Quantum Machine Intelligence (2020) 2: 3
https://doi.org/10.1007/s42484-020-00015-9

RESEARCH ARTICLE

Decoding surface code with a distributed neural network–based decoder

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

Received: 12 June 2019 / Accepted: 26 February 2020 / Published online: 13 March 2020

Abstract

There has been a rise in decoding quantum error correction codes with neural network–based decoders, due to the good decoding performance achieved and adaptability to any noise model. However, the main challenge is scalability to larger code distances due to an exponential increase of the error syndrome space. Note that successfully decoding the surface code under realistic noise assumptions will limit the size of the code to less than 100 qubits with current neural network–based decoders. Such a problem can be tackled by a distributed way of decoding, similar to the renormalization group (RG) decoders. In this paper, we introduce a decoding algorithm that combines the concept of RG decoding and neural network–based decoders. We tested the decoding performance under depolarizing noise with noiseless error syndrome measurements for the rotated surface code and compared against the blossom algorithm and a neural network–based decoder. We show that a similar level of decoding performance can be achieved between all tested decoders while providing a solution to the scalability issues of neural network–based decoders.

Keywords Quantum error correction · Quantum error detection · Surface code · Decoding · Artificial neural networks

1 Introduction

Quantum error correction (QEC) is for now considered to be the most time- and resource-consuming procedure in quantum computation. However, the way that quantum computing is currently envisioned, QEC is necessary for reliable quantum computation and storage. The need for QEC arises from the unavoidable coupling of the quantum system with the environment, which causes the qubit state to be altered (decohere). Altering the quantum state is perceived as errors generated in the quantum system. Through active error correction and fault-tolerant mechanisms that control error propagation and keep the error rates low, we can have the error-free desired state. Note that, in *fault-tolerant* techniques, errors can occur in the quantum system, but do not affect the quantum state in a catastrophic manner (Nielsen and Chuang 2002).

A critical sub-routine of QEC is *decoding*. Decoding involves the process of identifying the errors that occur in the quantum system and proposing corrections that keep the quantum state error-free. The importance of high-speed and accurate decoding lies in the fact that the time budget allowed for error correction is small, since qubits lose their state rapidly. Therefore, if the process of decoding exceeds the error correction time budget, errors will accumulate to the point that the error-free state cannot be retrieved.

Various classical decoding algorithms have been proposed over the years with a few examples of classical decoding algorithms being the Blossom algorithm (Edmonds 1965; Kolmogorov 2009; Fowler 2015; Fowler 2013), the maximum-likelihood algorithm (Bravyi et al. 2014), and the renormalization group (RG) algorithm (Duclos-Cianci and Poulin 2010a; Duclos-Cianci and Poulin 2010b). Recently, there is an increase in the development of *neural network–based decoders* that either consist exclusively of neural networks (Torlai and Melko 2017; Krastanov and Jiang 2017) or a classical module working together with neural networks (Varsamopoulos et al. 2017; Baireuther et al. 2018; Chamberland and Ronagh 2018; Ni 2018; Davaasuren et al. 2018). Neural network–based decoders exist with different designs in the way the decoding is performed, and a variety of types of neural networks has been explored, like feed-forward, recurrent, and convolutional neural networks.

✉ Savvas Varsamopoulos
svarsamo@gmail.com

[1] Quantum Computer Architecture Lab, Delft University of Technology, Delft, The Netherlands

[2] QuTech, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands

In Fig. 1, we present an abstract comparison between various decoding algorithms based on their decoding performance (accuracy) and their execution time (wall clock time), namely, the Markov Chain Monte Carlo (MCMC) (Hutter et al. 2014), the maximum-likelihood decoder (MLD) (Bravyi et al. 2014), the minimum weight perfect matching (MWPM) (Edmonds 1965; Kolmogorov 2009) that Blossom algorithm is based on, the neural network–based decoder (NNbD) (Varsamopoulos et al. 2019), the renormalization group (RG) (Duclos-Cianci and Poulin 2010a), and the cellular automaton (CA) (Herold et al. 2015). Decoding performance is typically calculated as the ratio of the number of logical errors created out of the decoder corrections over the number of error correction cycles run to accumulate these errors. Execution time is defined as the time spent from the moment that the input data arrive at the decoder until the time that the decoder proposes the corrections. As can be seen from Fig. 1, neural network–based decoders can reach equivalent decoding performance as classical algorithms while requiring smaller execution time. This is the main reason that neural network–based decoders are explored and various designs have been proposed recently. However, the main issue with such decoders is that scaling to larger quantum systems will be significantly harder compared with classical decoders, due to the required training process of the neural network. As the size of the system increases, more training samples need to be collected and then the neural network has to be trained based on them. The main challenge of NNbDs is that in order to reach similar decoding performance to classical algorithms as the quantum system is increasing, the amount of samples required to be collected increases in an exponential way, which makes the training harder and slower.

In this work, we will present a neural network–based decoder that performs decoding in a distributed fashion, therefore providing a solution for the issue of decoding large codes. We should mention that there exist classical algorithms that perform decoding in a distributed way, as can be found in (Duclos-Cianci and Poulin 2010a) and (Fowler 2015), but in this paper, we will provide a different approach of the distributed decoding concept. In (Duclos-Cianci and Poulin 2010a), the original idea of RG decoding approach is described and tested. The RG decoding is based on the division of the code into small tiles, in which a given number of physical qubits are included and error probabilities about the physical qubits inside all tiles are calculated. Then, these tiles are grouped into larger tiles and the error probabilities about the qubits are updated. This procedure is continued until only a single tile has remained containing all the physical qubits of the system. Based on the updated error probabilities of the largest tile, the decoder can propose a set of corrections. In (Fowler 2015), a distributed decoding approach is described, where the code is divided into small tiles. However, in this case, blossom algorithm is used to decode each tile, and based on the result of it and the neighbouring information between the tiles, the decoder can propose corrections for the whole code. Each tile is monitored by an application-specific integrated circuit (ASIC), which is dedicated for the tile.

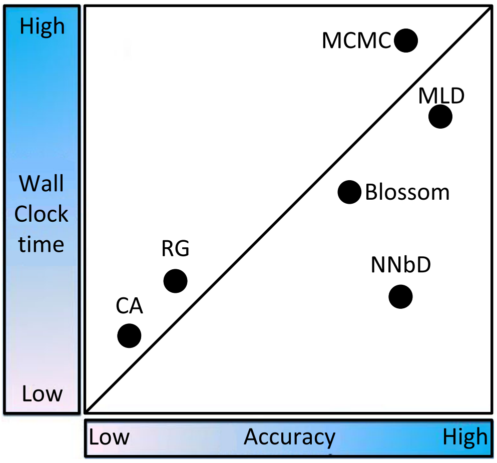

Fig. 1 Abstract comparison between decoding performance and execution time of various decoding algorithms

In our strategy, the code is divided into small overlapping regions, referred to as *overlapping tiles*, where local information about errors on physical qubits is obtained. Then, this local information is combined and a decoding for the whole code is obtained. We compare our algorithm to the unoptimized version of the blossom algorithm (Edmonds 1965; Kolmogorov 2009) and argue about the decoding performance achieved. Furthermore, we will provide reasoning for the potential high level of parallelization of our algorithm that will be suitable for a high-speed hardware implementation without loss of decoding performance. Also, the problem of the exponential increase of the error syndrome space is mitigated, since it is controlled by the selection of the size of the decoded regions. This allows neural network–based decoders to successfully decode larger codes.

The rest of the paper is organized in the following way: in Sections 2 and 3, we give a short introduction in quantum error correction and the concept of the RG decoding, respectively. In Section 4, we present the design of the distributed neural network–based decoder, and in Section 5, we provide the results in terms of decoding performance. Finally, in Section 6, we draw our conclusions about the distributed decoding approach.

2 Quantum error correction

Quantum computation is error-prone due to the fragility of the qubits, which lose their coherence through their interaction with the environment. Furthermore, quantum operations are still imperfect, altering the quantum state in unpredictable ways. These alterations are interpreted as errors in the quantum system, which are discretized into Pauli errors in order to be corrected in an easier way.

Quantum error correction involves an encoding process of the quantum information into multiple qubits and a decoding process that identifies and counteracts the noise that is inserted in the quantum system. Many unreliable *physical qubits* are encoded, similarly to classical error correction, to one more reliable qubit, known as *logical qubit*. There are many ways that encoding can be achieved; these encoding schemes are also known as quantum error correcting codes (Landahl et al. 2011; Fowler et al. 2012a; Suchara et al. 2011; Bravyi 2010; Bombin 2010; Bravyi et al. 2013), but we are focusing on the *surface code* (Gottesman 1997; Kitaev 2003).

Logical qubits are used both for quantum computation and memory; however, errors occur at the physical level. Therefore, a decoding process that will identify the errors on the physical qubits is required. At the end of the decoding process, corrections against identified errors are proposed by the decoder.

2.1 Surface code

The surface code is a topological stabilizer code with simple structure, local interactions, and high level of protection against errors (Dennis et al. 2002; Raussendorf and Harrington 2007; Fowler et al. 2009; Wang et al. 2011; Fowler et al. 2012a; Bombin and Martin-Delgado 2009; Bombin 2011; Bravyi and Kitaev 1998; Fowler et al. 2012b). A logical qubit in the surface code includes two types of physical qubits, namely, the *data qubits*, which store quantum information, and the ancillary or *ancilla qubits*, which can be used to find errors on the data qubits. The smallest version of a planar surface code (Bravyi and Kitaev 1998; Freedman and Meyer 2001), which requires the least amount of physical qubits, known as the *rotated surface code* (Horsman et al. 2012), is presented in Fig. 2.

A logical qubit is defined by its *logical operators* (X_L, Z_L), which are responsible for logical state changes. Any operator of the form X^n or Z^n that forms a chain which spans two boundaries of the same type can be considered a logical operator, with n being the amount of data qubits included in the logical operator. The operator with the smallest n is always selected; however, as can be seen from Fig. 2, there are multiple logical operators with $n = 3$, which is the smallest n for this code. Any one of them can be selected without further assumptions. For example, a valid X_L could be $X_0X_3X_6$ and a valid Z_L could be $Z_6Z_7Z_8$.

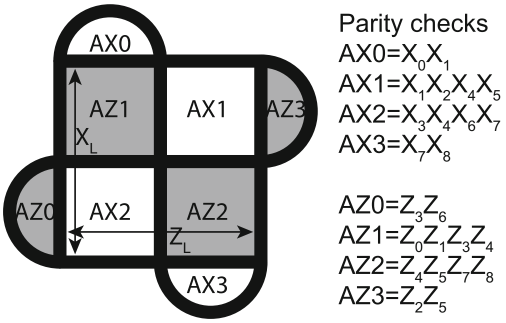

Fig. 2 Rotated surface code describing 1 logical qubit that consists of 17 physical qubits. The 9 qubits that are at the corners of the tiles (0–8) are data qubits, and the 8 qubits that are inside the tiles (A*X*i, A*Z*i) are ancilla qubits. The parity checks of the code are shown on the right side

The level of protection against errors is usually described with the metric known as *code distance*. Code distance, (d), is calculated as the minimum number of physical operations required to change the state of the logical qubit (Terhal 2015; Devitt et al. 2013). Therefore, for the logical qubit of Fig. 2, the code distance would be 3.

The relation between the code distance and the errors that can be successfully corrected is given by:

$$\text{weight of error} = \left\lfloor \frac{d-1}{2} \right\rfloor \tag{1}$$

According to Eq. 1, for a $d = 3$ surface code, all single errors (weight = 1) are going to be successfully corrected.

Since the errors are discretized into bit- and phase-flip errors, it is sufficient to only have two types of ancilla qubits, a Z-type for detecting bit-flips and an X-type for detecting phase-flips. Each ancilla qubit resides inside a tile and interacts with 4/2 neighbouring data qubits to perform a parity check operation. We provide the parity checks for a $d = 3$ rotated surface code in Fig. 2, as obtained by running the circuits depicted in Fig. 3. These circuits are run in parallel and constitute a surface code (error correction) cycle. Both circuits consist of initialization of the ancilla qubit, followed by a series of CNOT gates between the ancilla and the data qubits, followed by ancilla measurement.

The result of the ancilla measurement is a binary value that indicates whether the value of the parity check measured is the same as the one of the previous error correction cycle or not. When a parity check returns a different value between two consecutive surface code cycles, it is referred to as a *detection event*. By running the circuits of Fig. 3, we obtain the values for all parity checks and infer what errors have occurred. Gathering all parity check values out of a single surface code cycle forms the *error syndrome*.

2.2 Error decoding

A single data qubit error will cause two neighbouring parity checks to indicate two detection events (Z error in the bottom of the lattice in Fig. 4), unless the error occurs at the corner of the lattice which will lead to only one parity check indicating one detection event (Z error in the top corner of the lattice in Fig. 4). Multiple data qubit errors that occur near each other form chains of errors (X errors in Fig. 4), which causes only

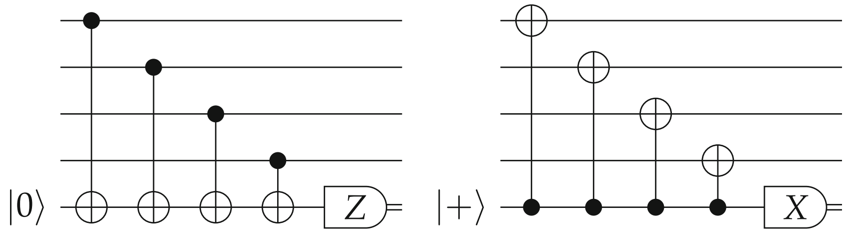

Fig. 3 Left: circuit for Z-type ancilla. Right: circuit for X-type ancilla

two detection events located at the parity checks existing at the endpoints of the error chain (Dennis et al. 2002; Terhal 2015; Fowler et al. 2012a, b).

In addition, the measurement process is also imperfect, which leads to different type of errors. When a measurement outcome is misinterpreted, a correction might be applied where no error existed and vice versa. The way that a measurement error is observed is by comparing the measurement values of multiple consecutive surface code cycles for the same parity check, as presented in Fig. 5.

In the case where the error probability for a data qubit error is equal to the error probability for a measurement error, d surface code cycles are deemed enough to successfully identify measurement errors (Raussendorf et al. 2007). When a measurement error is successfully identified, no correction is required.

Thus, through observation of the parity checks throughout multiple surface code cycles, identification of errors is made in space (data errors) and in time (measurement errors). The decoder, which is the module responsible for analysing the detection events and producing corrections against the errors that have occurred, receives the error syndrome out of one or multiple surface code cycles and produces a set of corrections to be applied.

However, totally suppressing the noise is unfeasible, since the decoder might misinterpret the information coming from the error syndrome. The main reason for such misinterpretations comes from the fact that the surface code is a degenerate code. This degeneracy means that different sets of errors create the same error syndrome. Therefore, based on the physical error rate of the quantum operations, different sets of errors are more likely than others. This puts an extra assumption to the decoder, since it should output different corrections based on the error probability. Based on all these reasons, it is evident that no decoder can perfectly suppress all noise.

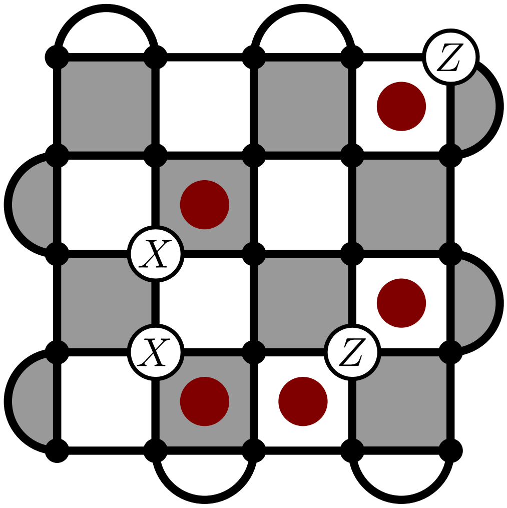

Fig. 4 Rotated surface code with code distance 5. Errors are denoted on top of the data qubits with X or Z, and detection events corresponding to these errors are shown with red dots

3 Decoding algorithms

The main parameters that define a good decoder are the decoding performance, the ability to efficiently scale to large code distances, and the execution time. There exist decoders that can reach good decoding performance, enough to make fault-tolerant quantum computing possible. Some of the classical algorithms are the *maximum-likelihood algorithm* (Bravyi et al. 2014), the *blossom algorithm* (Edmonds 1965; Kolmogorov 2009; Fowler 2015), and the *renormalization group* (RG) algorithm (Duclos-Cianci and Poulin 2010a; Duclos-Cianci and Poulin 2010b). The maximum-likelihood algorithm investigates the most probable error that has occurred that produces the observed error syndrome. This process can reach high decoding accuracy but is extremely time-consuming especially as the code distance increases. The execution time scales as $O(n\chi^3)$, with χ being an approximation parameter, as given in Bravyi et al. (2014). The blossom algorithm can reach slightly lower decoding performance than the maximum-likelihood decoder, but still good enough to be used in experiments. The execution time scales linearly with the number of qubits (Fowler 2013), but still

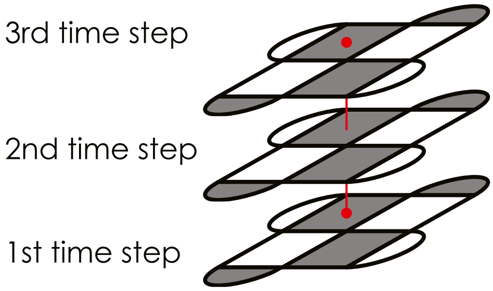

Fig. 5 Rotated surface code with code distance 3 at consecutive time steps. Alternating pattern on the measurement value of the same parity check indicates the presence of a measurement error

might not meet the small execution time requirements of contemporary experiments. However, there exists an optimized version of the blossom algorithm that claims a constant average processing time per detection round, which requires dedicated hardware (Fowler 2015). The renormalization group decoding provides a good solution for the decoding of large quantum systems, because decoding is performed in a local manner through distributed regions throughout the lattice. The RG algorithm can be highly parallelized, and the scaling is reported to be log(l), for an $l \times l$ code (Duclos-Cianci and Poulin 2010a). However, the decoding accuracy is not as good as the other two algorithms. Neural network–based decoders with a large variety of designs (Torlai and Melko 2017; Krastanov and Jiang 2017; Varsamopoulos et al. 2017; Baireuther et al. 2018; Chamberland and Ronagh 2018; Ni 2018; Davaasuren et al. 2018; Maskara et al. 2018; Darmawan and Poulin 2018; Sweke et al. 2018; Varsamopoulos et al. 2019) have been recently suggested that report similar or better decoding performance than the blossom and RG decoders, making them a potential candidate for decoding.

Currently, the time budget for error correction and decoding is small for most qubit technologies, due to the erroneous nature of the qubits and the imperfect application of quantum operations. Therefore, a high-speed version of a decoder would be necessary. This requirement leads us to neural network–based decoders which are shown to have constant execution time after being trained. However, in order to run complex algorithms, many qubits are required, and as mentioned earlier, scaling to large code distances with neural network–based decoders is extremely hard, since the amount of data required to train the algorithm grow exponentially with the number of qubits.

In this paper, we will present a neural network–based decoder that exploits the concept of distributed decoding, in a similar way to RG decoding and the parallel approach of (Fowler 2015). Based on such a distributed way of decoding, we limit the amount of training data required, making the distance of the code irrelevant.

Our previous efforts were mainly focused on developing neural network–based decoders that can achieve better decoding performance than classical decoding algorithms and report a constant execution time for each code distance for all range of physical error probabilities, which scales linearly with the code distance (Varsamopoulos et al. 2019). However, good decoding performance was harder to achieve as the code distance increased. The main problem was the exponential increase of the error syndrome space, which required an immensely large number of training samples in order for the decoder to achieve similar performance to the classical decoding algorithms for $d > 9$. We provide the size of the training datasets used for the code distances investigated in (Varsamopoulos et al. 2019) for the depolarizing error model in Table 1.

Table 1 Size of training datasets

Code distance	Training dataset size	Full dataset size
$d = 3$	256	2^8
$d = 5$	6×10^5	2^{24}
$d = 7$	5×10^6	2^{48}
$d = 9$	2×10^7	2^{80}

A way that the error space can be limited is through a distributed way of decoding similar to the RG algorithm. By dividing the code in small regions which are going to provide individual information about decoding every region of the code, the decoder can have enough information about decoding the whole code. Limiting the region that we want to locally decode, the error syndrome space is also limited, allowing us to increase the distance of the code without changing the decoding of each region.

RG decoding is similar to decoding concatenated codes, which have various levels of encoding, as can be seen in Fig. 6.

In these codes, decoding is achieved by passing the error information concerning the qubits from the lower level to the higher level. The information about errors is updated throughout the encoding levels. The decoding occurs at the last encoding level, and a final decision about the logical state is made.

The strategy of the RG decoding can be described according to Fig. 7. At first, the lattice is cut in small (green) tiles and the probability of an error occurring in all qubits included in that tile is evaluated. After gathering the updated error probabilities in the green tiles, the lattice is cut into bigger (red) tiles and the error probability of all qubits included in that tile is evaluated. This process is continued until there is only one tile left that includes all qubits in the code.

The same approach can be applied to surface code. However, the challenge here is that the parity checks cannot be broken down into constant size tiles in a way that every parity check corresponds to a single tile. Therefore, we need to use overlapping tiles, which will always include whole parity checks of the code in a single tile. The boundary qubits that belong to neighbouring tiles are treated as independent variables on each tile, and the error probability for the same qubit is different depending on the tile. The way that the error probabilities are usually calculated is by belief propagation (Duclos-Cianci and Poulin 2010a; Duclos-Cianci and Poulin 2010b) in the RG approach.

We decided to use the idea of overlapping tiles, but follow a different approach than the RG algorithm as we will explain in the following section.

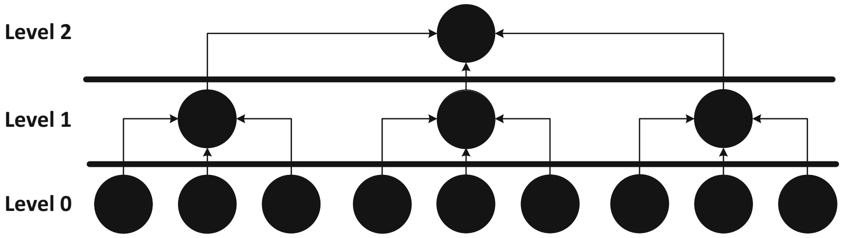

Fig. 6 Encoding levels of a concatenated code. At level 0, there are nine qubits that are encoded in three qubits at level 1 and these qubits are encoded in one qubit at level 2. Arrows show the information flow

4 Proposed algorithm based on distributed decoding

We developed a neural network–based decoder that performs distributed decoding based on the concept of the RG decoders. As mentioned, the main idea behind this algorithm is to make neural network–based decoders able to successfully decode large code distances. By restricting the decoding in small regions (tiles) of the lattice, the decoder does not have to explore a large error syndrome space, rather just decode every small tile and then combine the information out of all tiles.

The main difference between a distributed neural network–based decoder and the RG decoder is that the former only has one level of concatenation. Instead of moving from smaller tile to bigger tile until the whole lattice is a single tile, we segment the lattice into small equally sized tiles that are overlapping with each other, so that each tile includes whole parity checks of the code. Then, we obtain error information from each individual tile and combine the information out of all tiles to get the error information for the whole lattice. In this case, there is no need to calculate the error probability of all qubits and forward it to the next level of concatenation, rather find a way to combine the information arising from the each tile.

In order to decode based on the distributed decoding approach, we will use the same two-module decoder as was presented in (Varsamopoulos et al. 2019). Our decoding algorithm consists of two modules, a classical decoding module that we call *simple decoder* and a neural network. The simple decoder provides a naive decoding for the whole lattice, in which a chain is created between each detection event and its closest boundary of the same type. The corrections arising from the simple decoder occur in the data qubits underneath the chain. An example is provided in Fig. 8, where AZ5 and ancilla AX4 have indicated the presence of an error in their proximity. The proposed corrections of the simple decoder will be $Z5$, $Z11$ arising from ancilla AX4 and $X3$, $X7$ arising from ancilla AZ5.

The simple decoder receives the error syndrome for the whole lattice and provides a set of corrections for the whole lattice. This is a fast process since the corrections arising from each detection event are independent from the corrections arising from other detection events; therefore, they can be parallelized. However, the simple decoder cannot yield high decoding accuracy on its own, due to its simplistic design.

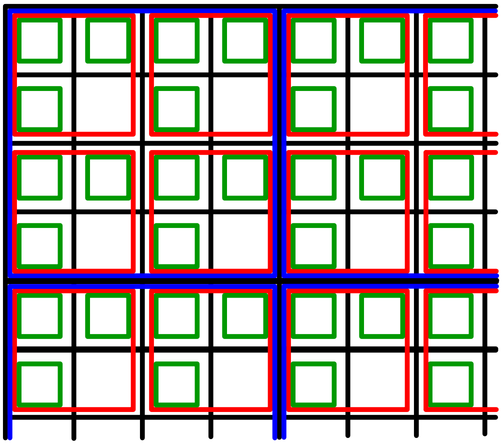

Fig. 7 Tile segmentation that represents the levels of concatenation in a concatenated code. The smallest level of concatenation is represented by the green tiles, the next level of concatenation is represented by the red tiles, the following level of concatenation is represented by the blue tiles, etc.

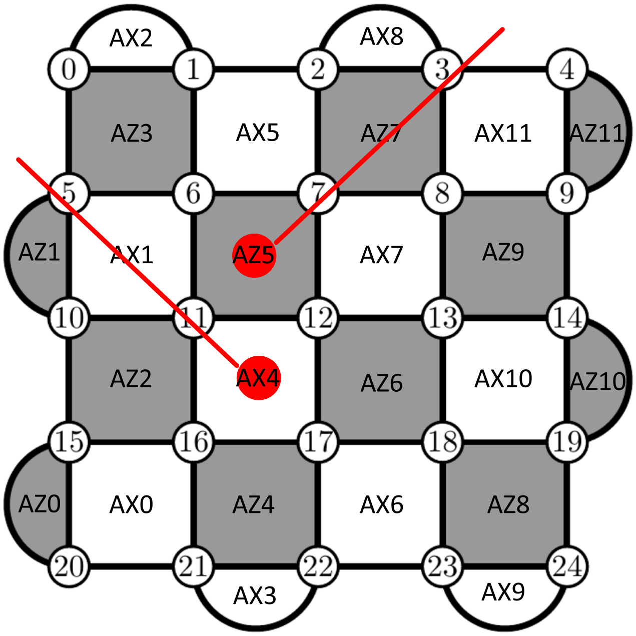

Fig. 8 Description of the simple decoder operation for the rotated surface code with distance 5. Detection events are presented with the red dots. Red lines indicate which data qubits are going to be corrected

That is why we also include the neural network that will work as a supervisor to the simple decoder. More accurately, the neural network will be trained to identify for which error syndromes the simple decoder will lead to a logical error. In the case where a logical error will be created out of the simple decoder corrections, the neural network will output the appropriate logical operator that will cancel the logical error out. As we showed in (Varsamopoulos et al. 2019), the combination of these two modules will provide high decoding performance.

In order to train the neural network, we create a training dataset by running surface code cycles and storing the error syndrome and the corresponding logical state of the logical qubit after the corrections of the simple decoder are applied. The size of the training dataset varies based on the code distance and the error model. For more information about all the parameters that affect the dataset, we refer the reader to our previous work (Varsamopoulos et al. 2019).

In Fig. 9, we provide an example of the segmentation of a $d = 5$ rotated surface code into four overlapping tiles of $d = 3$ rotated surface codes.

As can be seen from Fig. 9, each parity check is included in at most two tiles. The error syndrome obtained for the whole lattice ($d = 5$) is broken down into parts of the error syndrome that correspond to each small tile ($d = 3$). The error syndrome out of one surface code cycle consists of 24 bits, due to the 24 parity checks of the $d = 5$ code. The error syndrome will be cut into smaller parts of the initial error syndrome that fit the $d = 3$ tiles. Due to inclusion of the shared parity checks, the bits that are available out of the four $d = 3$ tiles are now 32. Each error syndrome of the $d = 3$ tile corresponds to a part of the complete error syndrome. The error probabilities of the logical state, Prob(I), Prob(X), Prob(Z), Prob(Y), that are associated with the given tile are averaged, and the probabilities for the logical state of each tile is provided. Then, the 4 probabilities concerning the logical state of each $d = 3$ tile are used as the inputs of the neural network, which will provide at the output the probabilities of the logical state for the whole lattice. Based on the output of the neural network, extra corrections are going to be applied in the form of the appropriate logical operator to cancel any potential logical error created by the simple decoder. The information contained in the 32 bits of the $d = 3$ tiles is now compressed to 16 bits that constitute the inputs of the neural network and represent the probabilities of contribution to the logical state out of every $d = 3$ tile. The results of the proposed algorithm which is based on the distributed decoding are presented in Section 5.1.

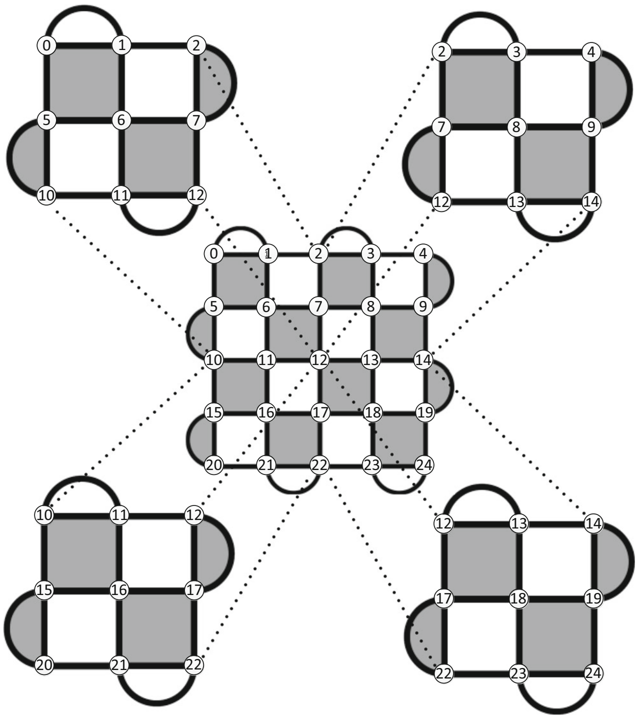

Fig. 9 Segmentation of a $d = 5$ rotated surface code into four overlapping tiles of $d = 3$ rotated surface codes

4.1 Optimizing for the size of training dataset

The scalability problem that all neural network–based decoders face is based on the exponential increase of the training samples required to efficiently decode. As an extension to our work on neural network–based decoders, we propose an alteration to our decoding algorithm in order to increase the important training samples included in the training dataset, without increasing the size of the dataset.

As mentioned, our decoding strategy is based on a two-module approach (simple decoder and neural network), where the neural network exists to increase the decoding performance of the simple decoder. However, the simple decoder can be designed in different ways, which will lead to different decoding performance for different designs. Therefore, an investigation of the performance of the simple decoder is crucial before the training of the neural network.

We observed that for all code distances investigated for the depolarizing error model, the simple decoder provided corrections that would lead to an error-free logical state (I) $\sim 42\%$ of the time. In those cases, the neural network would be unnecessary, since it would output the identity operator. Therefore, if we removed the error syndromes that the simple decoder corrects properly from the training dataset, then the dataset could be increased even further, with more relevant error syndromes. The only caveat is that another module, named binary neural network in Fig. 13, should be included to the decoder which will predict whether the obtained error syndrome will be properly corrected by the simple decoder or not. The binary logic neural network might be implemented in a simpler way, which will make the

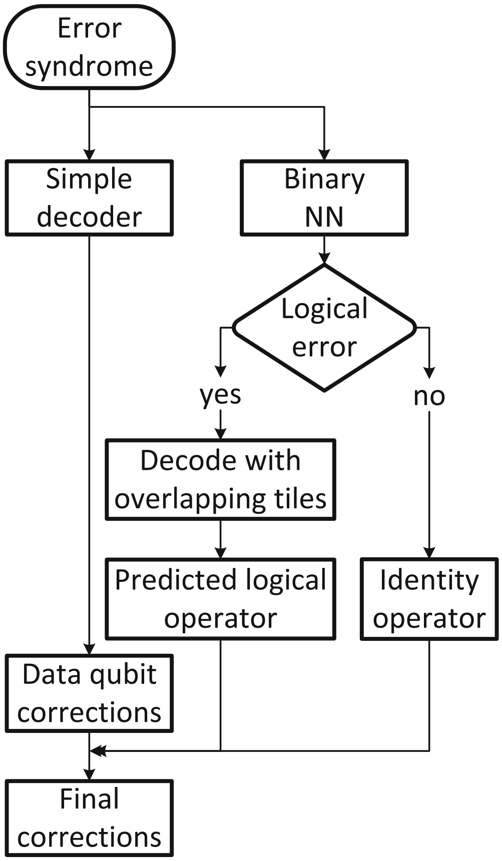

Fig. 10 Description of the design flow of the optimized version of the distributed decoder

binary classification task faster, instead of using a recurrent neural network as was chosen for this design.

A flowchart of the optimized algorithm with the inclusion of the extra neural network is presented in Fig. 10. We divide the operation of the neural network from the original design of distributed decoding to two neural networks, namely, a binary neural network and a neural network for distributed decoding.

The binary neural network will predict whether the obtained error syndrome will lead to a logical error or not. The input of the binary neural network is the obtained error syndrome for the whole lattice, and the output will be a binary value, indicating whether extra corrections need to be applied or not. These extra corrections will arise from the neural network for distributed decoding. This neural network will work similarly to the one in the original unoptimized strategy described in Section 4, but the training samples will be restricted to the error syndromes that lead to a logical error. The inputs and outputs of this neural network are previously explained. Note that we need to include all 4 logical states for this neural network, because there is still a probability of an unknown to training input to produce an error-free logical state. The results of the optimized version for the size of the dataset are presented in Section 5.2.

Table 2 Reduction in required inputs of the neural network

Code distance	Old inputs	New inputs
$d = 5$	24	16
$d = 7$	48	36
$d = 9$	80	64

5 Results

In order to check whether the distributed decoding algorithm can reach similar decoding performance as the other popular decoding algorithms, we tested it against an unoptimized version of the blossom algorithm (Edmonds 1965; Kolmogorov 2009) and our previous implementation of neural network–based decoder (Varsamopoulos et al. 2019) for the depolarizing error model with noiseless error syndrome measurements.

The depolarizing error model assumes errors only on the data qubits and perfect error syndrome measurements. Bit-flip (X) errors, phase-flip (Z) errors, and both bit- and phase-flip (Y) errors are assumed to be generated with equal probability of p/3. Such a simplistic error model is enough to prove that the distributed decoding algorithm that we propose can reach similar decoding performance to other decoding algorithms and that the scalability issues of neural network–based decoder are addressed.

The critical aspect of our decoder is the choice of the size of the overlapping tiles. Since there is only one level of concatenation, contrary to RG decoding, the size of the overlapping tiles plays a significant role in the algorithm. Having a large tile size might provide better decoding, for example, decoding a $d = 9$ surface code with $d = 7$ tiles might be more beneficial than decoding with $d = 3$ tiles, since there will be less shared parity checks and long error chains will be included in a single tile. However, the bottleneck that will make such a case decode poorly in our design is the inability of the decoder to handle properly the error syndromes unknown to the training dataset. Since it becomes exponentially harder to gather all the possible error syndromes as the code distance increases, the training dataset will be an incomplete set of all potential cases. In the case of an unknown to the training error syndrome, the neural network will not have any meaningful data to make a prediction, making the behaviour of the neural network inconsistent. Such a case occurs because there is an intermediate step between the cutting of the error syndrome into parts and the averaging of the probabilities of each part.

Based on that, we opted to always divide the lattice into $d = 3$ overlapping tiles, since the $d = 3$ case only consists of 256 different error syndromes. This is an easily obtained complete training dataset, to which any part of error syndrome of any large distance can deconstruct to. All possible error

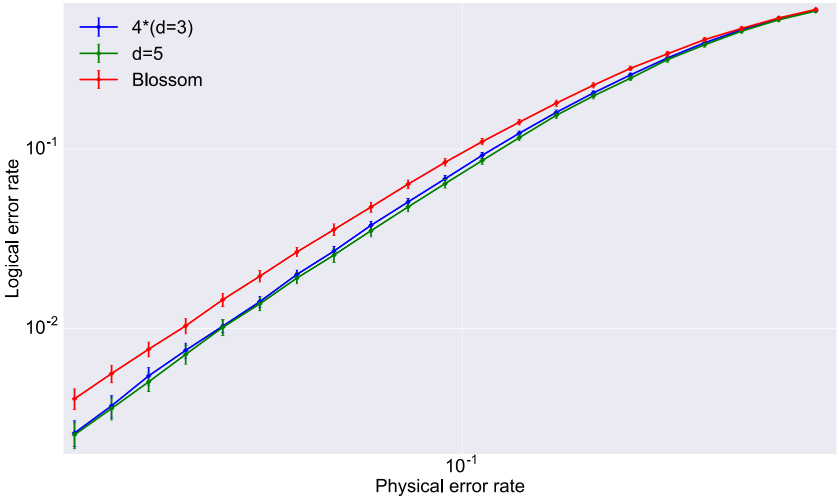

Fig. 11 Comparison of decoding performance between the distributed decoder with *four* overlapping tiles of $d = 3$ rotated surface codes inside a $d = 5$ rotated surface code (blue), the unoptimized version of the blossom algorithm (red), and the neural network–based decoder (green)

syndromes of the large lattice ($d > 3$) are represented through the $d = 3$ overlapping tiles, without having to explicitly sample all possible error syndromes for the large lattice.

The only downside of using $d = 3$ tiles is that there exist some error syndromes that are highly ambiguous to what logical state they lead. Fortunately, these ambiguous error syndromes are not extremely frequent, making the errors arising from this shortcoming rare.

Another benefit of the distributed decoding approach is that the number of inputs required by the neural network is decreased compared with decoding the whole lattice approach. The reduction of inputs of the neural network for the code distances tested are shown in Table 2.

5.1 Distributed decoding results

The comparison of the decoding performance between the distributed decoding, the neural network–based decoder from Varsamopoulos et al. (2019), and unoptimized version of the blossom algorithm for distance 5, 7, and 9 rotated surface codes is presented in Figs. 11, 12 and 13, respectively. Each point in these graphs has a confidence interval of 99.9% (Figs. 11, 12 and 13).

As can be seen from Figs. 11, 12, and 13, the distributed decoder can reach similar decoding performance to the compared decoders for $d = 5$, 7, and 9, respectively. In order to have a fair comparison between the two neural network–based decoders, we used the same dataset to train both decoders; therefore, the decoding performance should be comparable. These comparisons were used as a proof-of-concept to verify that a distributed decoding approach is feasible and that limitations are observed.

5.2 Optimized version results

The comparison of the decoding performance of this optimized version of the algorithm with the unoptimized

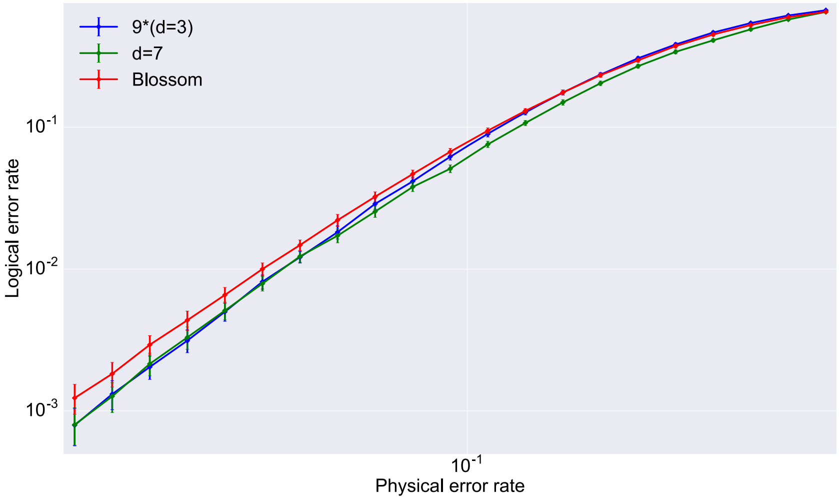

Fig. 12 Comparison of decoding performance between the distributed decoder with *nine* overlapping tiles of $d = 3$ rotated surface codes inside a $d = 7$ rotated surface code (blue), the unoptimized version of the blossom algorithm (red), and the neural network–based decoder (green)

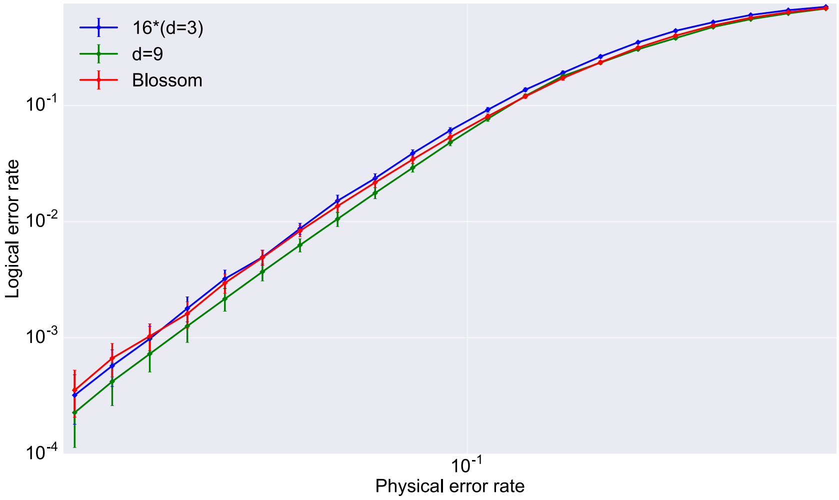

Fig. 13 Comparison of decoding performance between the distributed decoder with *sixteen* overlapping tiles of $d = 3$ rotated surface codes inside a $d = 9$ rotated surface code (blue), the unoptimized version of the blossom algorithm (red), and the neural network–based decoder (green)

one and the benchmarks that were used in this work for the largest code tested ($d = 9$) is presented in Fig. 14.

As expected, the optimized version with the two neural networks cannot achieve better decoding performance than the unoptimized version, since we kept the same training dataset for both designs in order to have a fair comparison. The binary neural network has the same dataset as the unoptimized version, but the neural network for distributed decoding only includes the ~ 58% of error syndromes that lead to a logical error.

An important clarification is that the optimization is mentioned in the context of the potential increase of the training dataset and not in terms of better decoding performance. However, the fact that we reached the same level of decoding performance with both designs, suggests that we can make these optimizations without any loss of decoding performance.

6 Conclusions

We presented a decoding algorithm that performs decoding in a distributed manner that can achieve similar decoding performance to existing decoders, like the blossom decoder and the neural network–based decoder for $d = 5$, 7, and 9. Furthermore, due to the distributed way of decoding and the deduction in the neural network inputs, larger codes can be potentially decoded. The problem of the exponential increase of the training dataset is mitigated through the distributed decoding strategy, where any error syndrome can be decomposed to smaller $d = 3$ tiles. However, large quantum systems will still require large amounts of training samples. Moreover, in terms of execution time, we assume that a highly parallel implementation for both the simple decoder and the neural network can potentially achieve a high-speed implementation of the algorithm. Finally, we provide an alternative version of the distributed decoding strategy that can

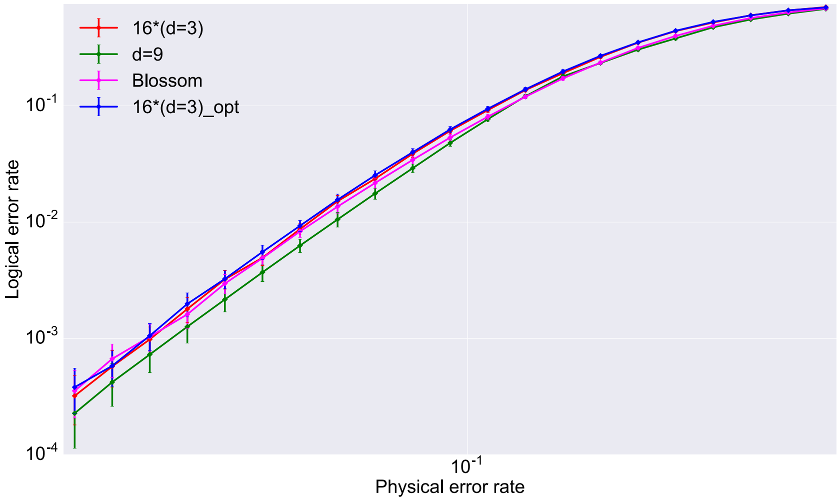

Fig. 14 Comparison between the optimized version of the distributed decoding (blue) to the unoptimized version (red), the unoptimized version of the blossom algorithm (pink), and the neural network–based decoder (green)

reach the same level of decoding performance as the original algorithm. The advantage of this alternative is the capability of using larger training datasets compared with other neural network–based decoders, making it easier to achieve better decoding performance for higher code distances.

References

Baireuther P, O'Brien TE, Tarasinski B, Beenakker CWJ (2018) Machine-learning-assisted correction of correlated qubit errors in a topological code. Quantum 2:48. https://doi.org/10.22331/q-2018-01-29-48

Bombin H (2010) Topological subsystem codes. Phys Rev A 81:032301. https://doi.org/10.1103/PhysRevA.81.032301

Bombin H (2011) Clifford gates by code deformation. New J Phys 13(4): 043005. https://doi.org/10.1088/1367-2630/13/4/043005

Bombin H, Martin-Delgado MA (2009) Quantum measurements and gates by code deformation. J Phys A Math Theor 42(9):095302. https://doi.org/10.1088/1751-8113/42/9/095302

Bravyi S (2010) Stabilizer subsystem codes with spatially local generators. IEEE Information Theory Workshop, p 1–5. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5592872&isnumber=5592637. Accessed 3 Mar 2020

Bravyi SB, Kitaev AY (1998) Quantum codes on a lattice with boundary. quant-ph/9811052

Bravyi S, Duclos-Cianci G, Poulin D, Suchara M (2013) Subsystem surface codes with three-qubit check operators. Quantum Inf Comput 13(11–12):963–985. http://dl.acm.org/citation.cfm?id=2535639.2535643. Accessed 3 Mar 2020

Bravyi S, Suchara M, Vargo A (2014) Efficient algorithms for maximum likelihood decoding in the surface code. Phys Rev A 90:032326. https://doi.org/10.1103/PhysRevA.90.032326

Chamberland C, Ronagh P (2018) Deep neural decoders for near term fault-tolerant experiments. Quantum Sci Technol 3(4):044002. http://stacks.iop.org/2058-9565/3/i=4/a=044002. Accessed 3 Mar 2020

Darmawan AS, Poulin D (2018) Linear-time general decoding algorithm for the surface code. Phys Rev E 97:051302. https://doi.org/10.1103/PhysRevE.97.051302

Davaasuren A, Suzuki Y, Fujii K, Koashi M (2018) General framework for constructing fast and near-optimal machine learning- based decoder of the topological stabilizer codes. arXiv:1801.04377

Dennis E, Kitaev A, Landahl A, Preskill J (2002) Topological quantum memory. J Math Phys 43(9):4452–4505. https://doi.org/10.1063/1.1499754

Devitt SJ, Munro WJ, Nemoto K (2013) Quantum error correction for beginners. Rep Prog Phys 76(7):076001. https://doi.org/10.1088/0034-4885/76/7/076001

Duclos-Cianci G, Poulin D (2010a) A renormalization group decoding algorithm for topological quantum codes. Information Theory Workshop (ITW), IEEE, p 1–5. https://doi.org/10.1109/CIG.2010.5592866

Duclos-Cianci G, Poulin D (2010b) Fast decoders for topological quantum codes. Phys Rev Lett 104:050504. https://doi.org/10.1103/PhysRevLett.104.050504

Edmonds J (1965) Paths, trees, and flowers. Can J Math 17:449–467. https://doi.org/10.4153/CJM-1965-045-4

Fowler A G (2013) Optimal complexity correction of correlated errors in the surface code. arXiv:1310.0863

Fowler AG (2015) Minimum weight perfect matching of fault tolerant topological quantum error correction in average o(1) parallel time. Quantum Inf Comput 15:145–158

Fowler AJ, Stephens AM, Groszkowski P (2009) High threshold universal quantum computation on the surface code. Phys Rev A 80: 052312. https://link.aps.org/doi/10.1103/PhysRevA.80.052312. Accessed 3 Mar 2020

Fowler AG, Mariantoni M, Martinis JM, Cleland AN (2012a) Surface codes: towards practical large-scale quantum computation. Phys Rev A 86: 032324. https://doi.org/10.1103/PhysRevA.86.032324

Fowler AG, Whiteside AC, Hollenberg LCL (2012b) Towards practical classical processing for the surface code. Phys Rev Lett 108:180501. https://doi.org/10.1103/PhysRevLett.108.180501

Freedman MH, Meyer DA (2001) Projective plane and planar quantum codes. Found Comput Math 1(3):325–332

Gottesman D (1997) Stabilizer codes and quantum error correction. Dissertation, Caltech

Herold M, Campbell E T, Eisert J, Kastoryano M J (2015) Cellular-automaton decoders for topological quantum memories. Npj Quantum Information 1. https://www.nature.com/articles/npjqi201510. Accessed 3 Mar 2020

Horsman C, Fowler AG, Devitt S, Meter RV (2012) Surface code quantum computing by lattice surgery. New J Phys 14(12):123011. https://doi.org/10.1088/1367-2630/14/12/123011

Hutter A, Wootton JR, Loss D (2014) Efficient Markov chain Monte Carlo algorithm for the surface code. Phys Rev A 89:022326. https://doi.org/10.1103/PhysRevA.89.022326

Kitaev A (2003) Fault-tolerant quantum computation by anyons. Ann Phys 303(1):2–30. http://www.sciencedirect.com/science/article/pii/S0003491602000180. Accessed 3 Mar 2020

Kolmogorov V (2009) Blossom V: a new implementation of a minimum cost perfect matching algorithm. Math Program Comput 1:43–67. https://doi.org/10.1007/s12532-009-0002-8

Krastanov S, Jiang L (2017) Deep neural network probabilistic decoder for stabilizer codes. Sci Rep 7:11003

Landahl A J, Anderson J T, Rice P R (2011) Fault-tolerant quantum computing with color codes. arXiv:1108.5738

Maskara M, Kubica A, Jochym-O'Connor T (2018) Advantages of versatile neural-network decoding for topological codes. arXiv:1802.08680

Ni X (2018) Neural network decoders for large-distance 2d toric codes. arXiv:1809.06640

Nielsen MA, Chuang IL (2002) Quantum computation and quantum information. Cambridge University Press, Cambridge

Raussendorf R, Harrington J (2007) Fault-tolerant quantum computation with high threshold in two dimensions. Phys Rev Lett 98:190504. https://doi.org/10.1103/PhysRevLett.98.190504

Raussendorf R, Harrington J, Goyal K (2007) Topological fault-tolerance in cluster state quantum computation. New J Phys 9(6):199–199. https://doi.org/10.1088/1367-2630/9/6./199

Suchara M, Bravyi S, Terhal B (2011) Constructions and noise threshold of topological subsystem codes. J Phys A Math Theor 44(15): 155301. http://stacks.iop.org/1751-8121/44/i=15/a=155301. Accessed 3 Mar 2020

Sweke R, Kesselring M S, van Nieuwenburg E P L, Eisert J (2018) Reinforcement learning decoders for fault-tolerant quantum computation. arXiv:1810.07207

Terhal BM (2015) Quantum error correction for quantum memories. Rev Mod Phys 87:307–346. https://doi.org/10.1103/RevModPhys.87.307

Torlai G, Melko RG (2017) Neural decoder for topological codes. Phys Rev Lett 119:030501 7. https://doi.org/10.1103/PhysRevLett.119.030501

Varsamopoulos S, Criger B, Bertels K (2017) Decoding small surface codes with feedforward neural networks. Quantum Sci Technol 3(1):015004. http://stacks.iop.org/2058-9565/3/i=1/a=015004. Accessed 3 Mar 2020

Varsamopoulos S, Bertels K, Almudever CG (2019) Designing neural network based decoders for surface codes. IEEE Trans Comput. https://doi.org/10.1109/TC.2019.2948612

Wang DS, Fowler AG, Hollenberg LCL (2011) Surface code quantum computing with error rates over 1%. Phys Rev A 83:020302. https://doi.org/10.1103/PhysRevA.83.020302

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.