Designing neural network based decoders for surface codes

Preprint · November 2018

DOI: 10.48550/arXiv.1811.12456

3 authors, including:

Savvas Varsamopoulos
PASQAL
15 PUBLICATIONS **560** CITATIONS
SEE PROFILE

Koen Bertels
Korea Advanced Institute of Science and Technology
348 PUBLICATIONS **6,736** CITATIONS
SEE PROFILE

Designing neural network based decoders for surface codes

Savvas Varsamopoulos,[1,2] Koen Bertels,[1,2] and Carmen G. Almudever[1,2]

[1]*Quantum Computer Architecture Lab, Delft University of Technology, The Netherlands*
[2]*QuTech, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands*
Email: S.Varsamopoulos@tudelft.nl

Recent works have shown that small distance quantum error correction codes can be efficiently decoded by employing machine learning techniques like neural networks. Various techniques employing neural networks have been used to increase the decoding performance, however, there is no framework that analyses a step-by-step procedure in designing such a neural network based decoder. The main objective of this work is to present a detailed framework that investigates the way that various neural network parameters affect the decoding performance, the training time and the execution time of a neural network based decoder. We focus on neural network parameters such as the size of the training dataset, the structure, and the type of the neural network, the batch size and the learning rate used during training. We compare various decoding strategies and showcase how these influence the objectives for different error models. For the noiseless syndrome measurements, we reach up to 50% improvement against the benchmark algorithm (Blossom) and for the noisy syndrome measurements, we show efficient decoding performance up to 97 qubits for the rotated Surface code. Furthermore, we show that the scaling of the execution time (time calculated during the inference phase) is linear with the number of qubits, which is a significant improvement compared to existing classical decoding algorithms that scale polynomially with the number of qubits.

I. Introduction

Constant active quantum error correction (QEC) is regarded as necessary in order to perform reliable quantum computation and storage due to the unreliable nature of current quantum technology. Qubits inadvertently interact with their environment even when no operation is applied, forcing their state to change (decohere). Moreover, application of imperfect quantum gates results in the introduction of errors in the quantum system. Quantum error correction is the mechanism that reverses these errors and restores the state to the desired one.

Quantum error correction involves an encoding and a decoding process. Encoding is used to enhance protection against errors by employing more resources (qubits). The encoding scheme that is used in this work is the rotated ***surface code*** [1]. Decoding is the process that is used to identify the location and type of error that occurred. As part of quantum error correction, decoding has a limited time budget that is determined by the time of a single round of error correction. In the case that the decoding time exceeds the time of quantum error correction, either the quantum operations are stalled or a backlog of inputs to the decoding algorithm is created [2]. Many classical decoding algorithms have been proposed with the most widely used being the ***Blossom algorithm*** [3]. Blossom algorithm has been shown to reach high decoding accuracy, but its decoding time scales polynomially with the number of qubits [4], which can be problematic for large quantum systems needed to solve complex problems. However, there are optimized versions of Blossom for topological codes, that report linear scaling with the number of qubits [5] and even a parallel version stating that the average processing time per detection round is constant, independent of the size of the system [6]). We propose the employment of neural networks, since they exhibit constant execution time after being trained without any parallelization required and have been proven to provide better decoding performance than many classical decoding algorithms[7–12].

In this work, we investigate various designs of neural network based decoders and compare them in terms of the decoding accuracy, training time and execution time. We analyze how different neural network parameters like the size of the training dataset, the structure and type of the neural network, the batch size and the learning rate used during training, affect the accuracy, the training and execution time of such a decoder.

The rest of the paper is organized as follows: in sections II and III we provide a brief introduction to quantum error correction and neural networks, respectively. In section IV, we explain how the different designs of neural network decoders work, as found in literature. Section V, presents the guidelines of utilization of the neural network parameters. In section VI, we provide the results with the best neural network decoder for the different error models. Finally, in section VII, we draw our conclusions about this research.

II. Quantum error correction

Similar to classical error correction, quantum error correction encodes a set of unreliable ***physical*** qubits to a more reliable qubit, known as ***logical*** qubit.

Various quantum error correcting codes have been developed so far, but in this work we only consider the surface code [13–16], one of the most promising QEC codes. The ***surface code*** is a topological stabilizer code that has a simple structure, local interactions between qubits and is proven to have high tolerance against errors [17]. It is usually defined over a 2D lattice of qubits, although higher dimensions can be used.

In the surface code, a logical qubit consists of physical qubits that store quantum information, known as ***data*** qubits, and physical qubits that can be used to detect errors in the logical qubit through their measurement, known as ancillary or ***ancilla*** qubits (see Figure 1).

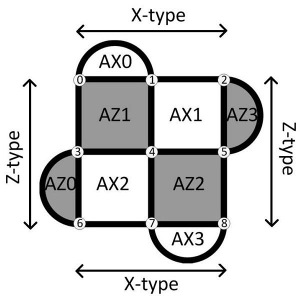
A logical qubit is defined by its ***logical operators*** ($\bar{X}$,$\bar{Z}$) that define how the logical state of the qubit can be changed. Any operator of the form $X^{\otimes n}$ or $Z^{\otimes n}$ that creates a chain that span both boundaries of the same type can be regarded as a logical operator, with n being the number of data qubits that are included in the logical operator. Typically, the logical operator with the smallest n is selected. For instance, in Figure 1, a logical $\bar{X}$ can be performed by applying these three bit-flip operations: $X_0 X_3 X_6$.

An important feature of the surface code is the code distance. ***Code distance***, referred to as ***d***, describes the degree of protection against errors. More accurately, is the minimum number of physical operations required to change the logical state [1] [18]. In surface code, the degree of errors (d.o.e.) that can be successfully corrected, is calculated according to the following equation:

$$\text{d.o.e.} = \left\lfloor \frac{d-1}{2} \right\rfloor \tag{1}$$

Therefore, for a d=3 code, only single X- and Z-type errors (degree = 1) can be guaranteed to be corrected successfully.

One of the smallest surface codes, which is currently being experimentally implemented, is the rotated surface code presented in Figure 1. It consists of 9 data qubits placed at the corners of the square tiles and 8 ancilla qubits placed inside the square and semi-circle tiles. Each ancilla qubit can interact with its neighboring 4 (square tile) or 2 (semi-circle tile) data qubits.



Parity checks (X-type)

$$\begin{array}{c} \\ \text{AX0} \\ \text{AX1} \\ \text{AX2} \\ \text{AX3} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \text{D0} & \text{D1} & \text{D2} & \text{D3} & \text{D4} & \text{D5} & \text{D6} & \text{D7} & \text{D8} \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

Parity checks (Z-type)

$$\begin{array}{c} \\ \text{AZ0} \\ \text{AZ1} \\ \text{AZ2} \\ \text{AZ3} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \text{D0} & \text{D1} & \text{D2} & \text{D3} & \text{D4} & \text{D5} & \text{D6} & \text{D7} & \text{D8} \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

FIG. 1. Rotated surface code with code distance 3. Data qubits are enumerated from 0 to 8 (D0-D8). X-type ancilla are in the center of the white tiles and Z-type ancilla are in the center of grey tiles.

As mentioned, ancilla qubits are used to detect errors in the data qubits. Although quantum errors are continuous, the measurement outcome of each ancilla is discretized and then forwarded to the decoding algorithm. Quantum errors are discretized into bit-flip (X) and phase-flip (Z) errors, that can be detected by Z-type ancilla and X-type ancilla, respectively.

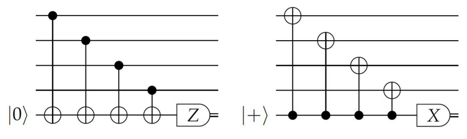


FIG. 2. Syndrome extraction circuit for individual Z-type (left) and X-type (right) ancilla, with the ancilla placed in the bottom. The ancilla qubit resides at the center of each grey or white tile, respectively.

The circuit that is used to collect the ancilla measurements for the surface code is known as ***syndrome extraction circuit***. It is presented in Figure 2 and it signifies one round of error correction. It includes the preparation of the ancilla in the appropriate state, followed by 4 (2) CNOT gates that entangle the ancilla qubit with its 4 (2) neighboring data qubits and then the measurement of the ancilla qubit in the appropriate basis. The measurement result of the ancilla is a parity-check, which is a value that is calculated as the parity between the state of the data qubits connected to it. Each ancilla performs a parity-check of the form of $X^{\otimes 4}/Z^{\otimes 4}$ (square tile) and $X^{\otimes 2}/Z^{\otimes 2}$ (semi-circle tile), as presented in Figure 2. When the state of the data qubits involved in a parity-check has not changed, then the parity-check will return the same value as in the previous error correction cycle. In the case where the state of an odd number of data qubits involved in a parity-check is changed compared to the previous error correction cycle, the parity-check will return a different value than the one of the previous cycle ($0 \leftrightarrow 1$). The change in a parity-check in consecutive error correction cycles is known as a ***detection event***.

Note that the parity-checks are used to identify errors in the data qubits without having to measure the data qubits explicitly and collapse their state. The state of the ancilla qubit at the end of every parity-check is collapsed through the ancilla measurement, but is initialized once more in the beginning of the next error correction cycle [19].

The parity-checks must conform to the following rules: i) must commute with each other, ii) must anti-commute with errors and iii) must commute with the logical operators. An easier way to describe these parity-checks is to view them as a matrix, as presented in Figure 1. A matrix containing the 4 X-type parity checks and a matrix containing the 4 Z-type parity-checks for the d=3 rotated surface code is presented. The notation Di refers to the i^{th} data qubit used in a given parity-check. A 1 in the matrix represents that a data qubit is involved in the parity check.

Gathering all measurement outcomes, forms the ***error syndrome***. Surface codes can be decoded by collecting the ancilla measurements out of one or multiple rounds of error correction and providing them to a decoding algorithm that identifies the errors and outputs data qubit corrections.

We provide a simple example of decoding for the d=3 rotated surface code presented in Figure 3 on the left side. The measurement of each parity-check operator returns a binary value indicating the absence or presence of a nearby data qubit error. Assume that a Z-type error has occurred on data qubit 4 and the initial parity-check has a value of 0. Ancilla A_{X_1} and A_{X_2} will return a value of 1, which indicates that a data qubit error has occurred in their proximity and ancilla A_{X_0} and A_{X_3} will return a value of 0 according to the parity-checks provided in Figure 1. However, due to the degenerate nature of the surface code, two different but complementary sets of errors can produce the same error syndrome. Regardless of which of the two potential sets of errors has occurred, the decoder is going to provide the same corrections every time the same error syndrome is observed. Therefore, there is an assumption when the decoder is designed of which corrections are going to be selected when each error syndrome is observed. For example, when the decoder observes ancilla A_{X_1} and A_{X_2} returning a value of 1, then there are two sets of errors that might have occurred: a Z error at data qubit 4 or a Z error at data qubit 2 and at data qubit 6. If the decoder is programmed to output a Z-type correction at data qubit 4, then in one case the error is going to be erased, but in the other case a logical error will be created. Based on that fact, there is no decoder that can always provide the right set of corrections, since there is a chance of misinterpretation of the error that have occurred.

A single error on a data qubit will be signified by a pair of neighboring parity-checks changing value from the previous error correction cycle. In the case where an error occurs at the sides of the lattice, only one parity-check will provide information about the error, because there is only one parity-check available due to the structure of the rotated surface code. Multiple data qubit errors that occur near each other, form one dimensional chains of errors which create only two detection events located at the endpoints of the chains (see Figure 3 on the left side and the red line on the right side). On the other hand, a measurement error, which is an error during the measurement itself, is described as a chain between the same parity-check over multiple error correction cycles (see the blue line in Figure 3 on the right side). This blue line represents an alternating pattern of the measurement values (0-1-0 or 1-0-1) coming from the same parity-check for consecutive error correction cycles. If such a pattern is identified and is not correlated with a data qubit error, then it is considered a measurement error, so no corrections should be applied.

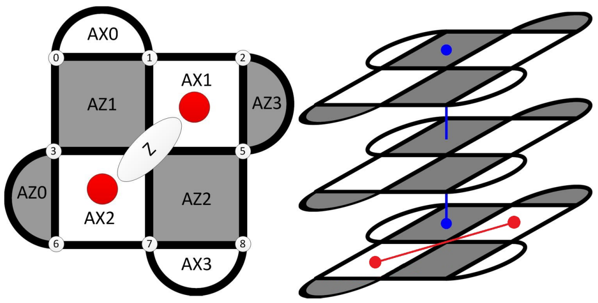


FIG. 3. Rotated surface code with code distance 3. Left: Phase-flip (Z) error at data qubit 4, which causes two neighboring parity checks to return a value of 1 (detection events shown in red). Right: Three consecutive rounds of error correction. The red dots indicate detection events that arise from a data qubit error and the blue dots indicate detection events that arise from a measurement error.

The parity-checks of the surface code protect the quantum state against quantum errors through continuous error correction and decoding. However, totally suppressing noise is not achievable, because as mentioned the decoder might misinterpret data qubit or measurement errors. Therefore, there have been developed algorithmic techniques that involve multiple error correction cycles to be run before corrections are proposed. In that way, measurement errors are easily identified by observing the error syndrome out of multiple error correction cycles and corrections for data qubit errors are more confidently proposed by observing the accumulation of errors throughout the error correction cycles.

There exist classical algorithms that can decode efficiently the surface code, however optimal decoding is a NP-hard problem. For example, maximum likelihood decoding (MLD) searches for the most probable error that produced the error syndrome, whereas minimum weight perfect matching (MWPM) searches for the least amount of errors that produced the error syndrome [5, 17]. MLD has a running time of $O(n\chi^3)$ according to [20] where χ is a parameter that controls the approximation precision and the optimized version of the Blossom algorithm shows linear scaling with the number of qubits. Also, a parallel version of the Blossom algorithm is described in [6] that claims constant execution time regardless of the size of the system.

Furthermore, current qubit technology offers a small time budget for decoding, making most decoders unusable for near-term experiments. For example, an error correction cycle of a d=3 rotated surface code with superconducting qubits, takes ~700nsec [2], which provides the upper limit of the allowed decoding time in this scenario. If noisy error syndrome measurements are also assumed, then d error correction cycles are required to provide the necessary information to the decoder, so in this scenario ~2.1μsec will be the upper limit for the time budget of decoding.

The way that Blossom algorithm performs the decoding is through a MWPM in a graph that includes the detection events that have occurred during the error correction cycles taken into account for decoding. If the number of qubits is increased, then the graph will be bigger and the decoding time will increase, assuming no parallelization.

An alternative decoding approach is to use neural networks to assist or perform the decoding procedure, since neural networks provide fast and constant execution time, while maintaining high application performance. In this paper, we are going to analyze decoders that include neural networks and compare them to each other and to an un-optimized version of the Blossom algorithm as described in [21].

III. Neural Networks

Artificial neural networks have been shown to reach high application performance and constant execution time after being trained on a set of data generated by the application.

An artificial neural network is a collection of weighted interconnected nodes that can transmit signals to each other. The receiving node processes the incoming signal and sends the processed result to its connected node(s). The processing at each node is different based on the different neural network parameters being used.

In this work, we focus on two types of neural networks known as ***Feed-forward neural networks (FFNN)*** and ***Recurrent neural networks (RNN)***. Feed-forward neural networks are considered to be the simplest type of neural network, allowing information to move strictly from input to output, whereas recurrent neural networks are considered to be more sophisticated, including feedback loops. The simple construction of FFNNs makes them extremely fast in appli-

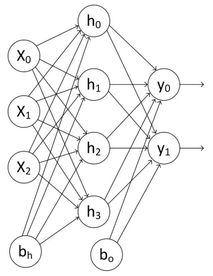


FIG. 4. The structure of an artificial neural network with an input layer (x), a hidden layer (h) and an output layer (y).

cations, however, RNNs are able to produce better results for more complex problems.

In Feed-forward neural networks, input signals x_i are passed to the nodes of the hidden layer h_i and the output of each node in the hidden layer acts as an input to the nodes of the following hidden layer, until the output of the nodes of the last hidden layer is passed to the nodes of the output layer y_i. The weighted connections between nodes of different layers are denoted as W_i and b is the bias of each layer. σ denotes the activation function that is selected, with popular options being the sigmoid, the hyperbolic tangent (tanh) and the rectified linear unit (ReLU). The output of the FFNN presented in Figure 4 is calculated as follows:

$$\vec{y} = \sigma(\hat{W}_0 \sigma(\hat{W}_h \vec{x} + \vec{b_h}) + \vec{b_0}) \tag{2}$$

In recurrent neural networks there is feedback that takes into account output from previous time steps y_{t-1}, h_{t-1}. RNNs have a feedback loop at every node, which allows information to move in both directions. Due to the feedback nature (see Figure 5), recurrent neural networks can identify temporal patterns of widely separated events in noisy input streams.

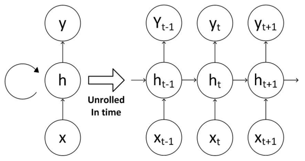
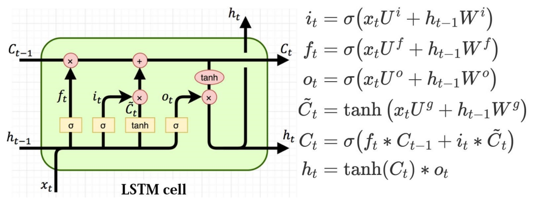


FIG. 5. A conceptual visualization of the recurrent nature of an RNN.

In this work, Long Short-Term Memory (LSTM) cells are used as the nodes of recurrent neural networks (see Figure 6). In an LSTM cell there are extra gates, namely the input, forget and output gate that are used in order to decide which signals are going to be forwarded to another node. W is the recurrent connection between the previous hidden layer and current hidden layer. U is the weight matrix that connects the inputs to the hidden layer. $\widetilde{C}$ is a candidate hidden state that is computed based on the current input and the previous hidden state. C is the internal memory of the unit, which is a combination of the previous memory, multiplied by the forget gate, and the newly computed hidden state, multiplied by the input gate [22]. The equations that describe the behavior of all gates in the LSTM cell are described in Figure 6.



h_t
C_{t-1}
C_t
f_t
i_t
$\tilde{C}_t$
o_t
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
LSTM cell
$i_t = \sigma(x_t U^i + h_{t-1} W^i)$
$f_t = \sigma(x_t U^f + h_{t-1} W^f)$
$o_t = \sigma(x_t U^o + h_{t-1} W^o)$
$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$
$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$
$h_t = \tanh(C_t) * o_t$

FIG. 6. Structure of the LSTM cell and equations that describe the gates of an LSTM cell.

The way that neural networks solve problems is not by explicit programming, rather "learning" the solution based on given examples. There exist many ways to "teach" the neural network how to provide the right answer, however in this work we are focusing on ***supervised learning***. Learning is a procedure which involves the creation of a map between an input and a corresponding output and in supervised learning the (input, output) pair is provided to the neural network. During training, the neural network adjusts its weights in order to provide the correct output based on the given input. Theoretically, at the end of training, the neural network should be able to infer the right output even for inputs that were not provided during training, which is known as ***generalization***.

Training is stopped when the neural network can sufficiently predict the right output to each training input. However, a definition of the closeness between the desired value and the predicted value needs to be defined. This metric is known as ***cost/loss function*** and guides the neural network towards the desired outcome by estimating the closeness between the predicted and the desired value. The cost function is calculated at the end of every training iteration after the weights have been updated. The cost function that we used is known as ***mean squared error***, which tries to minimize the average squared error between the desired output and the predicted output, given by

$$cost = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{3}$$

, where n is the number of data, Y_i is the target value and $\hat{Y}_i$ is the predicted value.

The procedure in which the weights are updated during training in order to minimize the cost function is known as ***backpropagation***. Backpropagation is a method that calculates the gradient of the cost function with respect to the weights, through the process of stochastic gradient descent. In order to be able to use neural networks to find solutions to a variety of applications (linear and non-linear), it is required to have a non-linear ***activation function*** at the processing step of every node. This function defines the contribution of this node to the subsequent nodes that it is connected to. The activation function that was used in this work was the Rectified Linear Unit.

IV. Designing neural network based decoders

Neural network based decoders for quantum error correcting codes have been recently proposed [7–12]. There are two categories in which they can be divided: i) decoders that search for exact corrections at the physical level and ii) decoders that search for corrections that restore the logical state. We are going to refer to the former ones as ***low level decoders*** [9, 10] and the latter ones as ***high level decoders*** [7, 8, 11, 12].

Since there are multiple techniques in designing a neural network based decoder, there is merit in figuring out which is the best design strategy. To achieve that, we implemented both designs as found in the literature and investigated their differences and similarities. Furthermore, we evaluate the decoding performance achieved with both decoders and investigate their training and execution time. We provide an analysis for various design parameters.

A. Inputs/Outputs

Low level decoders take as input the error syndrome and produce as output an error probability distribution for each data qubit based on the observed syndrome. Therefore, a prediction is made that attempts to correct exactly all physical errors that have occurred.

High level decoders take as input the error syndrome and produce as output an error probability for the logical state of the logical qubit. Based on this scheme, the neural network does not have to predict corrections for all data qubits, rather just for the state of the logical qubit, which makes the prediction simpler. This is due to the fact that there are only 4 options as potential logical errors, $\bar{I}, \bar{X}, \bar{Z}, \bar{Y}$, compared to the case of the low level decoder where the output is equivalent to the number of data qubits. Moreover, trying to correctly predict each physical error requires a level of high granularity which is not necessary for error correcting codes like the surface code.

B. Sampling and training process

During the sampling process, multiple error correction cycles are run and the corresponding inputs and outputs for each decoder are stored. Due to the degenerate nature of the surface code, the same error syndrome might be produced by different sets of errors. Therefore, we need to keep track of the frequency of occurrence of each set of errors that provide the same error syndrome.

For the low level decoder, based on these frequencies, we create an error probability distribution for each data qubit based on the observed error syndrome. For the high level decoder, based on these frequencies, we create an error probability distribution for each logical state based on the observed error syndrome.

When sampling is terminated, we train the neural network to map all stored inputs to their corresponding outputs. Training is terminated when the neural network is able to correctly predict at least 99% of the training inputs. Further information about the training process are provided in section V.

C. Evaluating performance

Designs for low level decoders typically include a single neural network. To obtain the predicted corrections for the low level decoder, we sample from the probability distribution that corresponds to the observed error syndrome for each data qubit, and predict whether a correction should be applied at each data qubit. However, this prediction needs to be verified before being used as a correction, because the proposed corrections must generate the same error syndrome as the one that was observed. Otherwise, the corrections are not valid (see Figure 7a-b). Only when the two error syndromes match, the predictions are used as corrections on the data qubits (see Figure 7a-c). If the observed syndrome does not match the syndrome obtained from the predicted corrections, then the predictions must be re-evaluated by re-sampling from the probability distribution. This re-evaluation step makes the decoding time non-constant, which can be a big disadvantage. There are ways to minimize the average amount of re-evaluations, however this is highly influenced by the physical error rate, the code distance and the strategy of re-sampling.

In Figure 7, the decoding procedure of the low level decoder is described with an example. On 7a, we present an observed error syndrome shown in red dots and the bit-flip errors on physical data qubits (shown with X on top of them) that created that syndrome. On 7b, the decoder predicts a set of corrections on physical data qubits and the error syndrome resulting from these corrections is compared against the observed error syndrome. As can be seen from 7a and 7b, the two error syndromes do not match therefore the predicted corrections are deemed **invalid**. On 7c, the decoder predicts a different set of corrections and the corresponding error syndrome to these corrections is compared against the observed error syndrome. In the case of 7a and 7c, the predicted error syndrome matches the observed one, therefore the corrections are deemed **valid**.

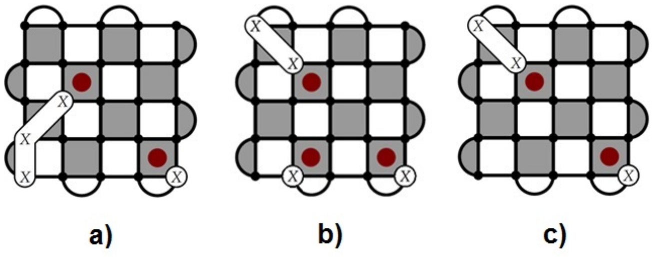


FIG. 7. Description of the decoding process of the low level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) **Invalid** data qubits corrections and the corresponding error syndrome. (c) **Valid** data qubits corrections and the corresponding error syndrome.

Designs for high level decoders typically involve two decoding modules that work together to achieve high speed and high level of decoding performance. Either both decoding modules can be neural networks [8] or one can be a classical module and the other can be a neural network[7, 11]. The classical module of the latter design will only receive the error syndrome out of the last error correction cycle and predict a set of corrections. In our previous experimentation [7], this classical module was called ***simple decoder***. The corrections proposed by the simple decoder do not need to exactly match the errors that occurred, as long as the corrections correspond to the observed error syndrome (valid corrections). The other module which in both cases is a neural network, should be trained to receive the error syndromes out of all error correction cycles and predict whether the corrections that are going to be proposed by the simple decoder are going to lead to a logical error or not. In that case, the neural network outputs extra corrections, which are the appropriate logical operator that erases the logical error. The output of both modules is combined and any logical error created by the corrections of the simple decoder will be canceled due to the added corrections of the neural network (see Figure 8).

Furthermore, the simple decoder is purposely designed in the simplest way in order to remain fast, regardless of the quality of proposed corrections. By adding the simple decoder alongside the neural network, the corrections can be given at one step and the execution time of the decoder remains small, since both modules are fast and operate in parallel.

In Figure 8, the decoding procedure of the high level decoder is described with an example. On 8a, we present an observed error syndrome shown in red dots and the bit-flip errors on physical data qubits (shown with X on top of them) that created that syndrome. On 8b, we present the decoding of the classical module known as simple decoder. The simple decoder receives the last error syndrome of the decoding procedure and proposes corrections on physical qubits by creating chains between each detection event and the nearest boundary of the same type as the parity-check type of the detection event. In Figure 8b, the corrections on the physical qubits are shown with X on top of them, indicating the way that the simple decoder functions. The simple decoder corrections are always deemed **valid**, due to the fact that the predicted and observed error syndrome always match based on the construction of the simple decoder. In the case of Figure 8a-b, the proposed corrections of the simple decoder are going to lead to an $\bar{X}$ logical error, therefore we use the neural network to identify this case and propose the application of the $\bar{X}$ logical operator as additional corrections to the simple decoder corrections, as presented in 8c.

V. Implementation parameters

In this section, we implement and compare both types of neural network based decoders as discussed in the previous section and argue about the better strategy to create such a decoder. The chosen strategy will be determined by investigation of how different implementation parameters affect the i) decoding performance, ii) training time and iii) execution time.

- The decoding performance indicates the accuracy of the algorithm during the decoding process. The typical way that decoding performance is evaluated is through lifetime simulations. In lifetime simulations, multiple error correction cycles are run and decoding is applied in frequent windows. Depending on the error model, a single error correction cycle might be enough to successfully decode, as in the case of perfect error syndrome measurements (window $=1$ cycle), or multiple error correction cycles might be required, as in the case of imperfect error syndrome measurements (window >1 cycle). When the lifetime simulations are stopped, the decoding performance is evaluated as the ratio of the number of logical errors found over the number of windows run until the simulations are stopped.

- The training time is the time required by the neural network to adjust its weights in a way that the training inputs provide the corresponding outputs as provided by the training dataset and adequate generalization can be achieved.

- The execution time is the time that the decoder needs to perform the decoding after being trained. It is calculated as the difference between the time when the decoder receives the first error syndrome of the decoding window and the time when it provides the output.

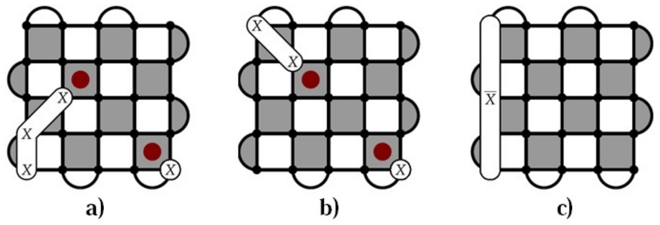


FIG. 8. Description of the decoding process of the high level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) Corrections proposed by the simple decoder for the observed error syndrome. (c) Additional corrections in the form of the $\bar{X}$ logical operator to cancel the logical error generated from the proposed corrections of the simple decoder.

A. Error model

These decoders were tested for two error models, the depolarizing error model and the circuit noise model.

The depolarizing model assigns X,Z,Y errors with equal probability $p/3$, known as depolarizing noise, only on the data qubits. No errors are inserted on the ancilla qubits and perfect parity-check measurements are used. Therefore, only a single cycle of error correction is required to find all errors.

The circuit noise model assigns depolarizing noise on the data qubits and the ancilla qubits. Furthermore, each single-qubit gate is assumed perfect but is followed by depolarizing noise with probability $p/3$ and each two-qubit gate is assumed perfect but is followed by a two-bit depolarizing map where each two-bit Pauli has probability $p/15$, except the error-free case, which has a probability of $1-p$. Depolarizing noise is also used at the preparation of a state and the measurement operation with probability p, resulting in the wrong prepared state or a measurement error, respectively. An important assumption is that the error probability of a data qubit error is equal to the probability of a measurement error, therefore d cycles of error correction are deemed enough to decode properly.

B. Choosing the best dataset

The best dataset for a neural network based decoder is the dataset that produces the best decoding performance. Naively, one could suggest that including all possible error syndromes, would lead to the best decoding performance, however, as the size of the quantum system increases, including all error syndrome becomes impossible. Therefore, we need to include as little but as diverse as possible error syndromes, which will provide the maximum amount of generalization, thus the best decoding performance, after training.

In our previous experimentation[7], we showed that sampling at a single physical error rate that always produces the fewest amount of corrections, is enough to decode small distance rotated surface codes with a decent level of decoding performance. This concept of always choosing the fewer amount of corrections is similar to the Minimum Weight Perfect Matching that Blossom algorithm uses.

After sampling and training the neural network at a single physical probability, the decoder is tested against a large variety of physical error rates and its decoding performance is observed. We call this approach, the ***single probability dataset*** approach, because we create only one dataset based on a single physical error rate and test it against many. Using the single probability dataset approach to decode various physical error probabilities is not optimal, because when sampling at low physical error rates, less diverse samples are collected, therefore the dataset is not diverse enough to correctly generalize to unknown training inputs.

The single probability approach is valid for a real experiment, since in an experiment there is a single physical error probability that the quantum system operates and at that probability the sampling, training and testing of the decoder will occur. However, this is not a good strategy for testing the decoding performance over a wide range of error probabilities. This is attributed to the degenerate nature of the surface code, since different sets of errors generate the same error syndrome. One set of errors is more probable when the physical error rate is small and another when it is high. Based on the design principles of the decoder, only one of these sets of errors, and always the same, are going to be selected when a given syndrome is observed regardless of the physical error rate. Therefore, training a neural network based decoder in one physical error rate and testing its decoding performance in a widely different physical error rate is not beneficial. The main benefit of this approach lies in the fact that only a single neural network has to be trained and used to evaluate the decoding performance for all the physical error rates that were tested. In the single probability dataset approach, the set with the fewer errors was always selected, because this set is more probable for the range of physical error rates that we are interested in.

To avoid such violations, we created different datasets that were obtained by sampling at various physical error rates and trained a different neural network at each physical error rate taken into account. We call this approach, the ***multiple probabilities datasets*** approach. Each dedicated training dataset that was created by a specific physical error probability is used to test the decoding performance at that same physical error probability and the probabilities close to that, but not all physical probabilities tested. Moreover, by sampling, training and testing the performance for the same physical error rate, the decoder has the most relevant information to perform the task of decoding.

The first step when designing a neural network based decoder is gathering data that will be used as the training dataset. However, as the code distance increases, the size of the space including all potential error syndromes gets exponentially large. Therefore, we need to decide at which point the sampling process is going to be terminated.

Based on the sampling probability (physical error rate), different error syndromes will be more frequent than others. We chose to include the most frequent error syndromes in the training dataset. In order to find the best possible dataset, we increase the dataset size until it stops yielding better results in terms of decoding performance. For each training dataset size, we train a neural network and evaluate the decoding performance.

It is not straightforward to claim that the optimal size of a training dataset is found, because there is no way to ensure that we found the minimum number of training samples that provide the best weights for the neural network, therefore generalization, after being perfectly trained. Thus, we rely heavily on the decoding performance that each training dataset achieves and typically use more training samples than the least amount required.

C. Structure of the neural network

While investigating the optimal size of a dataset, some preliminary investigation of the structure has been done, however only after the dataset is defined, the structure in terms of layers and nodes is explored in depth (see Figure 9).

A variety of different configurations of layers and nodes needs to be tested, so that the configuration with the highest accuracy of training in the least amount of training time can be adopted. The main factors that affect the structure of the neural network are the size of the training dataset, the similarity between the training samples and the type of neural network.

We found in our investigation that the number of layers selected for training are affected more by the samples, e.g. the similarity of the input samples, and less by the size of the training dataset. The number of nodes of the last hidden layer is selected to be equal to the number of output nodes. The rest of the hidden layers were selected to have decreasing number of nodes going from the first to the last layer, but we do not claim that this is necessarily the optimal strategy.

We implemented both decoder designs with feed-forward and recurrent neural networks. The more sophisticated recurrent neural network seems to outperform the feed-forward neural network in both the depolarizing and the circuit noise model. In Figure 10, it is evident that even for the small decoding problem of d=3 rotated surface code for the depolarizing error model, the RNN outperforms the FFNN in decoding performance. This is even more obvious at larger

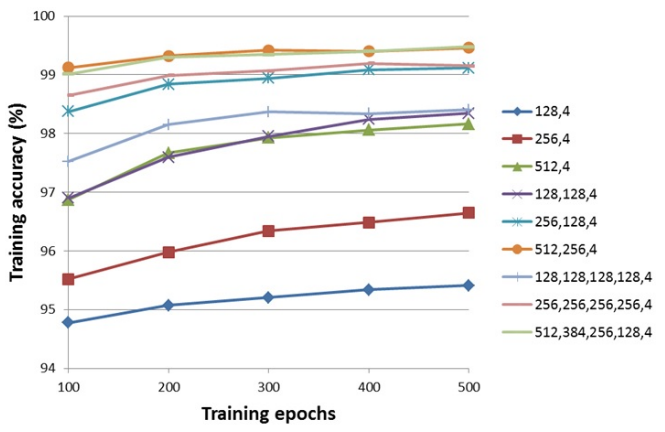


FIG. 9. Different configurations of layers and nodes for the d=3 rotated surface code for the ***depolarizing error model***. The nodes of the tested hidden layers are presented in the legend. Training stops at 500 training epochs for all configurations, since a good indication of the training accuracy achieved is evident by that point. Then, the one that reached the highest training accuracy continues training until the training accuracy cannot increase any more.

TABLE I. Pseudo-threshold values for the tested decoders (d=3) under depolarizing error model

Decoder	Pseudo-threshold
FFNN lld	0.0911
RNN lld	0.0949
FFNN hld	0.0970
RNN hld	0.0969
Blossom	0.0825

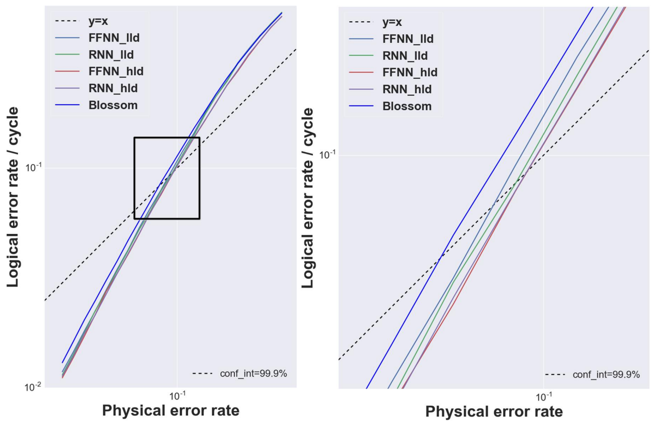


FIG. 10. Left: Comparison of decoding performance between Blossom algorithm, low level decoder and high level decoder for the ***d=3*** rotated surface code for the ***depolarizing error model***. Right: Zoomed in at the region defined by the square.

code distances and for the circuit noise model, where the recurrent neural network naturally fits better due to its nature. Moreover, training of the FFNN becomes much harder compared to the RNN as the size of the dataset increases, making the experimentation with FFNN even more difficult.

The metric that we use to compare the different designs is the ***pseudo-threshold***. The pseudo-threshold is defined as the highest physical error rate that the quantum device should operate in order for error correction to be beneficial. Operating at higher than the pseudo-threshold probabilities will cause worse decoding performance compared to an unencoded qubit. The protection provided by error correction is increasing as the physical error rate becomes smaller than the pseudo-threshold value, therefore a higher pseudo-threshold for a code distance signifies higher decoding performance.

The pseudo-threshold metric is used when a single code distance is being tested. When a variety of code distances are investigated, then we use the ***threshold*** metric. The threshold is a metric that represents the protection against noise for a family of error correcting codes, like the surface code. For the surface code, each code distance has a different pseudo-threshold value, but the threshold value of the code is only one.

The pseudo-threshold values for all decoders investigated in Figure 10 can be found as the points of intersection between the decoder curve and the black dashed line, which represents the points where the physical error probability is equal to the logical error probability ($y = x$). The pseudo-thresholds acquired from Figure 10 are presented in Table I.

The threshold value is defined as the point of intersection of all the curves of multiple code distances, therefore cannot be seen from Figure 10, since all curves involve d=3 decoders, but can be found in Figures 13 14 for the depolarizing and circuit noise model, respectively.

Another observation from Figure 10 and Table I is that the high level decoder is outperforming the low level decoder. Although there are ways to increase the decoding performance of the latter, mainly by re-designing the repetition step to find the valid corrections in less repetitions, we found no merit in doing so, since the decoding performance would still be similar to the high level decoder's and the repetition step would still not be eliminated.

Based on these observations, the results presented in Figures 13 and 14 in the Results section were obtained with the high level decoder with recurrent neural networks.

D. Training process

1. *Batch size*

Training in batches instead of the whole dataset at once, can be beneficial for the training accuracy and training time. By training in batches, the weights of the neural network are updated multiple times per training iteration, which typically leads to faster convergence. We used batches of 1000 or 10000 samples, based on the size of the training dataset.

2. *Learning rate*

Another important parameter of training that can directly affect the training accuracy and training time is the learning rate. The learning rate is the parameter that defines how big the updating steps will be for each weight at every training iteration. Larger learning rates in the beginning of training can lead the training process to a minimum faster during gradient descent, whereas smaller learning rates near the end of training can increase the training accuracy. Therefore, we devise a strategy of a step-wise decrease of the learning rate throughout the training process. If the training accuracy has not increased after a specified number of training iterations (e.g. 50), then the learning rate is decreased. The learning rates used range from 0.01 to 0.0001.

3. *Generalization*

The training process should not only be focused on the correct prediction of known inputs, but also the correct prediction of inputs unknown to training, known as generalization. Without generalization, the neural network acts as a Look-Up Table (LUT), which will lead to sub-optimal behavior as the code distance increases. In order to achieve high level of generalization, we continue training until the closeness between the desired and predicted value up to the 3^{rd} decimal digit is higher than 95% over all training samples.

4. *Training and execution time*

Timing is a crucial aspect of decoding and in the case of neural network decoders we need to minimize both the execution time and the training time as much as possible.

The training time is proportional to the size of the training dataset and the number of qubits. The number of qubits is increasing in a quadratic fashion, $2d^2 - 1$, and the selected size of the training dataset in our experimentation is increasing in an exponential way, 2^{d^2-1}. Therefore, training time should increase exponentially while scaling up.

However, the platform that the training occurs, affects the training time immensely, since training in one/multiple CPU(s) or one/multiple GPU(s) or a dedicated chip in hardware will result in widely different training times. The neural networks that were used to obtain the results in this work, required between half a day to 3 days, depending on the number of weights and the inputs/outputs of the neural network, on a CPU with 28 hyper thread cores at 2GHz with 384GB of memory.

In our simulations in a CPU, we observed the constant time behavior that was anticipated for the execution time, however a realistic estimation taking into account all the details of a hardware implementation that such a decoder might run, has not been performed by this or any other group yet. The time budget for decoding is different for different qubit technologies, however due to the inadequate fidelity of the quantum operations, it is extremely small for the time being, for example ~700nsec for a surface code cycle with superconducting qubits [19].

In Figure 11, we present the constant and non-constant execution time for the d=3 rotated surface code for the depolarizing noise model with perfect error syndrome measurements for the high level decoder and the low level decoder, respectively.

The low level decoder has to repeat its predictions before it predicts a valid set of corrections which makes the execution time non-constant. With careful design of the repetition step, the average number of predictions can decrease, however the execution time will remain non-constant. Based on the non-constant execution time and the inferior decoding performance compared to the high level decoder, the low level decoder was rejected.

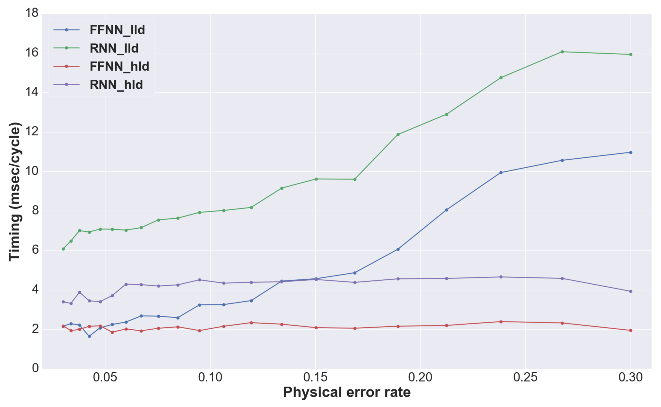


FIG. 11. Execution time for the high level decoder (hld) and the low level decoder (lld) for Feed-forward (FFNN) and Recurrent neural networks (RNN) for d=3 rotated surface code for the ***depolarizing error model***.

Moreover, the recurrent neural network typically uses more weights compared to the feed-forward neural network, which translates to higher execution time. However, the decoding performance and the training accuracy achieved with recurrent neural networks leads to better decoding performance. Thus, we decided to create high level decoders based on recurrent neural networks while taking into account all the parameters mentioned above.

The execution time for high level decoders appears to increase linearly with the number of qubits. This is justified by the fact that as the code distance increases, the operation of the simple decoder does not require more time, since all detection events are matched in parallel and independently to each other, and the size of the neural network increases in such a way that only a linear increase in the execution time is calculated. In Table II, we provide the calculated average time of decoding a surface code cycle under depolarizing noise for all distances tested with the high level decoder with recurrent neural networks.

TABLE II. Average time for surface code cycle under depolarizing error model

Code distance	Avg. time / cycle
d=3	4.14msec
d=5	11.19msec
d=7	28.53msec
d=9	31.34msec

There are factors such as the number of qubits, the type of neural network being used and the number of inputs/outputs of the neural network that influence the execution time. The main advantage against classical algorithms is that the execution time of such neural network based decoders is independent of the physical error probability.

In the following section we are presenting the results in terms of the decoding performance for different code distances.

VI. Results

As we previously mentioned, the way that decoding performance is tested is by running simulations that sweep a large amount of physical error rates and calculate the corresponding logical error rate for each of them. This type of simulations are frequently referred to as lifetime simulations and the logical error is calculated as the ratio of logical errors found over the error correction cycles performed to accumulate these logical errors.

The design of the neural network based decoder that was used to obtain the results is described in Figure 12 for the depolarizing and the circuit error model. For the case of the ***depolarizing error model***, neural network 1 is not used, so the input is forwarded directly to the simple decoder since perfect syndrome measurements are assumed. The decoding process is similar to the one presented in Figure 8.

The decoding algorithm for the ***circuit noise model*** consists of a simple decoder and 2 neural networks. Both neural networks receive the error syndrome as input. Neural network 1 predicts which detection events at the error syndrome belong to data qubit errors and which belong to measurement errors. Then, it outputs the error syndrome relieved of the detection events that belong to measurement errors to the simple decoder. The simple decoder provides a set of corrections based on the received error syndrome. Neural network 2 receives the initial error syndrome and predicts whether the simple decoder will make a logical error and outputs a set of corrections which combine with the simple decoder corrections at the output.

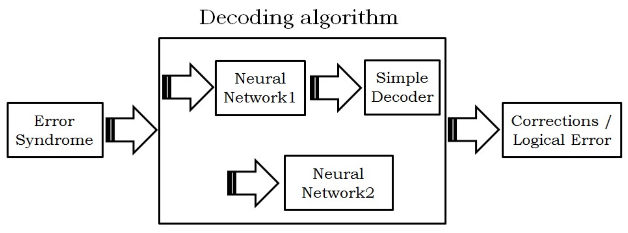


FIG. 12. The design for the high level decoder that was used for the depolarizing and the circuit noise model.

A. Depolarizing error model

For the depolarizing error model, we used 5 training datasets that were sampled at these physical error rates : 0.2, 0.15, 0.1, 0.08, 0.05. Perfect error syndrome measurements are assumed, so the logical error rate can be calculated per error correction cycle.

In Table III, we present the pseudo-thresholds achieved from the investigated decoders for the depolarizing error model with perfect error syndrome measurements for different distances. As expected, when the distance increases, the pseudo-threshold also increases. Furthermore, the neural network based decoder with the multiple probabilities datasets exhibits higher pseudo-threshold values, which is expected since it has more relevant information in its dataset.

As can be seen from Figure 13, the multiple probabilities datasets approach is providing better decoding performance for all code distances simulated. The fact that the

TABLE III. Pseudo-threshold values for the depolarizing error model

Decoder	d=3	d=5	d=7	d=9
Blossom	0.08234	0.10343	0.11366	0.11932
Single prob. dataset	0.09708	0.10963	0.12447	N/A
Multiple prob. dataset	0.09815	0.12191	0.12721	0.12447

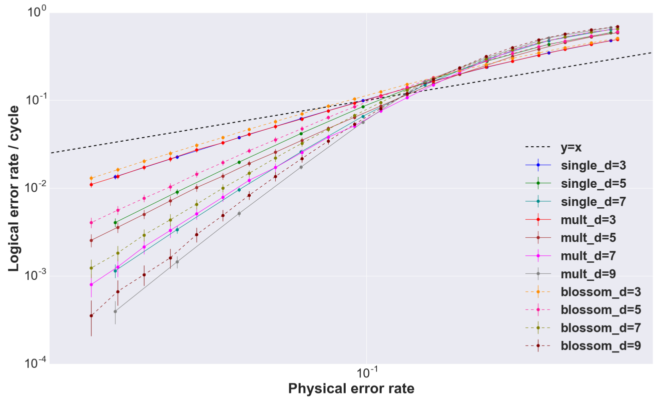


FIG. 13. Decoding performance comparison between the high level decoder trained on a single probability dataset, the high level decoder trained on multiple probabilities datasets and Blossom algorithm for the ***depolarizing error model*** with perfect error syndrome measurements. Each point has a confidence interval of 99.9%.

high level decoder is trained to identify the most frequently encountered error syndromes based on a given physical error rate, results in more accurate decoding information. Another reason for the improvement against the Blossom algorithm, is the ability of identifying correlated errors (-iY=XZ). For the depolarizing noise model with perfect error syndrome measurements, the Blossom algorithm is proven to be near-optimal, so we are not able to observe a large improvement in the decoding performance. Furthermore, the comparison is against the un-optimized version of Blossom algorithm [21], therefore it is mainly performed to get a frame of reference rather than an explicit numerical comparison.

We observe that for the range of physical error rates that we are interested in, which are below the pseudo-threshold, the improvement against Blossom algorithm is reaching up to 18.7%, 58.8% and 53.9% for code distance 3, 5 and 7, respectively for the smallest physical error probabilities tested.

The threshold of the rotated surface code for the depolarizing model has improved from 0.14 for the single probability dataset approach to 0.146 for the multiple probabilities datasets approach. The threshold of Blossom is calculated to be 0.142.

B. Circuit noise model

For the circuit noise model, we used 5 training datasets that were sampled at these physical error rates : 4.5x10^{-3}, 1.5x10^{-3}, 8.0x10^{-3}, 4.5x10^{-4}, 2.5x10^{-4}. Since, imperfect error syndrome measurements are assumed the logical error rate is calculated per window of d error correction cycles.

In Table IV, we present the pseudo-thresholds achieved for the circuit noise model with imperfect error syndrome measurements. Again, the neural network based decoder with multiple probabilities datasets is performing better than the single probability dataset. We were not able to use the Blossom algorithm with imperfect measurements for code distances higher than 3, therefore we decided not to include it. However, we note that the results that were obtained are similar to the results in the literature corresponding to the circuit noise model [23, 24].

TABLE IV. Pseudo-threshold values for the circuit noise model

Decoder	d=3	d=5	d=7
Single prob. dataset	$3.99\text{x}10^{-4}$	$9.23\text{x}10^{-4}$	N/A
Multiple prob. dataset	$4.44\text{x}10^{-4}$	$1.12\text{x}10^{-3}$	$1.66\text{x}10^{-3}$

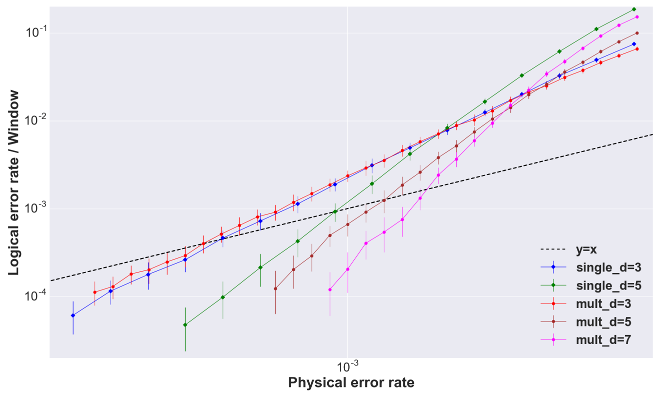


FIG. 14. Decoding performance comparison between the high level decoder trained on a single probability dataset and the high level decoder trained on multiple probabilities datasets for the ***circuit noise model*** with imperfect error syndrome measurements. Each point has a confidence interval of 99.9%.

We observe from Figure 14 that the results with the multiple probabilities datasets for the circuit noise model are significantly better, especially as the code distance is increased. The case of the d=3 is small and simple enough to be solved equally well by both approaches. The increased decoding performance achieved with the multiple probabilities datasets approach is based on the more accurate information for the physical error probability that is being tested.

The threshold of the rotated surface code for the circuit noise model has improved from $1.77\text{x}10^{-3}$ for the single probability dataset approach to $3.2\text{x}10^{-3}$ for the multiple probabilities datasets approach, that signifies that the use of dedicated datasets when decoding a given physical error rate is highly advantageous.

As mentioned, the single probability dataset is collected at a low physical error rate, for example around the pseudo-threshold value. Therefore, the size of the training dataset is similar for both the single and the multiple probabilities datasets for the low physical error rates. For higher physical error rates, we gather larger training datasets for the multiple probabilities datasets approach, which are also more relevant.

The space that needs to be sampled is getting exponentially larger to a point that is infeasible to gather enough samples to perform good decoding beyond d=7. The reason for this exponential growth is due to the way that we provide the data to the neural network. Currently, we gather all error syndromes out of all the error correction cycles and create lists out of them. Then, we provide these lists to the recurrent neural network all-together. Since the recurrent neural network can identify patterns both in space and time, we also provide the error correction cycle that provided that error syndrome (time stamp of each error syndrome). Then, the recurrent neural network is able to differentiate between consecutive error correction cycles and find patterns of errors in them.

In order to obtain efficient decoding regardless of the exponentially large state space, we restrict the space that we sample to the one containing the most frequent error syndromes occurring at the specified sampling (physical) error probability. However, even by employing such a technique, it seems impossible to continue beyond d=7 for the circuit noise model with the decoding approach that we used in this work. At the circuit noise model for d=7 for example, we gather error syndromes out of 10 error correction cycles and each error syndrome contains 48 ancilla qubits. Therefore, the full space that needs to be explored is 2^{10*48}, which is infeasible.

A different approach that minimizes the space that the neural network needs to search would be extremely valuable. A promising idea would be to provide the error syndromes of each error correction cycle one at a time, instead of giving them all-together, and keep an intermediate state of the logical qubit.

VII. Conclusions

This work focused on researching various design strategies for neural network based decoders. Such kind of decoders are currently being investigated due to their good decoding performance and constant execution time. They seem to have an upper limit at around 160 qubits, however by designing smarter approaches in the future, we can have neural network based decoders for larger quantum systems.

We emphasized mainly on the design aspects and the parameters that affect the performance of the neural networks and devised a detailed plan on how to approach them. We showed that we can have high decoding performance for quantum systems of about 100 qubits for both the depolarizing and the circuit noise model. We showed that a neural network based decoder that uses the neural network as an auxiliary module to a classical decoder leads to higher decoding performance.

Furthermore, we presented the constant execution time of such a decoder and showed that it increases linearly with the code distance in our simulations. We compared different types of neural networks, in terms of decoding performance and execution time, concluding that recurrent neural networks can be more powerful than feed-forward neural networks for such applications.

Finally, we showed that having a dedicated dataset for the physical error rate that the quantum system operates can increase the decoding performance.

References

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

[2] T. E. O'Brien, B. Tarasinski, and L. DiCarlo, "Density-matrix simulation of small surface codes under current and projected experimental noise," *npj Quantum Information*, vol. 3, 2017.

[3] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965. [Online]. Available: http://dx.doi.org/10.4153/CJM-1965-045-4

[4] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002. [Online]. Available: http://dx.doi.org/10.1063/1.1499754

[5] A. G. Fowler, "Optimal complexity correction of correlated errors in the surface code," *arXiv:1310.0863*, 2013.

[6] A. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time," *arXiv:1307.1740v3*, 2014.

[7] S. Varsamopoulos, B. Criger, and K. Bertels, "Decoding small surface codes with feedforward neural networks," *Quantum Science and Technology*, vol. 3, no. 1, p. 015004, 2018. [Online]. Available: http://stacks.iop.org/2058-9565/3/i=1/a=015004

[8] P. Baireuther, T. E. O'Brien, B. Tarasinski, and C. W. J. Beenakker, "Machine-learning-assisted correction of correlated qubit errors in a topological code," *Quantum*, vol. 2, p. 48, Jan. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-01-29-48

[9] G. Torlai and R. G. Melko, "Neural decoder for topological codes," *Phys. Rev. Lett.*, vol. 119, p. 030501, 7 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.119.030501

[10] S. Krastanov and L. Jiang, "Deep neural network probabilistic decoder for stabilizer codes," *Scientific Reports*, vol. 7, no. 11003, 2017.

[11] C. Chamberland and P. Ronagh, "Deep neural decoders for near term fault-tolerant experiments," *Quantum Science and Technology*, vol. 3, no. 4, p. 044002, 2018. [Online]. Available: http://stacks.iop.org/2058-9565/3/i=4/a=044002

[12] M. Maskara, A. Kubica, and T. Jochym-O'Connor, "Advantages of versatile neural-network decoding for topological codes," *arXiv:1802.08680*, 2018.

[13] A. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2 – 30, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003491602000180

[14] M. H. Freedman and D. A. Meyer, "Projective plane and planar quantum codes," *Foundations of Computational Mathematics*, vol. 1, no. 3, pp. 325–332, 2001.

[15] S. B. Bravyi and A. Y. Kitaev, "Quantum codes on a lattice with boundary," *arXiv preprint quant-ph/9811052*, 1998.

[16] H. Bombin and M. A. Martin-Delgado, "Optimal resources for topological two-dimensional stabilizer codes: Comparative study," *Physical Review A*, vol. 76, no. 1, p. 012305, 2007.

[17] A. G. Fowler, A. M. Stephens, and P. Groszkowski, "High-threshold universal quantum computation on the surface code," *Physical Review A*, vol. 80, no. 5, p. 052312, 2009.

[18] D. A. Lidar and T. A. Brun, *Quantum Error Correction.* Cambridge University Press, 2013.

[19] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, "Scalable quantum circuit and control for a superconducting surface code," *Phys. Rev. Applied*, vol. 8, p. 034021, Sep 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.8.034021

[20] S. Bravyi, M. Suchara, and A. Vargo, "Efficient algorithms for maximum likelihood decoding in the surface code," *Phys. Rev. A*, vol. 90, p. 032326, Sep 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.90.032326

[21] V. Kolmogorov, "Blossom V: a new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, vol. 1, pp. 43–67, 2009. [Online]. Available: http://dx.doi.org/10.1007/s12532-009-0002-8

[22] I. Changhau, "Lstm and gru – formula summary," July 2017. [Online]. Available: https://isaacchanghau.github.io/post/lstm-gru-formula/

[23] A. G. Fowler, A. C. Whiteside, A. L. McInnes, and A. Rabbani, "Topological code autotune," *PHYSICAL REVIEW X*, vol. 2, p. 041003, 2012.

[24] A. G. Fowler, A. M. Stephens, and P. Groszkowski, "High threshold universal quantum computation on the surface code," *arXiv:0803.0272*, 2012.