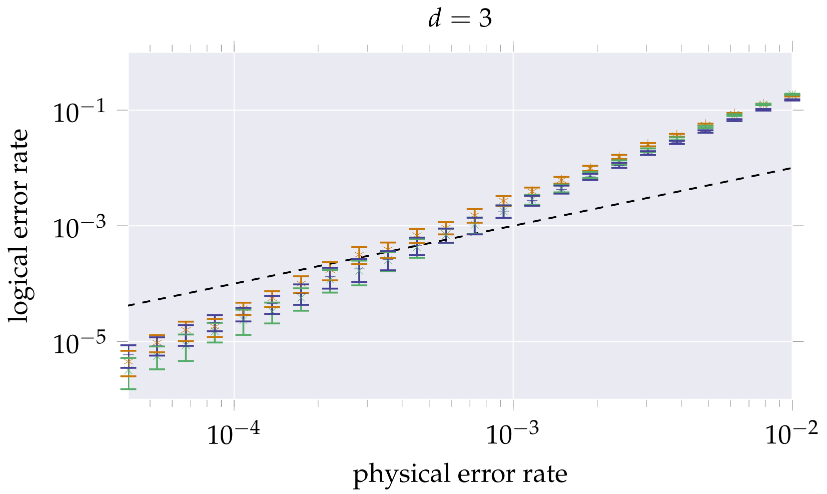


FIG. 8. code capacity error model with measurement errors for Surface Code distance 3 (top). Depolarizing error model with measurement errors for Surface Code distance 3 (middle). Circuit error model for Surface Code distance 3 (bottom). Performance comparison of the neural network decoder (blue) to the MWPM algorithm (orange) partial look-up table (green), and un-encoded qubit (black dashed line).

tion of neural networks (see Figure 5). Firstly, the output of each neuron can be evaluated independently, so the runtime is dominated by the time needed to take an inner product between the weight vector and an intermediate state in the network. Each multiplication can be performed independently, and summation requires a logarithmic-depth circuit composed of adders, arranged in a tree-like structure. This implies that any sub-exponential growth in the size of the hidden layer will result in sub-linear growth in the required decoding time.

To get a better idea of how these scaling arguments function in practice, we can bound the performance of a neural network decoder, and compare with similar bounds using the Blossom algorithm, running on a CPU. We consider an average instance of the decoding problem for the $d = 3$ rotated surface code correcting circuit noise, in which three syndrome differences occur during repeated measurement, leading to a graph with 6 vertices and 9 edges. We also neglect the runtime of the simple decoder, since, for the distance-3 code, it can be reduced to a simple look-up table.

To lower-bound the time required for the Blossom algorithm, we hard-code the relevant graph data into a C program and calculate the matching. This requires $\sim$ 500 ns, indicating that the Blossom algorithm can compute a correction on the timescale required for the distance-3 code. However, this neglects the time required to translate syndrome data from experimental hardware into the data structure required by the algorithm. To upper-bound this time, we write a second program which reads the graph data from a file, resulting in a runtime of $\sim 6\,\mu$s, slightly too slow to be compatible with an 800 ns syndrome measurement time.

To lower-bound the runtime of the neural network, we make the assumption that each arithmetic operation will require one clock cycle of the appropriate FPGA, typically 1–5 ns [38]. The network which corrects errors from the circuit model requires two multiplications (one for the hidden layer, and one for the output layer), 15 addition steps ($\lceil \log_2(32) \rceil + \lceil \log_2(704) \rceil$), and two evaluations of the sigmoid function, for a total of 19 serial steps, requiring 19–95 ns. To obtain an upper bound, we estimate the number of clock cycles required using a high-level synthesis tool [39], which permits hardware-synthesizable code to be generated from C, but does not ensure that the resulting code is optimized for speed. This results in an estimate of $\sim$ 3600 clock cycles, requiring $\sim$ 3.6–18 μs, again slightly too slow to be compatible with an 800 ns syndrome measurement time. Either decoder would require only minor optimization to attain compatibility at distance 3, an ideal topic for immediate future work.

In concert with this, we can extend the proposed decoder to the case where syndromes from a finite window are fed forward, as in [22]. In addition, we can begin testing the applicability of feedforward neural networks to surface codes with larger distance, as well as to alternate topological codes for which existing decoders do not attain high accuracy and speed simultaneously [40–42].

If feedforward neural networks can efficiently decode a large range of topological codes, it will also be interesting to determine the scope of their applicability. There are known families of sparse and/or high-rate stabiliser codes [43–45] for which fast and accurate decoders are not known. Aside from concatenated codes, which can be optimally decoded by fast message passing [34], neural networks provide an interesting avenue to quickly test the performance of small codes, with $\sim$ 50 stabiliser measurements.

In conclusion, feedforward neural networks provide a fast and accurate method to decode small surface codes, both for performing quantum error correction, as well as fault-tolerant operations. Given that the hardware requirements and anticipated runtime are relatively low, we expect feedfoward neural network decoders to be usable in the near term.

VIII. ACKNOWLEDGEMENTS

We thank Tom O'Brien, Paul Baireuther and Brian Tarasinski for useful discussions, and Imran Ashraf for his timely and invaluable assistance with high-level synthesis tools for FPGA programming. We acknowledge financial support from Intel in the context of the QuTech-Intel collaboration.

[1] IM Georgescu, S Ashhab, and Franco Nori. Quantum simulation. *Reviews of Modern Physics*, 86(1):153, 2014.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172. URL `http://dx.doi.org/10.1137/S0097539795293172`.

[3] Thomas Monz, Daniel Nigg, Esteban A. Martinez, Matthias F. Brandl, Philipp Schindler, Richard Rines, Shannon X. Wang, Isaac L. Chuang, and Rainer Blatt. Realization of a scalable Shor algorithm. *Science*, 351(6277):1068–1070, 2016. doi:10.1126/science.aad9480.

[4] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997. doi:10.1103/PhysRevLett.79.325. URL `https://link.aps.org/doi/10.1103/PhysRevLett.79.325`.

[5] Stephen Jordan. Quantum algorithms, October 2016. URL `http://math.nist.gov/quantum/zoo/`.

[6] Zhen Wu, Jun Li, Wenqiang Zheng, Jun Luo, Mang Feng, and Xinhua Peng. Experimental demonstration of the Deutsch-Jozsa algorithm in homonuclear multispin systems. *Phys. Rev. A*, 84:042312, Oct 2011. doi:10.1103/PhysRevA.84.042312. URL `https://link.aps.org/doi/10.1103/PhysRevA.84.042312`.

[7] Yang Liu and FeiHao Zhang. First experimental demonstration of an exact quantum search algorithm in nuclear magnetic resonance system. *Science China Physics, Mechanics & Astronomy*, 58(7): 1–6, 2015. ISSN 1869-1927. doi:10.1007/s11433-015-5661-z. URL `http://dx.doi.org/10.1007/s11433-015-5661-z`.

[8] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(4213), 2014. doi:10.1038/ncomms5213. URL `http://dx.doi.org/10.1038/ncomms5213`.

[9] L. DiCarlo, J. M. Chow, J. M. Gambetta, Lev S. Bishop, B. R. Johnson, D. I. Schuster, J. Majer, A. Blais, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature*, 460(7252):240–244, 2009. doi:10.1038/nature08121. URL `http://dx.doi.org/10.1038/nature08121`.

[10] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52 (4):R2493, 1995.

[11] C. Chamberland, P. Iyer, and D. Poulin. Fault-Tolerant Quantum Computing in the Pauli or Clifford Frame with Slow Error Diagnostics. *ArXiv e-prints*, April 2017.

[12] Khodjasteh Kaveh, Lidar Daniel A., and Viola Lorenza. Arbitrarily accurate dynamical control in open quantum systems. *Phys. Rev. Lett.*, 104:090501, Mar 2010. doi:10.1103/PhysRevLett.104.090501. URL `https://link.aps.org/doi/10.1103/PhysRevLett.104.090501`.

[13] F. Motzoi, J. M. Gambetta, P. Rebentrost, and F. K. Wilhelm. Simple pulses for elimination of leakage in weakly nonlinear qubits. *Phys. Rev. Lett.*, 103:110501, Sep 2009. doi:10.1103/PhysRevLett.103.110501. URL `https://link.aps.org/doi/10.1103/PhysRevLett.103.110501`.

[14] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. Caltech Ph.D. Thesis, 1997.

[15] Daniel A. Lidar and Todd A. Brun. *Quantum Error Correction*. Cambridge University Press, 2013.

[16] Peter W Shor. Fault-tolerant quantum computation. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 56–65. IEEE, 1996.

[17] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society*, A:385–410, 1998. doi: 10.1098/rspa.1998.0167. URL `https://link.aps.org/doi/10.1098/rspa.1998.0167`.

[18] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2 – 30, 2003. ISSN 0003-4916. doi:http://doi.org/10.1016/S0003-4916(02)00018-0. URL `http://www.sciencedirect.com/science/article/pii/S0003491602000180`.

[19] Michael H Freedman and David A Meyer. Projective plane and planar quantum codes. *Foundations of Computational Mathematics*, 1(3):325–332, 2001.

[20] Sergey B Bravyi and A Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052*, 1998.

[21] H Bombin and Miguel A Martin-Delgado. Optimal resources for topological two-dimensional stabilizer codes: Comparative study. *Physical Review A*, 76(1):012305, 2007.

[22] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002. doi:10.1063/1.1499754. URL `http://dx.doi.org/10.1063/1.1499754`.

[23] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80(5):052312, 2009.

[24] Yu Tomita and Krysta M. Svore. Low-distance surface codes under realistic quantum noise. *Phys. Rev. A*, 90:062320, Dec 2014. doi:10.1103/PhysRevA.90.062320. URL `https://link.aps.org/doi/10.1103/PhysRevA.90.062320`.

[25] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Number v. 1 in Algorithms and Combinatorics. Springer, 2003. ISBN 9783540443896.

[26] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4. URL `http://dx.doi.org/10.4153/CJM-1965-045-4`.

[27] Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009. doi:10.1007/s12532-009-0002-8. URL `http://dx.doi.org/10.1007/s12532-009-0002-8`.

[28] TE O'Brien, B Tarasinski, and L DiCarlo. Density-matrix simulation of small surface codes under current and projected experimental noise. *arXiv preprint arXiv:1703.04136*, 2017.

[29] R Versluis, S Poletto, N Khammassi, N Haider, DJ Michalak, A Bruno, K Bertels, and L DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *arXiv preprint arXiv:1612.08208*, 2016.

[30] Austin G. Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time. *Quantum Information & Computation*, 15:145–158, 2015.

[31] Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 341–346. IEEE, 2016.

[32] Eliya Nachmani, Elad Marciano, David Burshtein, and Yair Be'ery. RNN Decoding of Linear Block Codes. *arXiv preprint arXiv:1702.07560*, 2017.

[33] Giacomo Torlai and Roger G Melko. A neural decoder for topological codes. *arXiv preprint arXiv:1610.04238*, 2016.

[34] David Poulin. Optimal and efficient decoding of concatenated quantum block codes. *Phys. Rev. A*, 74:052333, Nov 2006. doi:10.1103/PhysRevA.74.052333. URL `https://link.aps.org/doi/10.1103/PhysRevA.74.052333`.

[35] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[36] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951. doi:10.1214/aoms/1177729694. URL `http://dx.doi.org/10.1214/aoms/1177729694`.

[37] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[38] *Virtex UltraScale FPGAs Datasheet : DC and AC switching characteristics*, March 2017.

[39] Yun Liang, Kyle Rupnow, Yinan Li, Dongbo Min, Minh N Do, and Deming Chen. High-level synthesis: productivity, performance, and software constraints. *Journal of Electrical and Computer Engineering*, 2012:1, 2012.

[40] Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical review letters*, 97(18):180501, 2006.

[41] Andrew J Landahl, Jonas T Anderson, and Patrick R Rice. Fault-tolerant quantum computing with color codes. *arXiv preprint arXiv:1108.5738*, 2011.

[42] Hector Bombin, Guillaume Duclos-Cianci, and David Poulin. Universal topological phase of two-dimensional stabilizer codes. *New Journal of Physics*, 14(7):073048, 2012.

[43] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2014.

[44] Alain Couvreur, Nicolas Delfosse, and Gilles Zémor. A construction of quantum LDPC codes from Cayley graphs. *IEEE Transactions on Information Theory*, 59(9):6087–6098, 2013.

[45] Dave Bacon, Steven T Flammia, Aram W Harrow, and Jonathan Shi. Sparse quantum codes from quantum circuits. *IEEE Transactions on Information Theory*, 63(4):2464–2479, 2017.