Comparing neural network based decoders for the surface code

Savvas Varsamopoulos,[1,*] Koen Bertels,[1] and Carmen Garcia Almudever[1]

[1]*Quantum and Computer Engineering, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands*

Matching algorithms can be used for identifying errors in quantum systems, being the most famous the Blossom algorithm. Recent works have shown that small distance quantum error correction codes can be efficiently decoded by employing machine learning techniques based on neural networks (NN). Various NN-based decoders have been proposed to enhance the decoding performance and the decoding time. Their implementation differs in how the decoding is performed, at logical or physical level, as well as in several neural network related parameters. In this work, we implement and compare two NN-based decoders, a low level decoder and a high level decoder, and study how different NN parameters affect their decoding performance and execution time. Crucial parameters such as the size of the training dataset, the structure and the type of the neural network, and the learning rate used during training are discussed. After performing this comparison, we conclude that the high level decoder based on a Recurrent NN shows a better balance between decoding performance and execution time and it is much easier to train. We then test its decoding performance for different code distances, probability datasets and under the depolarizing and circuit error models.

Quantum computers are a promising solution to a class of complex problems that classical supercomputers cannot currently solve or require an immensely large amount of time to solve. However, since quantum computing is still in its early stages, classical computers are still the driving force, using the prototypes of quantum computers as accelerators for specific applications.

In the recent past, there is an increasing dominance of heterogeneous, multi-core architectures with multiple processors. In such architectures, a classical core processor interacts with different co-processors such as Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), Tensor Processing Units (TPUs) and in this case a quantum processor. Such a quantum processor requires both classical and quantum computing components, because it needs a lot of monitoring and control from the classical part. Typically, when developing such an architecture, one has to develop a full stack going from algorithms up to the chip implementation.

Figure 1 provides an overview of the quantum system stack consisting of the following layers [1]:

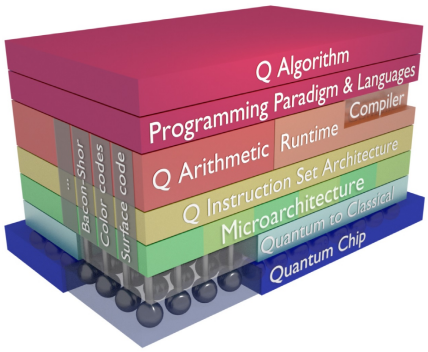

FIG. 1. Overview of the quantum computer system stack

- ◇ The top layers involve the quantum algorithms alongside the language constructs and compilers that are required to generate a series of instructions that belong to the Quantum Instruction Set Architecture (QISA).
- ◇ The micro-architecture layer translates these instructions into pulses to operate in the quantum chip. These pulses are sent through the quantum to classical interface.
- ◇ As can be seen by the 3^{rd} dimension of Figure 1, quantum error correction (QEC) is a key part when building a fault-

* svarsamo@gmail.com

tolerant quantum accelerator and it affects several layers of the stack, including the micro-architecture.

As shown in Figure 2, different micro-architectural blocks are required to keep track of the errors and identify the location and type of these errors such as the Decoding block and the Pauli frame unit. In this work, we focus on the Decoder logic block of Figure 2 that is part of the QEC process, as we will explain in the next paragraphs.

Constant active quantum error correction 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). In this paper, we limit ourselves to ***surface code*** encoding [3, 4]. 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 [5]. Many classical decoding algorithms have been proposed with the most widely used being the ***Blossom decoder*** [6]. Blossom has been shown to reach high decoding accuracy, but its execution time scales polynomially with the number of qubits [7], 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 [8] and even a parallel version stating that the average processing time per detection round is constant independent of the size of the system [9]. Also, there exist other decoders like the union find decoder that is presented in [10], which report an almost linear scaling of the execution time as the quantum system increases linearly. In addition, Blossom's execution time also depends on the physical error rate. Blossom performs a Minimum Weight Perfect Matching (MWPM) on a graph that is created based on the amount of errors that have been generated and detected. Then, higher error rates lead to bigger graphs and longer execution time. In this case, the execution time scales polynomially with the physical error rate.

An alternative to classical decoders is to use a neural network for identifying errors. They exhibit constant execution time with the physical error rate and their execution time scales linearly with the linear increase of the number of qubits. Also, they have been proven to provide better decoding performance than many classical decoding algorithms [11–17]. Note that, most of neural network based decoders use the neural network as a probability distribution, however there exist decoders that create an exact mapping between belief propagation and deep neural networks, as presented in [18].

Neural network based decoder implementations differ in how the neural network performs the decoding, the type and structure of the network, the amount of samples used for training and many other aspects. Therefore, their decoding performance and their reported execution time also differs depending on the implementation choices that were made.

The contributions of this paper can be summarized as follows:

1. So far, there is no thorough comparison made between different neural network based decoders under the same conditions. In this paper, we have implemented two different NN-based decoders, namely the high level and low level decoder, and analyzed their decoding performance and execution time while exploring different NN parameters. We show that the high level decoder

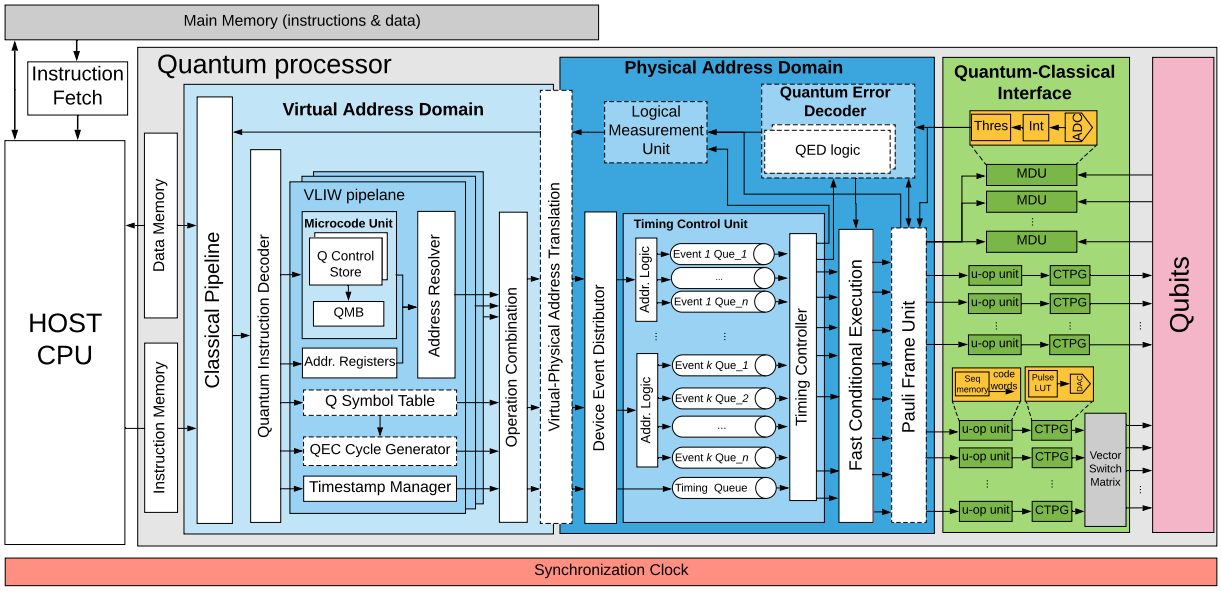

FIG. 2. Overview of the quantum micro-architecture [2]

exhibits constant execution time regardless of the physical error rate and scales linearly with the linear increase of the qubits in the quantum system, while able to reach at least equivalent decoding performance to the Blossom decoder (baseline).

2. Usually when NN-based decoders are designed, achieving the highest decoding performance is the main goal, however, in this work we emphasize the importance of small execution time, due to the limited available time budget of quantum error correction. Therefore, the neural network based decoders implemented in this paper should present a good balance between decoding performance and execution time.

3. We analyze how the choice of the dataset affects the training of neural network based decoders. We show that sampling and training at the different physical error rates that the decoder is being evaluated, leads to higher decoding performance than sampling and training to a single physical error rate and evaluating the decoder at a vastly different error rate.

The rest of the paper is organized as follows: in section I we provide a brief introduction to quantum error correction. In section II, we explain how the different NN-based decoders, as found in literature, are implemented. In section III, many parameters of the neural network based decoders are discussed. In section IV, we provide the results with the best neural network based decoder for the different error models. Finally, in section V, we draw our conclusions about this research.

I. 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 [3, 4, 19–21], 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 [7, 20, 22–28]. It is usually defined as a planar lattice of qubits over two dimensions.

In the surface code, a logical qubit consists of physical qubits that store quantum informa-

tion, known as ***data*** qubits, and physical qubits that are used to detect errors in the logical qubit through their measurement, known as ancillary or ***ancilla*** qubits (see Figure 3). 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.

An important feature of the surface code is the code distance. ***Code distance***, (***d***), describes the degree of protection against errors. More accurately, is the minimum number of physical operations required to change the logical state [3, 29]. In surface code, the degree of errors (d.o.e.) that can be successfully corrected, is calculated according to the following equation:

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

The smallest surface code created is known as the rotated surface code [30] and it is presented in Figure 3. 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.

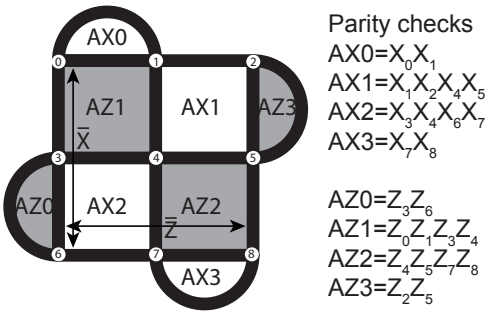

FIG. 3. Rotated surface code with code distance 3. Data qubits are enumerated from 0 to 8. X-type ancilla are in the center of the white tiles and Z-type ancilla are in the center of grey tiles

As mentioned, ancilla qubits are used to detect errors in the data qubits. Although quantum errors are continuous, the measurement outcome of each ancilla discretizes quantum errors into bit-flip (X) and phase-flip (Z) errors, that can be detected by Z-type ancilla and X-type ancilla, 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 4 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 binary value, which 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 4. 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***.

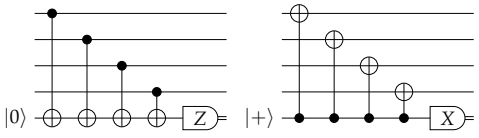

FIG. 4. Syndrome extraction circuit for individual Z-type (left) and X-type (right) ancilla, with the ancilla placed in the bottom

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 [31].

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 example of these parity-checks for a d=3 rotated surface code is presented in Figure 3. The notation X_i or Z_i refers to the i^{th} data qubit used in a given parity-check.

Gathering all measurement outcomes, forms the ***error syndrome***. Surface code 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.

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. 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 5 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 5 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. Therefore, to properly distinguish between data and measurement errors, multiple error correction cycles need to be run before corrections are proposed.

There exist decoding algorithms that can decode efficiently the surface code, however optimal decoding is a NP-hard problem [32]. For example, maximum likelihood decoding (MLD) searches for the most probable error that produced the error syndrome, whereas Blossom searches for the least amount of errors that produced the error syndrome [8, 33]. MLD has a significantly higher decoding performance

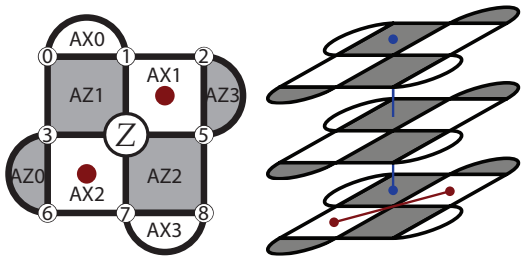

FIG. 5. Rotated surface code with code distance 3. Left: Phase-flip (Z) error at data qubit 4, which causes two detection events (shown in red). Right: Three consecutive rounds of error correction. The red dots indicate detection events that arise from a data qubit error and the blue dots indicate detection events that arise from a measurement error

than Blossom. However,it has an exponentially increasing execution time while the code distance increases linearly. For that reason, approximate versions of MLD have been developed like the [34], which reports a running time of $O(n\chi^3)$, where χ is a parameter that controls the approximation precision.

Although the Blossom decoder reaches lower decoding performance than MLD, the execution time of the optimized version exhibits linear scaling with the number of qubits. Furthermore, note that there exists a parallel version of Blossom described in [9], that claims constant execution time regardless of the size of the system. Therefore, there is a clear trade-off between decoding performance and execution time, which is a key aspect of the decoder as there is a limited time budget for error correction.

The time budget for decoding is calculated based on the time required by the quantum operations involved in an error correction cycle. Therefore, the time budget varies based on the type of quantum technology, the choice of quantum error correcting code and the way quantum operations are performed. For example, in the case of superconducting qubits for a d=3 rotated surface code reported in [5, 31], the time budget for an error correction cycle is calculated to be ˜700nsec, making most decoders unusable for near-term experiments based on these parameters. Furthermore, if noisy error syndrome measurements are 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. However, as quantum technology matures and the fidelity of quantum operations increase, the time budget for QEC will also change.

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 discuss decoders that include neural networks and compare them to each other and to the un-optimized version of the Blossom algorithm as described in [35]. In the following two sections we describe the implementation details of the NN-based decoders and the tuning of certain NN parameters.

II. Neural network based decoders

Artificial neural networks (ANN) have been shown to reach high application performance and constant execution time after being trained on a set of data generated by the application. ANN 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). In this work, we focus on two types of neural networks known as ***Feed-forward neural networks (FFNN)*** and ***Recurrent neural networks (RNN)***. In the case of RNNs, we used ***Long Short-Term Memory (LSTM) cells***. Since it was easy to generate input data and their corresponding output from the simulations, we used ***supervised learning*** to train the neural network. Finally, we used the ***mean squared error rate*** as a cost function and the ***Rectified Linear Unit (ReLU)*** as the activation function.

Neural network based decoders for quantum error correcting codes have been recently proposed [11–14, 16, 17]. 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*** (lld) [13, 14] and the latter ones as ***high level decoders*** (hld) [11, 12, 16, 17].

In this paper, we implement both decoders and compare them to each other and with the un-optimized Blossom decoder. The comparison is in terms of the decoding performance and the execution time. In the next sections, each decoder implementation is explained and the differences between them are presented.

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 III.

C. Implementation details

Implementations of 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 6a and b), since the decoder is predicting corrections for a different error syndrome than the one that was observed. In such a case the decoding performance will decrease significantly. Only when the two error syndromes match, the predictions are used as corrections on the data qubits (see Figure 6a and 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 6, the decoding procedure of the low level decoder is described with an example. On 6a, 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 6b, 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 6a and 6b, the two error syndromes do not match, therefore the predicted corrections are deemed **invalid**. On 6c, 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 6a and 6c, the predicted error syndrome matches the observed one, therefore the corrections are deemed **valid**.

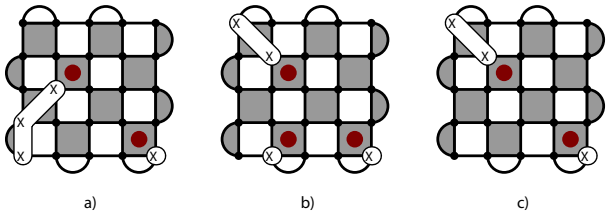

FIG. 6. Description of the decoding process of the low level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) **Invalid** data qubits corrections and the corresponding error syndrome. (c) **Valid** data qubits corrections and the corresponding error syndrome

Implementations of 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 [12] or one can be a neural network and the other one a non-neural network module[11, 16]. The non-neural network 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 [11], this 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 7).

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 7, the decoding procedure of the high 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, we present the decoding of the 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 error. In Figure 7b, 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 7a-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 7c.

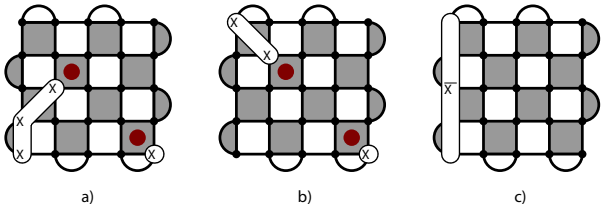

FIG. 7. Description of the decoding process of the high level decoder for a d=5 rotated surface code. (a) Observed error syndrome shown in red dots and bit-flip errors on physical data qubits shown with X on top of them. (b) Corrections proposed by the simple decoder for the observed error syndrome. (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

III. Implementation parameters

In this paper, we compare both decoder designs. To achieve that, we implement the high level and low level decoder and test them under the same conditions. We investigate how different implementation parameters affect the following metrics:

1. **The decoding performance:** it 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 = d cycles). 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.

2. **The execution time:** it 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. For the low level decoder, the execution time will be the inference time of the neural network. For the high level decoder, the execution time will be the determined as the maximum time between the execution time of the simple decoder and the inference time of

the neural network.

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

A. Choosing the training dataset

The first step when designing a neural network based decoder is gathering data that will be used as the training dataset. The best dataset for a neural network based decoder is the dataset that achieves the highest decoding performance. Naively, one could suggest that including all possible error syndromes, would lead to the best decoding performance, however, as the code distance increases (size of the system increases), including all error syndrome becomes infeasible. As the code distance increases, the size of the space including all potential errors gets exponentially large as shown in Table I (state space). Therefore, we need to include as little but as diverse as possible error syndromes, which will provide the maximum amount of generalization. The size of the training dataset that we used for each code distance is also presented in Table I (training dataset). As shown, by employing such a technique, it seems impossible to continue beyond d=7 for the circuit noise model. For that distance, 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 $3.1\text{x}10^{144}$. For the depolarizing error model, the highest code distance that we were able to decode efficiently was d=9. A potential candidate to assist in overcoming this scalability challenge and go to higher code distances is a distributed decoding strategy as found in [36] and [37].

The idea of using a distributed technique is to avoid processing all the error information from the whole code all-together. By limiting the amount of error information that is being processed at all times, the error syndrome space is always limited, thus making it easier to scale to larger code distances. In [36], a neural network decoder that is designed based on the Renormalization Group (RG) technique is explained. In RG decoding, the code is initially divided into small overlapping regions. Then, the error information out of these regions is combined to represent the error information of larger regions. This process of updating the error information while moving from a smaller region to a larger one is continued until the whole code ends up being one single region. In [37], a similar approach is described. The main difference is that the code is always divided into specific size of overlapping regions and the information out of these small regions is forwarded to the whole code at once. Therefore, both techniques rely on dividing the code into overlapping regions and then updating the error information of each region in order to decode the whole code. Based on such a scheme, the maximum number of error information, which is translated into the maximum number of inputs of the neural network decoder, is always capped to a value that is much smaller compared to the case of the non-distributed approach.

In our previous experimentation[11], we showed that sampling at a single physical error rate that always produces the fewest amount

TABLE I. Dataset sizes for the depolarizing and the circuit noise error model

Code distance	State space Depolarizing/ Circuit	Training dataset Depolarizing/ Circuit
d=3	256 / $4.3\text{x}10^{9}$	256 / $5\text{x}10^{5}$
d=5	$1.6\text{x}10^{7}$ / $4.3\text{x}10^{9}$	$2\text{x}10^{5}$ / $2\text{x}10^{6}$
d=7	$2.8\text{x}10^{14}$ / $3.1\text{x}10^{144}$	$3\text{x}10^{6}$ / $2\text{x}10^{7}$
d=9	$1.2\text{x}10^{24}$ / $9.7\text{x}10^{288}$	$2\text{x}10^{7}$ / -

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 realistic for an experiment, since in an experiment there is only one 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 due 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. When training the neural network, only one of these sets of errors, and always the same, is going to be selected for a given error syndrome regardless of the physical error rate being tested. Therefore, training a neural network based decoder in one physical error rate and testing its decoding performance in a widely different physical error rate might lead to poor decoding performance. 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 are tested. In the single probability dataset approach, the set with the fewer errors is always selected, because this set is more probable for the range of physical error rates that we are interested in.

To achieve a better decoding performance, we created multiple datasets that were obtained by sampling at various physical error rates and trained a different neural network at each physical error rate that was sampled. We call this approach, the ***multiple probabilities datasets*** approach. Each dedicated training dataset that was created by sampling at a specific physical error probability is used to test the decoding performance at that same physical error probability and the probabilities close to that. 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.

As we mentioned, depending 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. Since the objective of the decoding performance is to reach at least equivalent decoding performance to Blossom (baseline) with the smallest possible dataset, we increase the dataset size until it reaches Blossom's performance. Note that we do not claim to find the optimal training dataset to maximize the decoding performance, rather have a good balance between decoding performance and execution time. As we mentioned, we take the Blossom's performance as a baseline since it is the most popular decoder. Due to the multiple probabilities datasets, we train each dataset and evaluate the decoding performance for that probability and the ones close to it.

The results of using a single probability dataset versus multiple probabilities datasets are shown in Figures 12 and 13 and will be dis-

cussed in Section 5.

B. Structure of the neural network

While investigating the 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.

Initially, different configurations of layers and nodes have been tested for: i) distance 3 and 5 surface code, ii) the high- and low level decoder, iii) RNN and FFNN, and iv) the depolarizing error model. In our configurations, 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. As an example, in Figure 8 we present the investigation of multiple configurations of nodes and layers for the d=3 high level decoder using a RNN. Training stops at 5000 training epochs, since a good indication of the training accuracy achieved is evident by that point. Then, the one that reached the highest training accuracy was selected (in this case we chose the 16, 4). We continued training it till at least the same decoding performance as Blossom was reached.

In addition, we observed through this experimentation that the main factors that affect the configuration of the neural network are the size of the training dataset, the similarity between the training samples and the type of neural network. For instance, we found 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.

As we just mentioned, after selecting the configuration that showed the highest training accuracy we continue training it. In Figure 9, the decoding performance of the high level and low level decoder with FFNN and RNN and Blossom are shown. For even the small decoding problem of d=3, the more sophisticated Recurrent neural network seems to slightly outperform the Feed-forward neural network. This can also be seen in Table II, where the ***pseudo-threshold*** values for d=3 and d=5

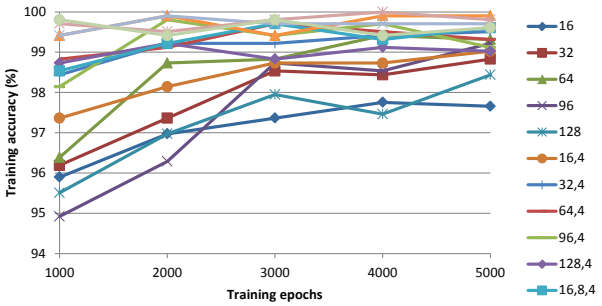

FIG. 8. Different configurations of layers and nodes for the high level decoder using a RNN (d=3, depolarizing error model). The nodes of the tested hidden layers are presented in the legend

are shown. 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 for a given code distance. Then, the highest the pseudo-threshold, the highest the decoding performance. The pseudo-threshold values for all decoders investigated in Figure 9 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 ($y = x$).

TABLE II. Pseudo-threshold values for the tested decoders (d=3 and d=5) under depolarizing error model

Decoder	Pseudo-threshold d=3	Pseudo-threshold d=5
FFNN lld	0.0911	0.1124
RNN lld	0.0949	0.1172
FFNN hld	0.0970	0.1219
RNN hld	0.0970	0.1233
Blossom	0.0825	0.1037

Note that, the difference between the decoding performance of the RNN and FFNN is even more obvious at larger code distances and for the circuit noise model, where the RNN naturally fits better due to its temporal dynamic behavior. 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.

Another observation from Figure 9 and Table II is that the high level decoder is outper-

forming 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.

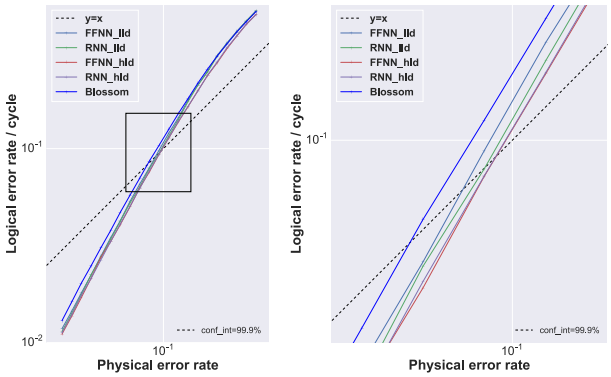

FIG. 9. Left: Comparison of decoding performance between Blossom algorithm, low level decoder and high level decoder for the d=3 rotated surface code for the depolarizing error model. Right: Zoomed in at the region defined by the square

C. 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 stepwise 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 time and execution time*

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

Training time: it 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. It 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.

Execution time: In Figure 10, we present the constant and non-constant execution time with

the physical error rate for the high level decoder and the low level decoder, respectively. The low level decoder has to repeat its predictions before it provides 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.

Note that results shown in Figure 10 are only to illustrate the behavior of the execution time of the two decoders, therefore the time scale is not representative. In order to obtain a realistic time estimate, a hardware implementation should be made which has not been performed by this or any other group yet.

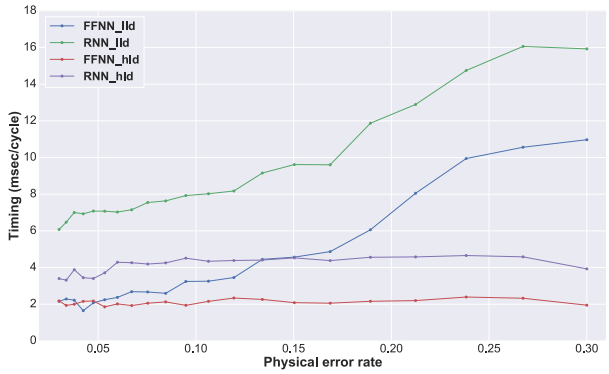

FIG. 10. Execution time for the high level decoder (hld) and the low level decoder (lld) for Feed-forward (FFNN) and Recurrent neural network (RNN) for d=3 rotated surface code for the depolarizing error model

As seen in Figure 10, the RNN has a higher execution time compared to the FFNN. This is due to the RNN typically using more weights compared to the FFNN.

Although the execution time of the high level decoders is constant with the error rate, it appears to increase linearly with the code distance. In Table III, 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 RNNs.

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.

Based on the analysis presented in this section, we can conclude that:

TABLE III. 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

1. High level decoders can achieve higher decoding performance compared to low level decoders and they exhibit constant execution time with the physical error rate.

2. Recurrent neural networks reach higher decoding performance compared to Feed-forward neural networks, mainly due to the use of more weights, but that leads to a higher execution time.

3. Recurrent neural networks are much easier to train compared to Feed-forward neural networks, and have the inherent notion of time, which is beneficial for the circuit noise model. Training and evaluating a decoder based on Feed-forward neural networks becomes extremely difficult as the code distance increases.

Based on these observations, we decided to create high level decoders based on Recurrent neural networks, in order to have a good balance between execution time and decoding performance. As we mentioned, to accomplish that balance, we search for the smallest dataset that can achieve equivalent or better decoding performance than Blossom (baseline). Increasing the number of samples in the training dataset will lead to better decoding performance at the cost of a bigger neural network requiring more training time and larger execution time. In addition, Recurrent neural networks are easier to train, allowing to decode higher code distances.

IV. Results

In this section, we analyze the decoding performance of the high level decoder based on

RNN for: i) different code distances, ii) the depolarizing and circuit error model, and iii) both datasets, single and multiple probabilities.

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. Decoding performance can also be evaluated by the pseudo-threshold for a given code distance or by the threshold for a set of several code distances for the same QEC code. The threshold represents the protection against noise for a family of error correcting codes, like the surface code. The threshold value is defined as the point of intersection of all the curves of multiple code distances (see Figures 12 and 13).

The flow from input to output of the neural network based decoder that was used to obtain the results is described in Figure 11 for both the depolarizing and the circuit error model. For the case of the depolarizing error model, neural network 1 is not used, so the error syndrome is forwarded directly to the simple decoder since perfect syndrome measurements are assumed. The decoding process is similar to the one presented in Figure 7.

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 (updated) 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.

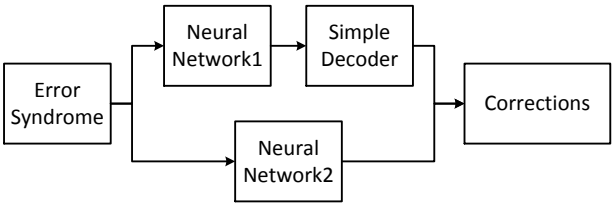

FIG. 11. The flow from input to output for the high level decoder that was used for the depolarizing and the circuit noise model

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.

Figure 12 shows the decoding performance achieved by the high level decoder when trained using single and multiple probabilities datsets and Blossom (baseline) under the depolarizing error model for different code distances (d= 3, 5, 7, 9). The neural network based decoder with the multiple probabilities datasets exhibits higher decoding performance (lower logical error rate), which is expected since it has more relevant information in its dataset. This can also be seen in Table IV, where the multiple probabilities datasets show higher pseudo-thresholds than the single probability dataset. Note that, the NN-based decoder has been trained to have similar or better decoding performance than Blossom, which is also reflected by the pseudo-threshold values.

TABLE IV. Pseudo-threshold values for the depolarizing error model. The pseudo-threshold increases as the code distance increases

Decoder	d=3	d=5	d=7	d=9
Blossom	0.0823	0.1034	0.1137	0.1193
Single prob. dataset	0.0971	0.1096	0.1245	N/A
Multiple prob. dataset	0.0982	0.1219	0.1272	0.1245

The threshold of the rotated surface code for the depolarizing model has improved from

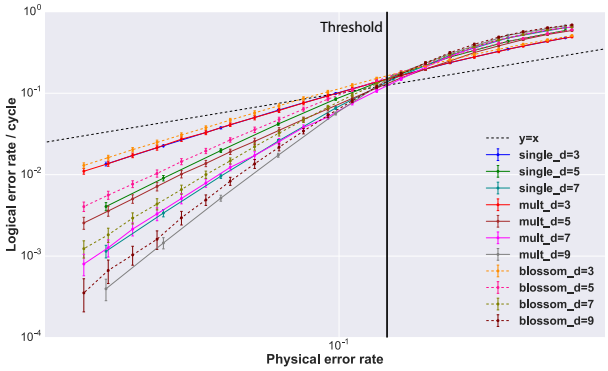

FIG. 12. Decoding performance comparison between the high level decoder trained on a single probability dataset, the high level decoder trained on multiple probabilities datasets and Blossom decoder for the depolarizing error model with perfect error syndrome measurements. Each point has a confidence interval of 99.9%

0.140 for the single probability dataset approach to 0.146 for the multiple probabilities datasets approach, while the threshold of Blossom is calculated to be 0.142. Therefore, the goal of reaching equivalent decoding performance to Blossom was achieved. However, we should mention that the theoretical upper limit for the depolarizing error model with noiseless error syndrome measurements for the toric code is 0.189 [38]. Also, there exist various decoders that can reach close to the upper limit decoding performance for the surface code like the Maximum Likelihood Decoder (0.18) [34], the Markov Chain Monte Carlo (0.17) [39], the neural decoder described in [14] (0.165) and for the toric code the Renormalization Group decoder (0.164) [40].

B. Circuit noise model

For the circuit noise model, we used 5 training datasets that were sampled at the following physical error rates: $4.5\text{x}10^{-3}$, $1.5\text{x}10^{-3}$, $8.0\text{x}10^{-3}$, $4.5\text{x}10^{-4}$, $2.5\text{x}10^{-4}$. Since, imperfect error syndrome measurements are assumed the logical error rate is calculated per window of d error correction cycles.

As in the previous section, we observe from Figure 13 and Table V 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. In the case of d=7, we observe that for the physical error rates tested above the pseudo-threshold value, which is the sampling and training probability, the decoder does not perform efficiently, rather its performance is closer to the performance of d=5. However, below the pseudo-threshold value the samples included in the training dataset become relevant again and the performance resembles the performance of d=7.

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 in terms of pseudo-threshold are close to the results in the literature corresponding to the circuit noise model. The optimized version of Blossom as described in [23, 41] achieves $4.20\text{x}10^{-4}$, $2.10\text{x}10^{-3}$ and $3.2\text{x}10^{-3}$ for d=3, 5 and 7, respectively. Moreover, another neural network based decoder which is presented in [16], reports $3.18\text{x}10^{-4}$ and $7.11\text{x}10^{-4}$ for d=3 and d=5, respectively.

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

Decoder	d=3	d=5	d=7
Single prob. dataset	$3.99\text{x}10^{-4}$	$9.23\text{x}10^{-4}$	$1.41\text{x}10^{-3}$
Multiple prob. dataset	$4.44\text{x}10^{-4}$	$1.12\text{x}10^{-3}$	$1.66\text{x}10^{-3}$

The threshold of the rotated surface code for the circuit noise model has improved from $2.25\text{x}10^{-3}$ 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. The surface code threshold for the Blossom algorithm is ~$6\text{x}10^{-3}$ [23, 41].

V. Conclusions

This work focused on comparison between different decoding strategies that employ neural networks to decode the rotated surface code. Such kind of decoders are currently being in-

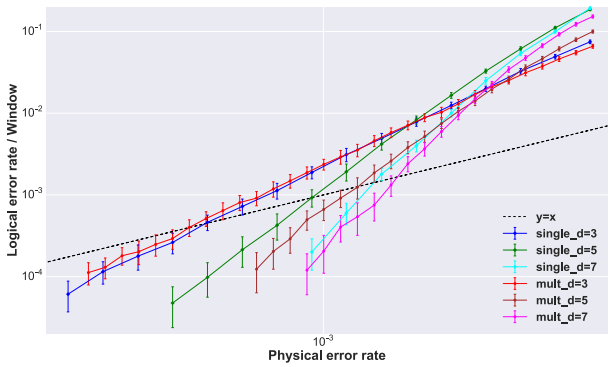

FIG. 13. Decoding performance comparison between the high level decoder trained on a single probability dataset and the high level decoder trained on multiple probabilities datasets for the circuit noise model with imperfect error syndrome measurements. Each point has a confidence interval of 99.9%

vestigated due to their good decoding performance and constant execution time.

We implemented two different ways that neural networks can be used to assist (high level decoder) or perform (low level decoder) the decoding and discussed how tuning of the neural network parameters will affect their decoding performance and decoding time (execution time).

We showed that the high level decoder can achieve better decoding performance when using the same dataset size and has a constant execution time with the physical error rate for a given code distance and it increases linearly with respect to the code distance. For the FFNN and the RNN, we observed that although FFNNs can have smaller execution time, RNNs can be trained easier, therefore successfully decode higher code distances and achieve better decoding performance. Based on these observations, we further analyzed the decoding performance of a high level decoder based on a RNN.

We showed that sampling and training based on multiple datasets at different physical error rates, can increase the decoding performance due to the higher relevance of the samples. As expected, equivalent decoding performance to Blossom was achieved for $d \leq 9$ for the depolarizing error model with noiseless syndrome measurements and successful decoding was achieved for $d \leq 7$ for the circuit noise model with noisy syndrome measurements.

We were not able to go to higher code distances due to the exponential increase of the dataset with the linear increase of the code distance, which poses an important challenge to such kind of decoders since much larger code distances need to be employed to have meaningful quantum computation and storage. A possible solution would be to use techniques based on distributed decoding.

VI. Acknowledgments

The authors would like to acknowledge funding from Intel Corporation.

References

[1] X Fu, L Riesebos, L Lao, CG Almudever, F Sebastiano, R Versluis, E Charbon, and K Bertels. A heterogeneous quantum computer architecture. In *Proceedings of the ACM International Conference on Computing Frontiers (CF)*, pages 323–330. ACM, 2016. doi: 10.1145/2903150.2906827.

[2] X Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50)*, pages 813–825. IEEE/ACM, 2017. doi: 10.1145/3123939.3123952.

[3] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. Ph.D. dissertation, Caltech Theory department, California Institute of Technology, Pasadena, California, USA, 1997.

[4] 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`.

[5] 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*, 3, 2017. doi: https://doi.org/10.1038/s41534-017-0039-x.

[6] 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`.

[7] 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.

[8] A. G. Fowler. Optimal complexity correction of correlated errors in the surface code. *arXiv:1310.0863*, 2013. URL https://arxiv.org/abs/1310.0863.

[9] A.G. Fowler. Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time. *arXiv:1307.1740v3*, 2014. URL https://arxiv.org/abs/1307.1740.

[10] Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv:1709.06218*, 2017. URL https://arxiv.org/abs/1709.06218.

[11] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2018. URL http://stacks.iop.org/2058-9565/3/i=1/a=015004.

[12] Paul Baireuther, Thomas E. O'Brien, Brian Tarasinski, and Carlo W. J. Beenakker. Machine-learning-assisted correction of correlated qubit errors in a topological code. *Quantum*, 2:48, January 2018. ISSN 2521-327X. doi:10.22331/q-2018-01-29-48. URL https://doi.org/10.22331/q-2018-01-29-48.

[13] Giacomo Torlai and Roger G. Melko. Neural decoder for topological codes. *Phys. Rev. Lett.*, 119:030501, 7 2017. doi:10.1103/PhysRevLett.119.030501. URL https://link.aps.org/doi/10.1103/PhysRevLett.119.030501.

[14] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific Reports*, 7(11003), 2017. doi:10.1038/s41598-017-11266-1.

[15] Nikolas P. Breuckmann and Xiaotong Ni. Scalable neural network decoders for higher dimensional quantum codes. *Quantum*, 2, May 2018.

[16] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3(4):044002, 2018. URL http://stacks.iop.org/2058-9565/3/i=4/a=044002.

[17] M. Maskara, A. Kubica, and T. Jochym-O'Connor. Advantages of versatile neural-network decoding for topological codes. *Physical Review A*, 99(5), 2019. doi: https://doi.org/10.1103/PhysRevA.99.052351.

[18] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *Physical Review Letters*, 122, 2019. doi: https://doi.org/10.1103/PhysRevA.99.052351.

[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. URL https://arxiv.org/abs/quant-ph/9811052.

[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. doi: https://doi.org/10.1103/PhysRevA.76.012305.

[22] Robert Raussendorf and Jim Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, 98:190504, May 2007. doi:10.1103/PhysRevLett.98.190504. URL https://link.aps.org/doi/10.1103/PhysRevLett.98.190504.

[23] Austin G. Fowler, Ashley M. Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Phys. Rev. A*, 80:052312, Nov 2009. doi:10.1103/PhysRevA.80.052312. URL https://link.aps.org/doi/10.1103/PhysRevA.80.052312.

[24] David S. Wang, Austin G. Fowler, and Lloyd C. L. Hollenberg. Surface code quantum computing with error rates over 1%. *Phys. Rev. A*, 83:020302, Feb 2011. doi:10.1103/PhysRevA.83.020302. URL https://link.aps.org/doi/10.1103/PhysRevA.83.020302.

[25] Austin G. Fowler, Adam C. Whiteside, and Lloyd C. L. Hollenberg. Towards practical classical processing for the surface code. *Phys. Rev. Lett.*, 108:180501, May 2012. doi:10.1103/PhysRevLett.108.180501. URL https://link.aps.org/doi/10.1103/PhysRevLett.108.180501.

[26] H Bombin and M A Martin-Delgado. Quantum measurements and gates by code deformation. *Journal of Physics A: Mathematical and Theoretical*, 42(9):095302, feb 2009. doi:10.1088/1751-8113/42/9/095302. URL https://doi.org/10.1088%2F1751-8113%2F42%2F9%2F095302.

[27] H Bombin. Clifford gates by code deformation. *New Journal of Physics*, 13 (4):043005, apr 2011. doi:10.1088/1367-2630/13/4/043005. URL https://doi.org/10.1088%2F1367-2630%2F13%2F4%2F043005.

[28] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012. doi:10.1103/PhysRevA.86.032324. URL https://link.aps.org/doi/10.1103/PhysRevA.86.032324.

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

[30] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012. doi:10.1088/1367-2630/14/12/123011. URL https://doi.org/10.1088%2F1367-2630%2F14%2F12%2F123011.

[31] 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*, 8:034021, Sep 2017. doi:10.1103/PhysRevApplied.8.034021. URL https://link.aps.org/doi/10.1103/PhysRevApplied.8.034021.

[32] Kao-Yueh Kuo and Chung-Chin Lu. On the hardness of decoding quantum stabilizer codes under the depolarizing channel. pages 208–211, 01 2012. ISBN 978-1-4673-2521-9.

[33] 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. doi:https://doi.org/10.1103/PhysRevA.80.052312.

[34] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Phys. Rev. A*, 90:032326, Sep 2014. doi:10.1103/PhysRevA.90.032326. URL https://link.aps.org/doi/10.1103/PhysRevA.90.032326.

[35] 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.

[36] Xiaotong Ni. Neural network decoders for large-distance 2d toric codes. *arXiv:1809.06640*, 2018. URL https://arxiv.org/abs/1809.06640.

[37] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Decoding surface code with a distributed neural network based decoder. *arXiv preprint arXiv:1901.10847*, 2019. URL https://arxiv.org/abs/1901.10847.

[38] H. Bombin, Ruben S. Andrist, Masayuki Ohzeki, Helmut G. Katzgraber, and M. A. Martin-Delgado. Strong resilience of topological codes to depolarization. *Phys. Rev. X*, 2:021004, Apr 2012. doi:10.1103/PhysRevX.2.021004. URL https://link.aps.org/doi/10.1103/PhysRevX.2.021004.

[39] James R. Wootton and Daniel Loss. High threshold error correction for the surface code. *Phys. Rev. Lett.*, 109:160503, Oct 2012. doi:10.1103/PhysRevLett.109.160503. URL https://link.aps.org/doi/10.1103/PhysRevLett.109.160503.

[40] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Phys. Rev. Lett.*, 104:050504, Feb 2010. doi:10.1103/PhysRevLett.104.050504. URL https://link.aps.org/doi/10.1103/PhysRevLett.104.050504.

[41] A. G. Fowler, A. C. Whiteside, A. L. McInnes, and A. Rabbani. Topological code autotune. *PHYSICAL REVIEW X*, 2:041003, 2012. doi:10.1103/PhysRevX.2.041003.